IBM Surveillance Insight for Financial Services
Version 2.0.3

IBM Surveillance Insight for Financial Services Solution Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 117.

Product Information

This document applies to Version 2.0.3 and may also apply to subsequent releases.

Contents

Introduction

Use IBM® Surveillance Insight® for Financial Services to proactively detect, profile, and prioritize non-compliant behavior in financial organizations. The solution ingests structured and unstructured data, such as trade, electronic communication (emails, chats), and voice data, to flag risky behavior. Surveillance Insights helps you investigate sophisticated misconduct faster, by prioritizing alerts and reducing false positives, and reduce the cost of misconduct.

Some of the key problems that financial firms face in terms of compliance misconduct include:

- Fraudsters using sophisticated techniques making it hard to detect misconduct.
- Monitoring and profiling are hard to do proactively and efficiently with constantly changing regulatory compliance norms.
- A high rate of false positives increases the operational costs of alert management and investigations.
- Siloed solutions make fraud identification difficult and delayed.

IBM Surveillance Insight for Financial Services addresses these problems by:

- Leveraging key innovative technologies, such as behavior analysis and machine learning, to proactively identify abnormalities and potential misconduct without pre-defined rules.
- Using evidence-based reasoning that aids streamlined investigations.
- Using risk-based alerting that reduces false positives and negatives and improves the efficiency of investigations.
- Combining structured and unstructured data from different siloed systems into a single platform to perform analytics.

IBM Surveillance Insight for Financial Services takes a holistic approach to risk detection and reporting. Surveillance Insight combines structured data such as stock market data (trade data) with unstructured data such as emails and voice data, and it uses this data to perform behavior analysis and anomaly detection by using machine learning and natural language processing.

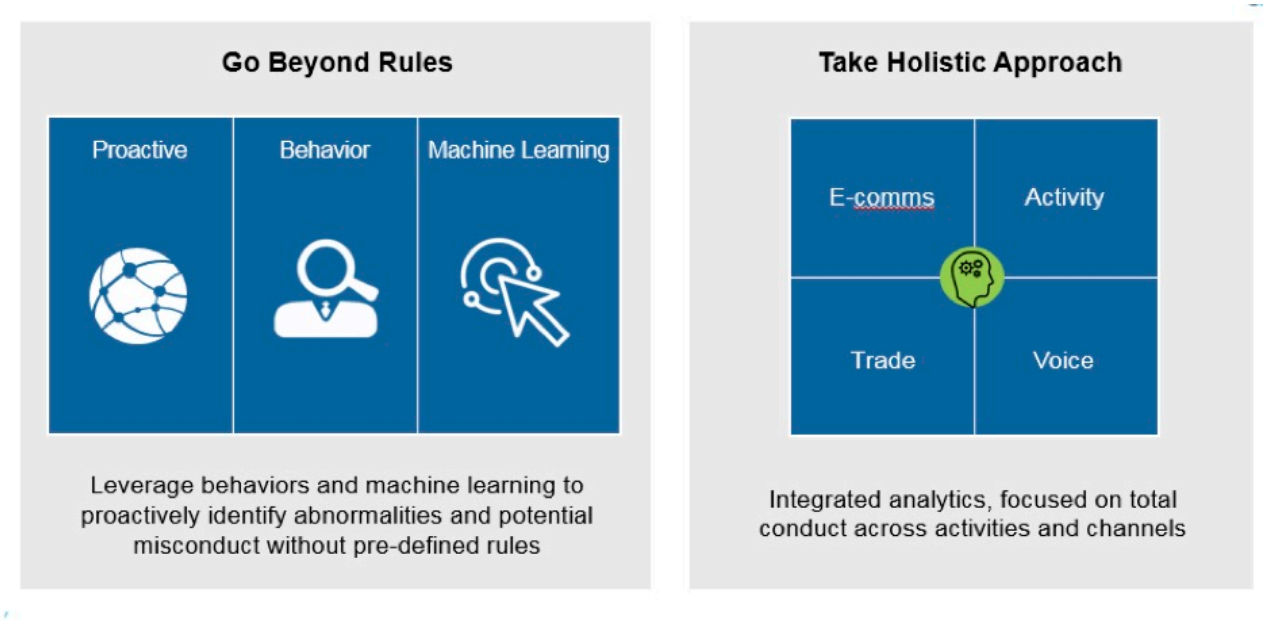


Figure 1: Surveillance Insight overview

Audience

This guide is intended for administrators and users of the IBM Surveillance Insight for Financial Services solution. It provides information on installation and configuration of the solution, and information about using the solution.

Finding information and getting help

To find product documentation on the web, access IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSWTQQ).

Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products. Some of the components included in the IBM Surveillance Insight for Financial Services have accessibility features. For more information, see Accessibility features.

The HTML documentation has accessibility features. PDF documents are supplemental and as such, include no added accessibility features.

Forward-looking statements

This documentation describes the current functionality of the product. References to items that are not currently available may be included. No implication of any future availability should be inferred. Any such references are not a commitment, promise, or legal obligation to deliver any material, code, or functionality. The development, release, and timing of features or functionality remain at the sole discretion of IBM.

Samples disclaimer

Sample files may contain fictional data manually or machine generated, factual data that is compiled from academic or public sources, or data that is used with permission of the copyright holder, for use as sample data to develop sample applications. Product names that are referenced may be the trademarks of their respective owners. Unauthorized duplication is prohibited.

Chapter 1. IBM Surveillance Insight for Financial Services

IBM Surveillance Insight for Financial Services provides you with the capabilities to meet regulatory obligations by proactively monitoring vast volumes of data for incriminating evidence of rogue trading or other wrong-doing through a cognitive and holistic solution for monitoring all trading-related activities. The solution improves current surveillance process results and delivers greater efficiency and accuracy to bring the power of cognitive analysis to the financial services industry.

The following diagram shows the high-level IBM Surveillance Insight for Financial Services process.

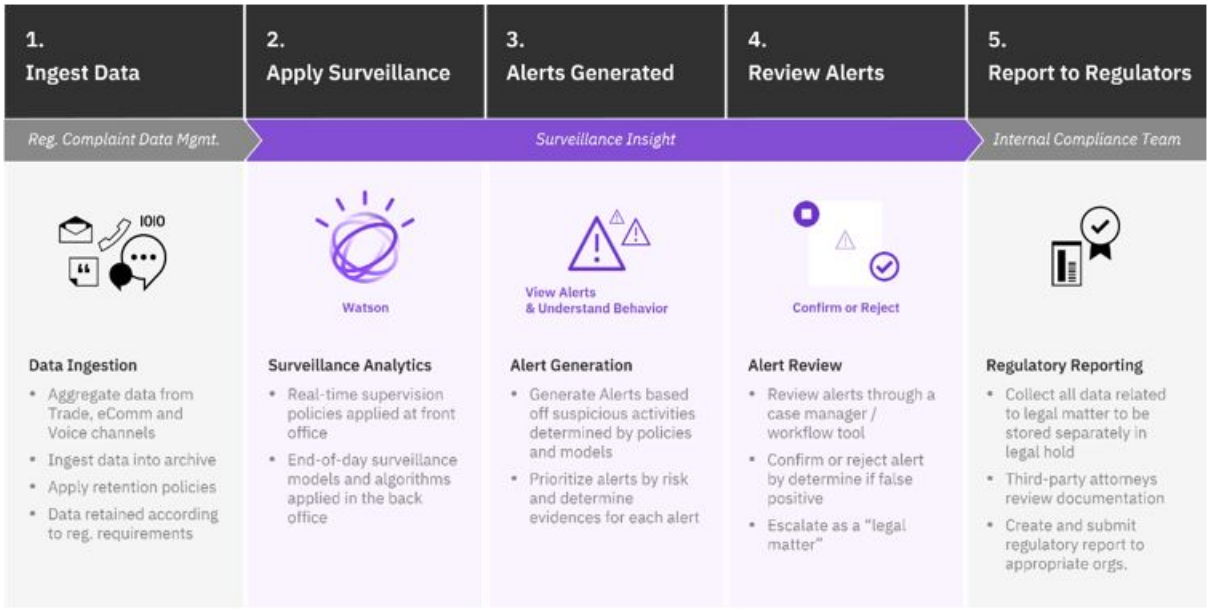


Figure 2: High-level process

1. As a first step in the process, data from electronic communications (such as email and chat), voice data, and structured stock market data are ingested into IBM Surveillance Insight for Financial Services.
2. The data is analyzed.
3. The results of the analysis are risk indicators with specific scores.
4. The evidences and their scores are used by the inference engine to generate a consolidated score. This score indicates whether an alert needs to be created for the current set of risk evidences. If needed, an alert is generated and associated with the related parties and stock market tickers.
5. The alerts and the related evidences that are collected as part of the analysis can be viewed in the IBM Surveillance Insight for Financial Services Workbench.

After the alerts are created and the evidences are collected, the remaining steps in the process are completed outside of IBM Surveillance Insight for Financial Services. For example, case investigators must work on the alerts and confirm or reject them, and then investigation reports must be sent out to the regulatory bodies as is required by compliance norms.

The solution architecture

IBM Surveillance Insight for Financial Services is a layered architecture that is made up of several components.

The following diagram shows the different layers that make up the product:

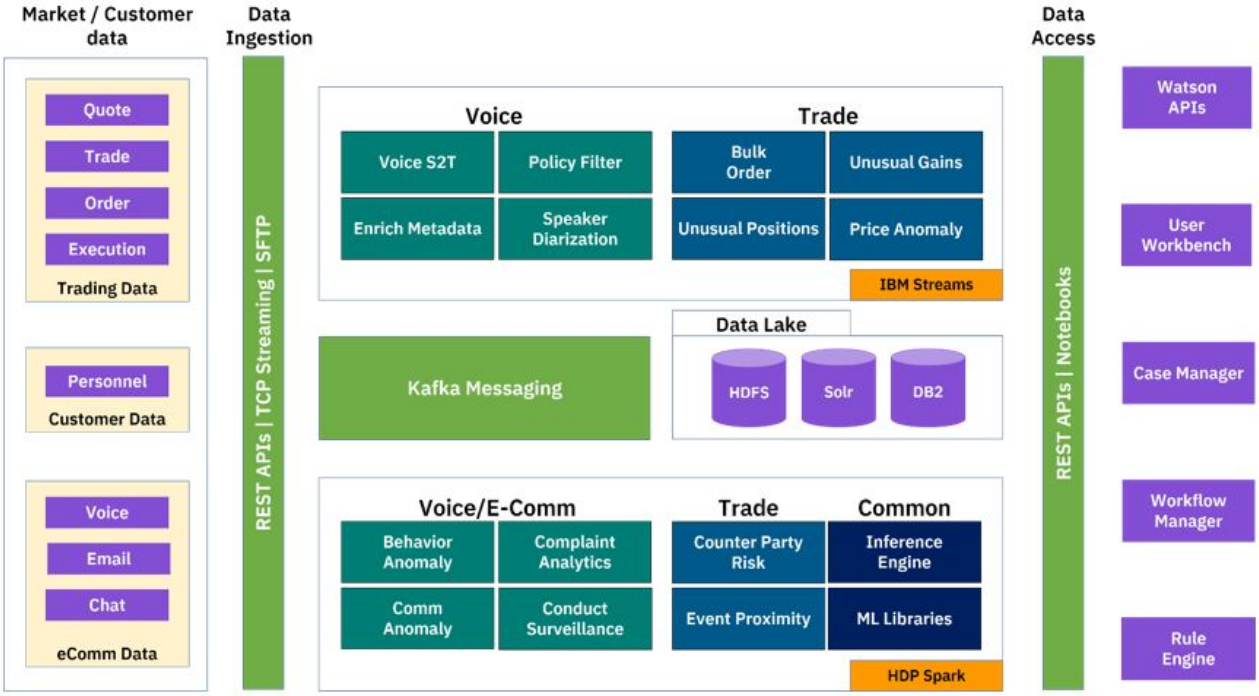


Figure 3: Product layers

- The data layer shows the various types of structured and unstructured data that is consumed by the product.
- The data ingestion layer contains the FTP/TCP-based adaptor that is used to load data into Hadoop. The Kafka messaging system and REST APIs are used for loading e-communications into the system.

  Note: IBM Surveillance Insight for Financial Services does not provide the adaptors with the product.
- The analytics layer contains the following components:
  - Specific use case implementations that leverage the base toolkit operators.
  - Speech 2 Text and Speaker diarization APIs are used in Voice surveillance.
  - Unusual gain, bulk order, and other operators are used for Trade surveillance.
  - NLP APIs are used for e-comm surveillance.
  - The Spark Streaming API is used by Spark jobs as part of the use case implementations.
  - Watson NLC/NLU APIs are used to perform Complaint analytics.
  - A surveillance library that contains the common components that provide core platform capabilities such as alert management, reasoning, and the policy engine.
- Apache Kafka is used as an integration component in the use case implementations and to enable asynchronous communication between the Streams jobs and the Spark jobs.
  - The data layer primarily consists of data in Apache Hadoop, Apache Solr and IBM Db2®.
  - The day-to-day market data is stored in Hadoop. It is accessed by using the spark-sql APIs.
  - Solr is used to index content to enable search capabilities in the workbench.

- Data in Db2 is accessed by using traditional relational SQL. REST services are provided for data that needs to be accessed by the user interfaces and for certain operations, such as alert management.
- Surveillance Insight provides the capability to detect risk from different types of data (trade, e-comm, and voice) and report to a case manager system so that further investigation can be carried out.
- User workbenches like Surveillance Dashboard and Complaints Dashboard are provided to view the results of the analysis.

The deployment architecture

IBM Surveillance Insight for Financial Services is a multi-node architecture where different nodes host different parts of the solution.

The following diagram shows a high-level overview of the deployment architecture.

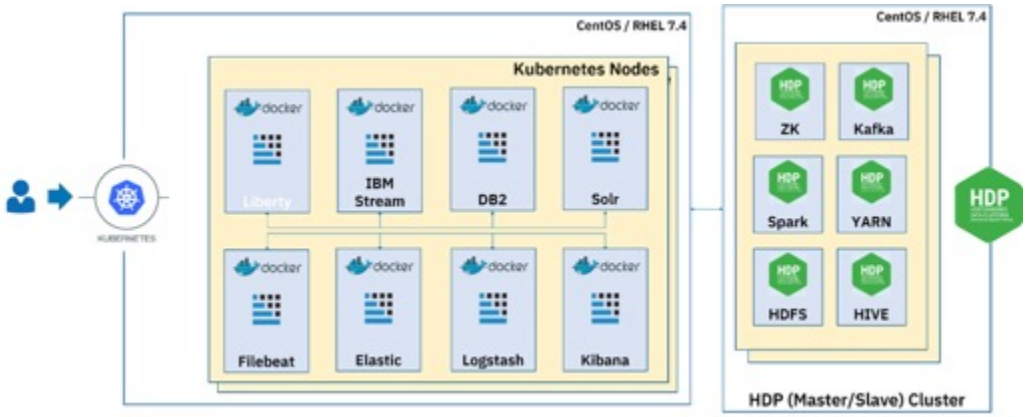


Figure 4: Deployment architecture

IBM Surveillance Insight for Financial Services supports either CentOS v7.4 or RHEL v7.4 operating systems.

IBM Surveillance Insight for Financial Services supports a small (3-node) or medium (6-node) topology for the Hortonworks Data Platform (HDP) installation. HDP provides big data technologies, such as HDFS, Spark, and Kafka.

Apart from HDP, all of the other components are dockerized and are installed and managed by using Kubernetes. The Kubernetes environment can be installed on one node and any configuration up to five nodes.

The security architecture

IBM Surveillance Insight for Financial Services secures data in-motion or at-rest.

The following diagram shows a high-level overview of the security architecture.

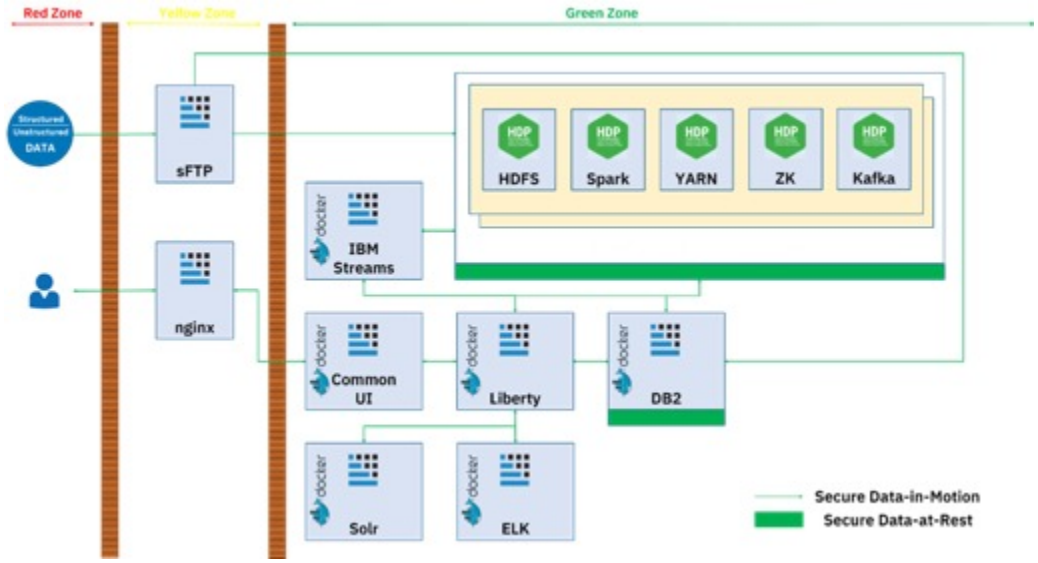


Figure 5: Security architecture

Data is transferred on a TLSv1.2 secured channel within the different components or outside the solution. An AES algorithm is used for encryption when data is stored in Db2 and HDFS

The components

IBM Surveillance Insight for Financial Services includes a base layer of components, and then four components cater to different data channels.

The components are:

- IBM Surveillance Insight for Financial Services (base)
- IBM Electronic Communication Surveillance Analytics
- IBM Trade Surveillance Analytics
- IBM Voice Surveillance Analytics
- IBM Complaints Analytics

The IBM Surveillance Insight for Financial Services (base) component is a prerequisite for any of the other channel-related components. The IBM Electronic Communication Surveillance Analytics component is a prerequisite for IBM Voice Surveillance Analytics or IBM Trade Surveillance Analytics.

Chapter 2. IBM Surveillance Insight data schemas

IBM Surveillance Insight for Financial Services processes data only on consent of the customer. The solution stores and retains all of the data in the system for an agreed upon time with a customer. Backup and archiving are implemented according to customer requirements. Access to the data is based on customer engagement.

The following tables show the different types of data that IBM Surveillance Insight for Financial Services uses to derive insights.

Customer data

Table 1: Customer data

Data type	Data name	Purpose
Contact information	Names of individuals Home or business address Other geographic information, such as postcode or zipcode Email address, Phone number, Chat address	Used to tag an individual based on insights derived from the communication and market data.
Other Personal Identifier	Age	Used for displaying trends based on age groups in the dashboard.
Technical identifiers	Device identifiers, including device IDs and serial numbers User IDs and login names IP addresses	Used to tag an individual based on insights derived from the communication data. IDs and login names are used for authentication and authorization of individuals. IP addresses are used for auditing.

On request, an individual's information can be removed. For more information, see Chapter 9, "GDPR compliance," on page 113.

Communication data

Table 2: Communication data

Data type	Data name	Purpose
Usage-based identifiers	Person-to-person messaging like email, chat and voice calls	Used for performing analysis to detect possible fraud.

Transactional data

Transactional data, such as trade, quote, execution, order, and ticker data, is collected and used for analyzing possible fraud activities such as pump and dump, off market, and so on.

The Db2 data schema

IBM Surveillance Insight for Financial Services stores the major entities of business domain in Db2 database schema.

The major entities are party, communication, alert, trade, model, and complaint.

Party view

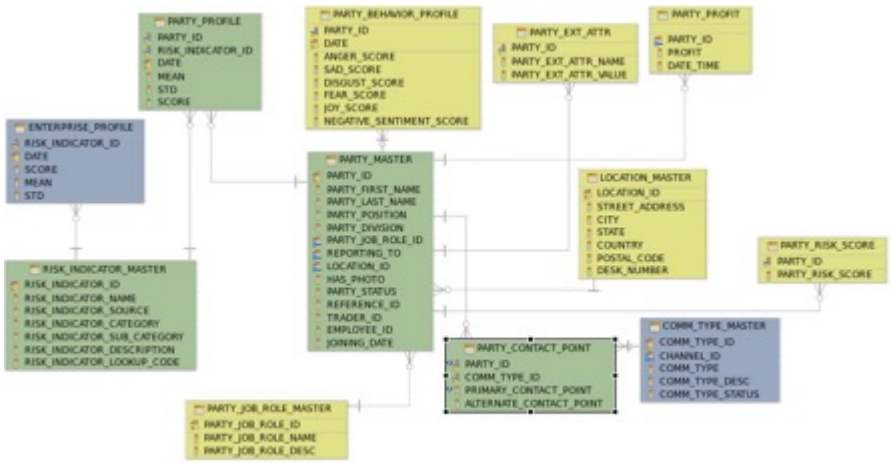


Figure 6: Party view schema

Table name	Description	Populated by
Party_Master	Stores basic party details. Reference ID refers to id in the core systems. TraderId contains mapping to the id that the party uses for trading.	Populated during initial data load and periodically kept in sync with the customer's core master data system.
Party Risk Score	Stores overall risk score for the party based on the party's current and past alert history.	Scheduled Surveillance Insight job that runs daily.
Party Behavior Profile	Stores date-wise scores for various behavior parameters of the party.	Surveillance Insight Ecomm Job daily.
Party Profile	Stores date wise, risk indicator wise statistics for every party.	Surveillance Insight Ecomm Job that updates count field for every communication that is analyzed. Updates the other fields on a configurable frequency.

Table name	Description	Populated by
Enterprise Profile	This table maintains the Mean and Std Deviation for each date and Risk Indicator combination. The mean and standard deviation is for the population. For example, the values are computed using data for all parties.	This table is populated by a Spark job that is based on the frequency at which the Spark job is run. The job reads the date parameter and for that date, populates the Enterprise profile table. Surveillance Insight expects to populate the data in the Enterprise profile only one time for a specific date.
Party Contact Point	Contains email, voice, and chat contact information for each party.	Populated during the initial data load and periodically kept in sync with the customer's core master data system.
Party Profit	Profit made by the party from trades.	Populated by the spoofing use case implementation.
Party Job Role Master	Master table for party job roles.	Populated during the initial data load and periodically kept in sync with the customer's core master data system.
Comm Type Master	Master table for communication types such email, voice, and chat.	Populated during the initial data load and periodically kept in sync with the customer's core master data system.
Location Master	Master table with location details of the parties.	Populated during the initial data load and periodically kept in sync with the customer's core master data system.
Party_Ext Attr	Table to allow extension of party attributes that are not already available in the schema.	Populated during implementation. Not used by the provided use cases.
Risk_Indicator_Master	Master table with all of the risk indicators	Populated during initial load with a list of risk indicators that were detected by Surveillance Insight. New risk indicators can be created through the Design Studio **Risk Model** tab.

Communication view

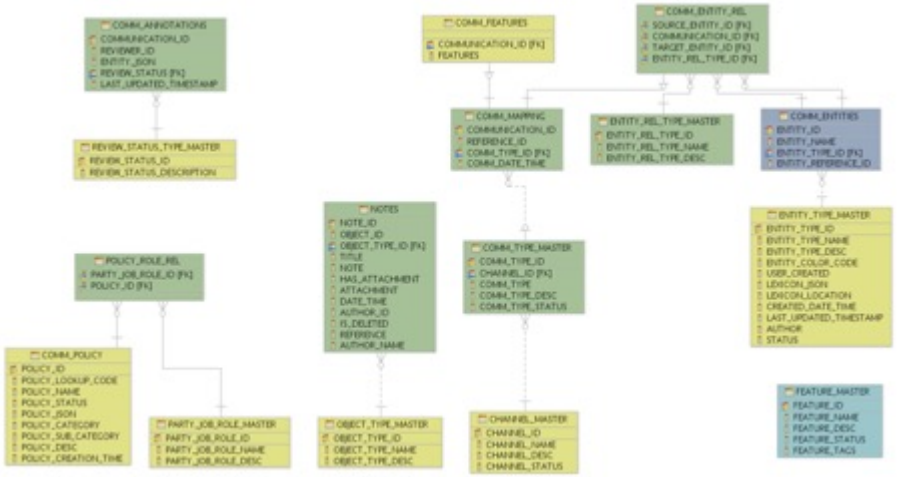


Figure 7: Communication view schema

Table name	Description	Populated by
Comm Entities	Stores entities that are extracted from electronic communications.	Populated by the SIFS e-comm components during e-comm analysis.
Comm Entities Rel	Stores relationships between entities that are extracted from the electronic communications.	Populated by the SIFS e-comm components during e-comm analysis.
Comm Policy	This table maintains the policy details registered in Surveillance Insight. The table has data for both system and role level policies.	Populated through the Policy REST service. The service supports create, update, activate, and deactivate features.
Policy Role Rel	This table maintains the policy to role mapping. For role level policies, the relationship for policy and job role is stored in this table.	Populated when the policy is created in the system by using the REST service. Updates to this table are not supported. It is recommended to create a new policy if there any changes in role.
Feature Master	This table contains a master list of all of the features that are supported by Surveillance Insight for Financial Services.	Master table. Populated during Surveillance Insight product setup.
Comm Type Master	Master table that stores the different communication types, such as voice, email, and chat.	Populated with the supported communication types during product installation.

Table name	Description	Populated by
Comm Feature	This table contains the feature JSON for each communication that is processed by Surveillance Insight. The CORE_FEATURES_JSON column contains the JSON for all of the features in the Feature Master. For metadata, the JSON is stored in the META_DATA_FEATURES_JSON column. The table also provides a provision to store custom feature values in the CUSTOM_FEATURES_JSON column.	This table is populated for every communication that is processed by Surveillance Insight for Financial Services.
Channel Master	Master table that stores the different communication channels.	Populated with the supported channel types during product installation. The channels are e-comm and voice.
Entity Type Master	Master table for the type of entities that are extracted from the electronic communications.	Populated with the supported types during product installation. The types are people, organization, and ticker.
Entity Rel Type Master	Master table for types of relationships that are possible between entities that are extracted from the electronic communications.	Populated with the supported types during product installation. The types are Mentions and Works For.
NOTES	Stores all of the notes that are created by the case investigators for the alerts.	Surveillance Insight Note service populates this table when triggered from the Surveillance Insight front-end.
OBJECT_TYPE_MASTER	Master table for storing different type of objects against which notes are added.	The table is populated by the communication and alert objects.
REVIEW_STATUS_TYPE_MASTER	Master table for storing the different types of review statuses for a voice communication.	The table is populated by the in-progress and completed statuses.
Comm_Annotations	Table to store annotations detected/annotated for a .	This table stores all of the annotations that are detected by the solution for ecomm and complaints data and also stores the updates made by user to add/ remove annotations.
Comm_Mapping	Table to store communication metadata details.	Stores communication details like communication type, communication id, source reference id, and communication date and time.

Alert view

Figure 8: Alert view schema

Table name	Description	Populated by
Alert	Core table that stores the alert data.	This table is populated by any use case that creates an alert. The createAlert REST service must be used to populate this table.
Risk Evidence	Core table that stores each of the risk evidences that are identified during the data analysis.	This table is populated by any use case that creates risk evidences. The createEvidence REST service can be used to populate this table.
Alert Evidence Rel	Links an alert to multiple evidences and evidences to alerts.	This table is populated by any use case that creates an alert. The createAlert REST service must be used to populate this table.
Alert Involved Party Rel	Links the parties who are involved in an alert with the alert itself.	This table is populated by any use case that creates an alert. The createAlert REST service must be used to populate this table.
Alert Risk Indicator Score	Identifies the risk indicators and corresponding scores that are associated with an alert.	This table is populated by any use case that creates an alert. The createAlert REST service must be used to populate this table.
Alert Ticker Rel	Links the tickers that are associated with an alert to the alert itself.	This table is populated by any use case that creates an alert. The createAlert REST service must be used to populate this table.
NOTES	Stores all of the notes that are created by the case investigators for the alerts.	Surveillance Insight Note service populates this table when triggered from the Surveillance Insight front-end.

Table name	Description	Populated by
OBJECT_TYPE_MASTER	Master table for storing different type of objects against which notes are added.	The table is populated by the communication and alert objects.
Evidence Involved Party Rel	Links the parties that are involved in a risk evidence to the evidence itself.	This table is populated by any use case that creates risk evidences. The createEvidence REST service can be used to populate this table.
Evidence Ticker Rel	Links any tickers that are associated with an evidence to the evidences itself.	This table is populated by any use case that creates risk evidences. The createEvidence REST service can be used to populate this table.
Alert Source Master	Master table for source systems that can generate alerts.	Populated with one alert source for Surveillance Insight during product installation. More sources can be created, depending on the requirements.
Risk Indicator Master	Master table for risk indicators that are generated by the use cases.	Populated with certain indicators for e-comm and trade risk during product installation. More indicators can be created, depending on the requirements.
Risk Evidence Type Master	Master table for evidence types, such as trade, email, and voice.	Populated with a certain type during product installation. More types can be created, depending on the requirements.
Risk Model Master	Master table for the risk models that are used for generating the reasoning graph.	Populated during the product installation with following models: pump-and-dump, spoofing, and party risk behavior. More models can be populated, depending on the requirements of new use cases.
Risk Model Type Master	Master table for the types of risk models that are supported.	Populated during the product installation with rule and Bayesian network types.
Alert Ext Attr	This table allows for the extension of alert attributes.	Not used by Surveillance Insight for Financial Services. This table is meant for customer implementations, if required.

Trade view

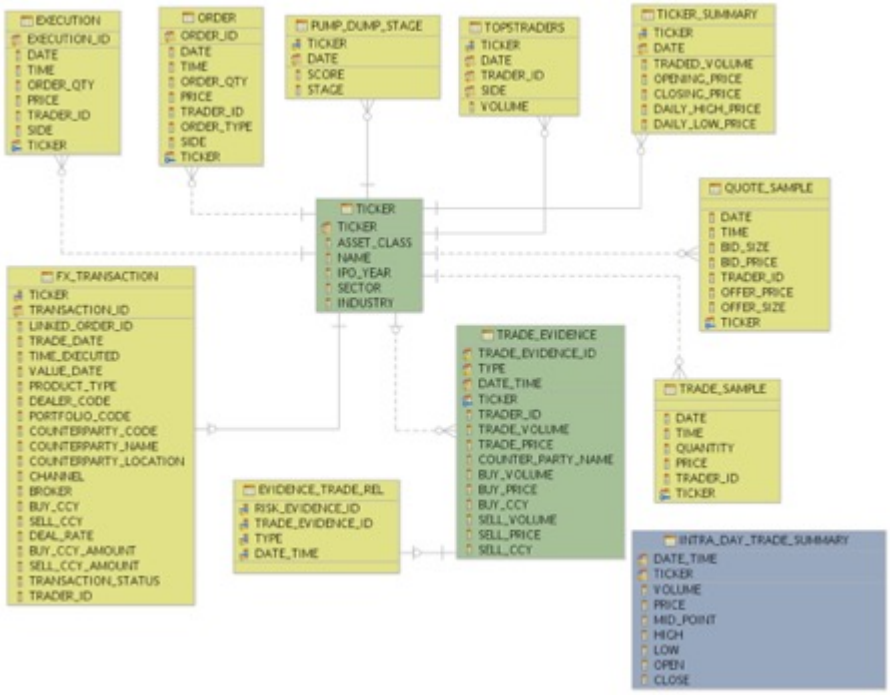


Figure 9: Trade view schema

Table name	Description	Populated by
Ticker	Master table that stores basic ticker information.	This table is populated during the initial data load and whenever new tickers are found in the trade data.
Trade Sample	This table contains samples of trade data from the market. The trades that go into this table depend on the specific use case. The trades are primarily evidences of some trade risk that is detected. Typically, these trades are sampled from the market data that is stored in Hadoop. They are stored here for easy access by the user interface layer.	Currently, the spoofing and pump-and-dump use cases populate this table whenever a spoofing alert is identified.
Trade Evidence	This table contains evidences for specific type like order, trade, execution etc.,. that needs to be shown in the trade charts for trade use cases. This table, along with the evidence_trade_ rel table allow linking the evdences to the alerts through the evidence ids	This table is populated by the Trade Evidence Persistence Spark job as and when it receives the risk evidence from the use case implementation.

Table name	Description	Populated by
Evidence Trade Rel	This table links the trade evidences to the risk evidences and thus indirectly link them to the alert. This table helps the UI services figure out what trade evidences are relevant for a specific alert.	This table is populated by the Trade Evidence Persistence Spark job as and when it receives the risk evidence from the use case implementation.
FX Transaction	This table contains the transaction details for forex transactions. This table is synonymous to the Trade Evidence table except that it has additional fields that are specific to forex.	This table is populated by the Trade Evidence Persistence Spark job as and when it receives the risk evidence from the use case implementation.
Quote Sample	This table contains samples of quote data for certain durations of time. The quotes that go into this table depend on the specific use case. The quotes are evidences of some kind of risk that is identified by the specific use case. These quotes are sampled from the market data and stored in Hadoop. They are stored in this table for easy access by the user interface layer.	Currently, the spoofing use case populates this table whenever a spoofing alert is created. The sampled quote is the max (bid price) and min (offer price) for every time second.
Order	This table contains orders that need to be displayed as evidence for some alert in the front end. The contents are copied from the market data in Hadoop. The specific dates for which the data is populated depends on the alert.	Currently, the spoofing use case populates this table whenever a spoofing alert is identified.
Execution	This table contains orders that need to be displayed as evidence for some alert in the front end. The contents are copied from the market data in Hadoop. The specific dates for which the data is populated depends on the alert.	Currently, the spoofing use case populates this table whenever a spoofing alert is identified.
Pump Dump Stage	This table contains the pump-and-dump stage for each ticker that shows pump or dump evidence.	This table is populated by the pump-and-dump use case implementation.
Top5Traders	This table contains the top five traders for buy and sell sides for each ticker that shows pump or dump evidence. This table is populated daily.	This table is populated by the pump-and-dump use case implementation.

Table name	Description	Populated by
Intra Day Trade Summary	This table is meant to be used for the Trade charts in the trade use cases. This table contains summary data for trade/ transaction data on a daily basis.	This table is populated by the Trade Evidence Persistence Spark job as and when it receives the risk evidence from the use case implementation.
Ticker Summary	This table is meant to store ticker summary details.	This table is populated with ticker summary details.

Model view

The following diagram shows the logical data model of the Voice Language Model and Discovery Model tables in the SIFS database.

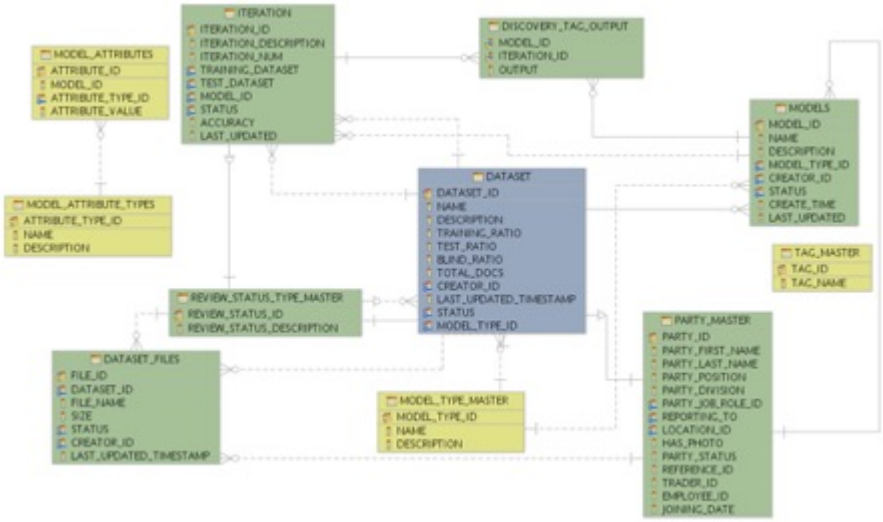


Figure 10: Model view

Table 3: Model view		
Table name	**Description**	**Populated by**
Dataset	This table contains datasets for training or testing of Voice language and Discovery Models	Populated by Voice Language Model tool or Discovery Model use case using DatasetService Rest Service
Dataset Files	This table holds the reference to the file name used for training or testing	Populated by Voice Language Model tool or Discovery Model use case using DatasetService Rest Service
Discovery Tag Output	This table holds the output tagged during discovery process	Populated by Discovery Model tool when user manually tags terms against lexicons
Iteration	This table tracks testing Iteration and its results	Populated by Voice Language Model tool or Discovery Model use case using IterationService Rest Service

Table 3: Model view (continued)		
Table name	**Description**	**Populated by**
Model Type Master	This table is the Master table for different Models	Populated by Voice Language Model tool or Discovery Model use case using ModelService Rest Service
Models	This table contains the different Voice language or Discovery Models detail	Populated by Voice Language Model tool or Discovery Model use case using ModelService Rest Service
Review Status Type Master	This is the master table for capturing the status of Iterations, Datasets and Models	Populated by Voice Language Model tool or Discovery Model use case using Dataset, Iteration or Model Rest Services
Tag master	This is master table with list of tags detected by the solution	Populated as part of initial data load
Model Attribute	This table contains custom attributes and its values for a model	Populated by Voice Language Model tool or Discovery Model use case using ModelService Rest Service
Model Attribute Types	This is the master table which defines custom attributes for a model	Populated by Voice Language Model tool or Discovery Model use case using ModelService Rest Service

Complaints view

The following diagram shows the logical data model of the Complaints tables in the SIFS database.

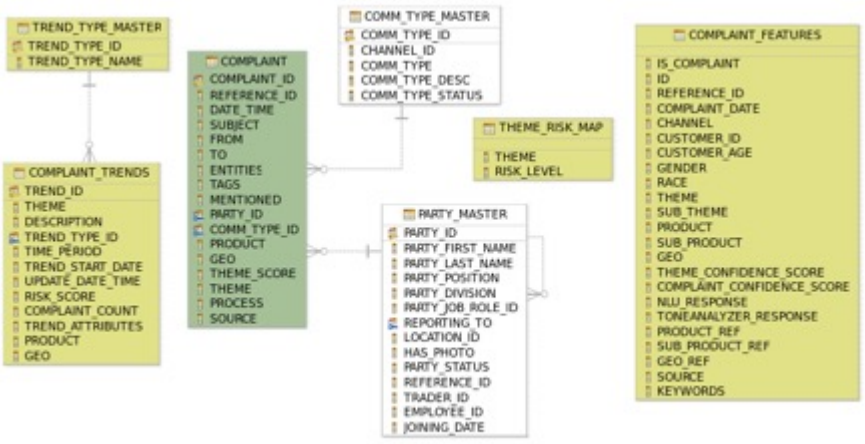


Figure 11: Complaints view

Table 4: Complaints view		
Table name	**Description**	**Populated by**
Complaint	This table contains the details of each complaint that is processed by the pipeline. The table does not contain the actual complaint text. The text is saved in Solr to enable search through the dashboard. The table contains the results from models that processed the complaint (THEME.THEME_SCORE, TAGS, ENTITIES) and some basic information that is obtained from the raw data	Populated by Complaints pipeline job
Complaint Features	This table contains all the features detected as part of Complaints pipeline	Populated by Complaints pipeline job
Complaint Trends	The COMPLAINT_TRENDS table contains the trends that are identified in the complaints and the trend attributes that were used to compute the trend	Populated by Complaints Trend detection job
Trend Type Master	The TREND_TYPE_MASTER table contains the master data for the trend direction (upwards, downwards, neutral)	Populated as part of initial data load
Theme Risk Map	The THEME_RISK_MAP table contains the master data for supported themes and their risk level	Populated as part of initial data load

Solr data schema

Search capability for IBM Surveillance Insight for Financial Services is supported by Apache Solr.

Indexing is performed on the content of communication data, such as email, chat, or voice messages, and on the analytical results. Users can use keywords or attributes for searching content within the communication data or analytical results.

IBM Surveillance Insight for Financial Services have two core repositories in Solr, namely sifs and complaints.

Schema structure of sifs

The core sifs is used to store and index data for e-mail communication, voice communication, and trade.

Indexing is performed on three types of objects:

- Alert
- E-comm (includes email, chat, and voice)
- Employee

Employee data is indexed as a one-time initial load. Currently, updates to employee data are not relayed into the indexing system.

E-comm content and the analytic results are indexed.

The email body, collated chat content, and all of the utterances of voice speech are indexed.

When an alert is created, certain attributes of the alert are also indexed. Updates to an alert, such as adding new evidence to an existing alert is also updated in Solr.

The global search feature is sourced from the indexing system.

All of the Solr fields have been made "Stored" as "true". This enables the actual value of the field to be retrieved by queries.

Table 5: Schema structure of sifs

Field name	Field type	Is indexed?	Is multivalued?	Description
commid	String	yes	no	Unique id to identify a communication
commstarttime	pdate	yes	no	Time when communication started
commendtime	pdate	yes	no	Time when communication is completed
commtype	String	yes	no	Possible values are e-mail, chat, phone, ipaddress, loginname
commsubject	text_general	yes	no	Subject of the communication
commtext	text_general	no	no	Content of the communication e.g the e-mail body in case of e-comm
sourcereferenceid	String	yes	no	Unique id to identify the communication at source e.g for voice communication it is “global call id” or “gcid”
comminitiatorname	text_general	yes	no	Name of the communication initiator
comminitiatorid	Sting	yes	no	Id of communication initiator
comminitiatorcontact	text_general	yes	no	e-mail or phone number of the communication initiator

Table 5: Schema structure of sifs (continued)				
Field name	**Field type**	**Is indexed?**	**Is multivalued?**	**Description**
commparticipantsid	String	yes	yes	list of ids of participants in a communication
commparticipantsname	text_general	yes	yes	list of names of participants in a communication
commparticipantscontact	text_general	yes	yes	list of contacts of participants in a communication
commtags	String	yes	yes	Tags attached to a communication like “joy”, “positive”. The tags are attached to a communication either by machine learning model or by end user through UI
commdescription	text_general	yes	no	Description of the communication
commchannel	text_general	yes	no	The channel through which communication came e.g “voice”, “e-comm”
calldirection	String	yes	no	Calldirection values can be “initiated” or “received”. Initiated: A voice call is being originated from an employee to an external entity. Received: A voice call is received by an employee from external entity.
callduration	pint	yes	no	Duration of the voice call
deviceid	String	yes	yes	Stores the device ids used for communication e.g. device ids used for voice communication

Table 5: Schema structure of sifs (continued)

Field name	Field type	Is indexed?	Is multivalued?	Description
personalextension	String	yes	no	Store personal extension number, used by voice PCAP
phonenumber	String	yes	yes	Store phone number, used by voice PCAP
*_score	pfloat	yes	no	Various scores identified by the machine learning services e.g anger_score, disgust_score, sad_score, joy_score
*_keywords	text_general	yes	yes	Various keywords identified by the machine learning services e.g all_keywords, joy_keywords, fear_keywords, sad_keywords
classification	String	yes	yes	Classify the document as “confidential” or “non-confidential”
entity	text_general	yes	yes	Entity identified by machine learning services from a communication
evidenceids	text_general	yes	yes	Id of evidences attached to an alert communication
doctype	text_general	yes	no	Solr documentation type. The values can be comm, party, alert
alertid	text_general	yes	yes	Id to identify an alert uniquely
alerttype	text_general	yes	no	Different type of alerts: “Party behavior Risk”, “Spoofing”, “Pump Dump”, “Off-Market”

Table 5: Schema structure of sifs (continued)				
Field name	**Field type**	**Is indexed?**	**Is multivalued?**	**Description**
alertstatus	text_general	yes	no	Alert status can be one of the possible values: "New", "Closed", "In Progress", "Automatically Escalated", "Manually Escalated"
assetclass	text_general	yes	no	Persist assetclass values e.g. “FX”, “equity”
ticker	text_general	yes	no	Tickers attached to an alert e.g NPDZ, PDZ, EuroUSD
participants	text_general	yes	yes	Id of Participants in a communication
tradernames	text_general	yes	yes	The trader/party/ employee against which the alert is generated
alertevidenceids	text_general	yes	yes	Evidence ids helps to generate the alert
id	text_general	yes	no	Unique id used by Solr to identify a document
activealerts	pint	yes	no	No of active alerts e.g the alerts which are not in closed state
employeeid	string	yes	no	Id of the employee when the solr doctype is a “party”
description	string	yes	no	
name	text_general	yes	no	Name of the employee
partyid	String	yes	no	Id of the party when the Solr doctype is a “party”.
pastviolations	pint	yes	no	No of alerts against a party which are closed
city	text_general	yes	no	Party’s city details

Table 5: Schema structure of sifs (continued)				
Field name	**Field type**	**Is indexed?**	**Is multivalued?**	**Description**
state	text_general	yes	no	Party's state details
role	text_general	yes	no	Role played by party in the organization
riskrating	pfloat	yes	no	Numeric value to quantify risk of a party to organization
text	copyField	copyField	copyField	This special field stores a copy of the content of all the above fields. This makes it easier for searching the Solr content

Schema structure of complaints

The core complaint is a dynamic schema that is used to store and index data for Complaints analytics. The Complaints Explore search feature is sourced from the indexing system.

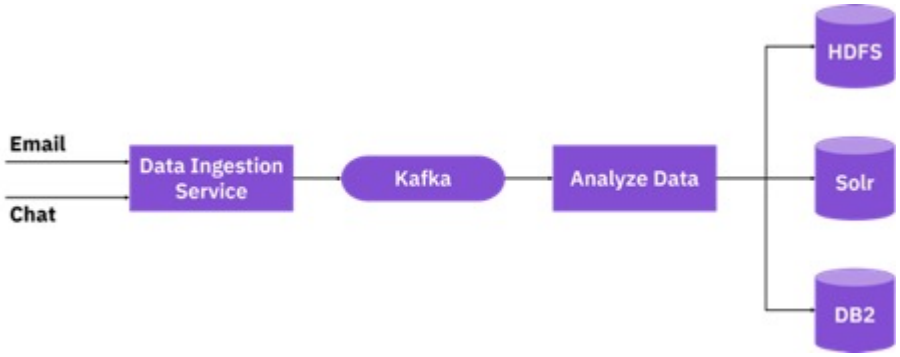
Table 6: Schema structure of complaints				
Field name	**Field type**	**Is indexed?**	**Is multivalued?**	**Description**
age_group_s	String	yes	no	Age group of customer. For example, lt_30 means less than 30.
category_tm	String	yes	yes	This is an array of space separated Complaint Category and Process
channel_s	String	yes	no	Channel id of channels like web, cfpb, email, etc., from where complaints are derived
complaint_date_dt	pdate	no	no	Date when a complaint is created
complaint_text_s	String	yes	no	General Complaint text for a given complaint

Table 6: Schema structure of complaints (continued)				
Field name	**Field type**	**Is indexed?**	**Is multivalued?**	**Description**
customer_age_i	Int	no	no	Age of customer who sent complaints
customer_id_s	String	yes	no	Unique id to identify the customer id
geo_ref_s	String	yes	no	Geography location of complaint received
id	String	yes	no	Unique id used by Solr to identify a document
id_s	text_general	yes	no	Unique id used for distinguishing complaint id
iscomplaint_s	String	yes	no	Classifies whether it's a complaint (true) or non-complaint (false) by Boolean value
keyword_tm	String	yes	yes	List of array of keywords that are annotated for complaints category and process
product_ref_s	String	yes	no	Product name of complaints related to
reference_id_s	String	yes	no	Unique id of complaint data from the source

Chapter 3. IBM Electronic Communication Surveillance Analytics

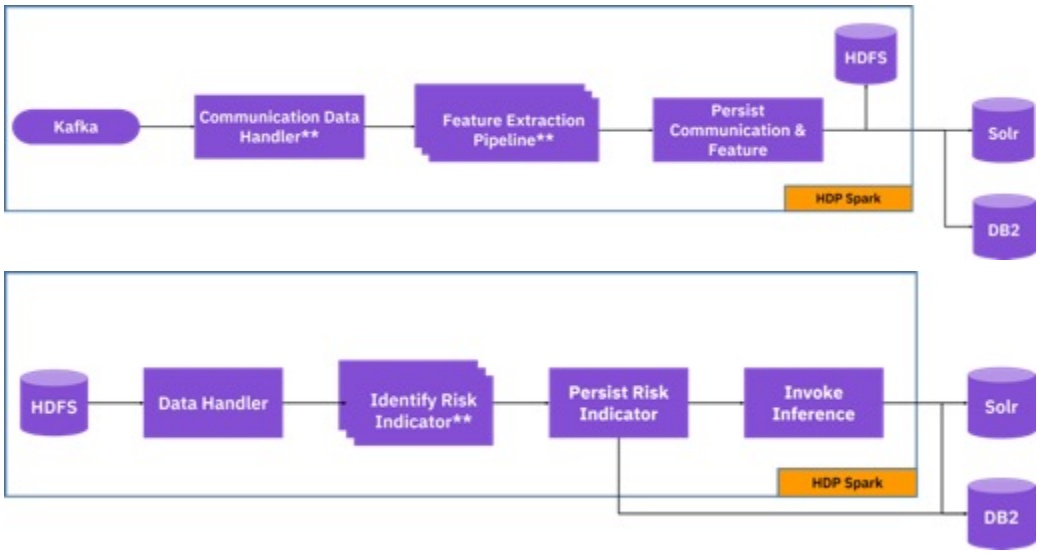
IBM Electronic Communication Surveillance Analytics processes unstructured data such as chat, email, and voice data. Data is evaluated against various features, based on pre-defined policies, and risk indicators are computed. These risk indicators are then analysed by the inference engine to detect alarming conditions to generate alerts.

The following diagram shows the end-to-end flow for IBM Electronic Communication Surveillance Analytics component.



E-comm data is processed further to extract features and detect risk. Data is persisted in HDFS, Solr, and Db2. The persisted data can be used later for further analysis and monitoring.

- The data schema for Db2 is explained in “The Db2 data schema” on page 6. Communication meta-data, derived data such as extracted feature, detected risk indicators, and generated alerts information is stored in Db2.
- The data schema for Solr is explained in “Solr data schema” on page 16. E-comm data is persisted in Solr to enable searching capabilities.
- E-comm extracted features are stored in HDFS so that risk can be detected from those extracted features. Risk is detected at the end of each day for all of the data, whereas features are extracted and stored in HDFS when the e-comm data is ingested.



IBM Electronic Communication Surveillance provides some pre-built components for Identify Risk Indicators, Feature Extraction Pipeline, and Communication Data Handler. Based on use case requirements, custom implementation for these components can be provided.

The following e-comm-related feature extractors that are available in Surveillance Insight for Financial Services:

- Emotion and sentiment detection
- Concept mapper
- Entity extractor
- Document classifier (confidential)

The following e-comm-related risk indicators are available in Surveillance Insight for Financial Services:

- Anger anomaly
- Sad anomaly
- Negative sentiment anomaly
- Inbound anomaly
- Outbound anomaly
- Confidential anomaly
- Unusual mail size
- Unusual number of attachments
- Unusual communication timings
- Unusual number of recipients
- Recruit victims
- Recruit co-conspirators
- Efforts on recruit victims
- Efforts on recruit co-conspirators
- Intent to Use Insider Information

E-Comm data ingestion

IBM Electronic Communication Surveillance processes e-comm data based on policy. At least one policy must be defined in the system to be able to process the e-comm data. A policy is a user-defined document that controls the features that need to be extracted.

After policies are created, the e-comm data can be ingested into the solution. Policies can be created and updated by using the REST services. For more information, see Policy service APIs.

A policy can be defined at the system level or per role.

System level policy

System level features are extracted from every communication. The following is an example of a system level policy:

```
{
    "policy": {
        "policyname": "Policy 1",
        "policycode": "POL1",
        "policytype": "system",
        "policysubcategory": "Sub1",
        "policydescription": "System Policy 1",
        "features": [ {
            "name": "emotion"
        },{
            "name": "entity extractor"
        },{
            "name": "recruitvictims"
        },{
            "name": "recruitconspirators"
```

```
        },{
            "name": "ticker"
        },{
            "name": "insiderinformation"
        }
        ]
    }
}
```

Role level policy

Role level features are extracted based on the initiator party's role and the features that are defined for the role. The following is an example of a role level policy:

```
{
    "policy": {
        "policyname": "Policy 2",
        "policycode": "POL2",
        "policytype": "role",
        "policysubcategory": "Sub2",
        "policydescription": "Role Level Policy",
        "role": [
            "Trader",
            "Banker"
        ],
        "features": [{
            "name": "emotion"
        },{
            "name": "entity extractor"
        }]
    }
}
```

If no policies are defined, the solution creates a dynamic policy that is based on the features (FEATURE_MASTER table) and entities (ENTITY_TYPE_MASTER table) that are defined in the solution.

Sample e-comm email and chat

A sample email xml is available here. (www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.3/samplefile/SurveillanceInsightSampleEcommEmail.xml)

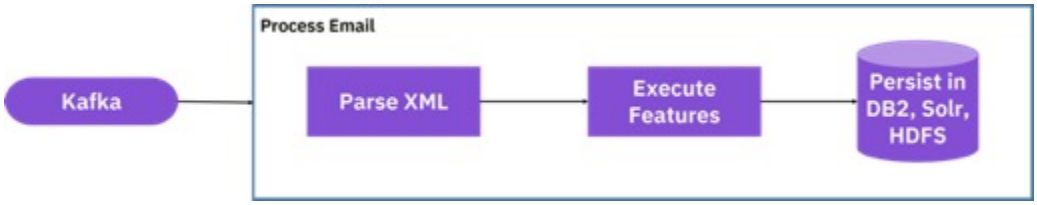
A sample chat xml is available here. (www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.3/samplefile/SurveillanceInsightSampleEcommChat.xml)

E-Comm feature extraction

After the e-comm data is ingested into a Kafka topic, it is processed by set of Spark jobs.

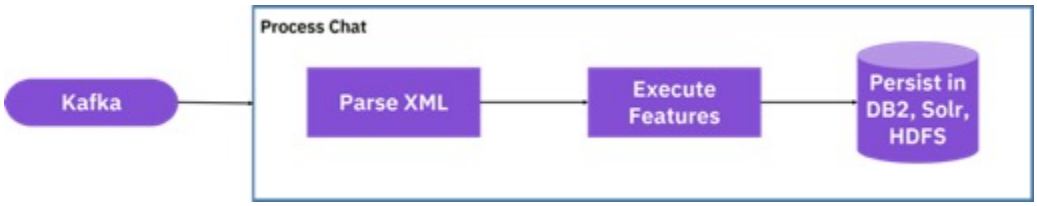
PersistEmail job

This job processes email xml and extracts features based on the defined policies.



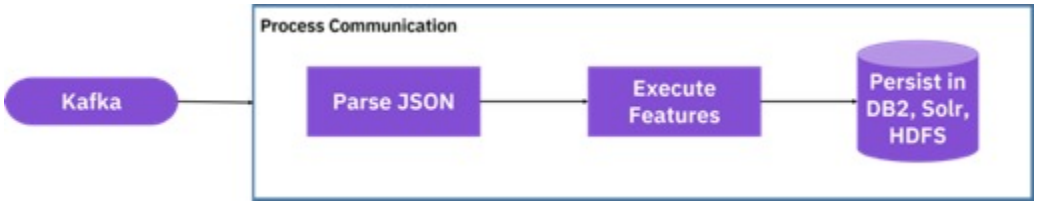
PersistChat job

This job processes chat xml and extracts features based on the defined policies



PersistComm job

This job processes voice and communication data in JSON format and extracts features based on the defined policies.



The above Spark jobs use the ProcessCommunication API.

```
ProcessCommunication processCommunication = new ProcessCommunication();
processCommunication.setConfigProp(sifsProperties);
processCommunication.setMessageType(ECommConstants.EMAIL_MESSAGE_TYPE);

processCommunication.process("PersistEmail")
```

The Spark job must invoke the interface as shown in the code example above.

The ProcessCommunication API provides an interface to add custom features to the e-comm pipeline. For example, if you have implemented an emotion feature transformer using the Spark ML Pipeline Transformer API and you want it to be invoked as part of e-comm pipeline, you must add that feature before you invoke the process. For example:

```
ProcessCommunication processCommunication = new ProcessCommunication();
//Initialize Transformer
EmotionFeatureExtractor emotionFeatureExtractor = new
EmotionFeatureExtractor(configProp.getProperty("dictPath"), configProp.getProperty("rulesPath"))
.setInputCol("commText")
.setOutputCol("emotionFeature")
.setFeatureName("emotion");
processCommunication.addFeature(emotionFeatureExtractor);
processCommunication.process("PersistEmail");
```

The PersistChat and the PersistComm jobs use the same ProcessCommunication API and set the appropriate message type so that data is read from the respective Kafka topic and further processed to extract features.

The PersistEmail, PersistChat, and PersistComm jobs performs following tasks:

1. The job loads all of the parties and their contact points in memory as part of the initialization.
2. The job reads communication data from the Kafka topic in micro batches.
3. The data in each micro batch is processed further:
    a. The data from Kafka is parsed and converted into communication objects.
    b. The policy service is invoked and all of the policies are registered in the system. If no policy is found, a dynamic policy is created with all of the registered features and entities in the Surveillance database.

c. The communication object is enriched with master data, such as party id, job role for initiator, and all participants, and a feature matrix is created based on the eligible policy per initiator.

d. The e-comm pipeline is executed for the features identified in the feature matrix.

e. The REST service is invoked to persist the communication and its associated entities in the SIFS database and in Solr. The REST service (/SIFSServices/commservice/v1/createComm/) creates data in the Comm mapping table, Comm features table, the Comm entity related tables, and also persists data in Solr.

f. The communication and its extracted features are persisted in the HDFS. Data is stored in HDFS in columnar format in form of csv files. A sample file with data is available here. (www.ibm.com/support/knowledgecenter/SSWTQQ_2.0.3/samplefile/SurveillanceInsightSampleEcommFeatureExtraction.csv).

Communication schema

IBM Electronic Communication Surveillance transforms email and chat xml structure to communication objects.

The communication objects use the following fields:

Table 7: Communication schema objects

Field name	Field type	Field description
commType	String	Communication Type such as email, chat, phone
commChannel	String	Communication channel such as voice and e-comm
commText	String	Communication text
commStartTime	String	Communication start time
commEndTime	String	Communication end time
commSubject	String	Communication subject
globalCommId	String	Communication reference ID of the source
metaFeatures	String	The meta features of the communication in JSON format
initiator	String	Communication initiator contact details
participants	Array	Communication participants contact details

HDFS comm schema

E-comm data is processed by the Spark jobs and the extracted features are stored in HDFS so that they can used by the risk scoring module to detect risk.

The following table explains the fields that are stored in HDFS.

Table 8: HDFS comm schema		
Field name	**Field type**	**Field description**
commId	String	Communication Id
commType	String	Type of communication, either e-mail, chat, voice
commChannel	String	Communication channel such as voice and e-comm
globalCommId	String	Communication id received from source data
commStartTime	String	Communication start time. The format is yyyy-mm-dd hh:mm:ss
commEndTime	String	Communication end time. The format is yyyy-mm-dd hh:mm:ss
commSubject	String	Subject used mainly for e-mail
initiator	String	Contact point such as email address or chat id or phone number of communication initiator
initiatorId	String	Party Id of communication initiator
initiatorName	String	Name of communication initiator
participants	String	Contact point such as email address or chat id of communication participants separated by ;
participantsId	String	Party Id of communication participants separated by ;
participantsName	String	Name of communication participants separated by ;
metaFeatures	String	Features extracted from e-comm data such as totalSize, noOfAttachments
emotionFeature	String	Emotion and sentiment feature json extracted from e-comm data
entityFeature	String	Entities detected from e-comm data. Entities are represented in json format
conceptFeature	String	Concepts detected from e-comm data. All concepts are represented in json format
classificationFeature	String	Classifiers detected from e-comm data. Classifiers are represented in json format

E-Comm risk scoring

Communication data is extracted in the form of features and is further analyzed by the inference engine to detect if any risk is found and then publishes that information in the form of alerts to the Case Manager.

Analyze communication job

After the e-comm data is processed by the Spark job, it is available in HDFS in form of communication data and its extracted features. This data is further analyzed at the end of day by the analyze communication Spark job.

The analyze communication job is implemented by using the ProcessCommunication API. It supports a mechanism to add new risk indicators.

```
ProcessCommunication processCommunication = new ProcessCommunication();
processCommunication.setConfigProp(sifsProperties);
BehaviorRiskIndicator behaviorRiskIndicator = new BehaviorRiskIndicator()
.setInputCol("emotionFeature").setSadOutputCol("RD2")
.setAngerOutputCol("RD1").setSentimentalOutputCol("RD3")
.setSelfThreshold(Double.valueOf(sifsProperties.getProperty("behSelfThreshold ")))
.setPopThreshold(Double.valueOf(sifsProperties.getProperty("behPopThreshold")
))
.setRiskScoreThreshold(Double.valueOf(sifsProperties.getProperty("riskScoreTh reshold")))
processCommunication.addRiskIndicator(behaviorRiskIndicator);

processCommunication.analyze("AnalyzeComm",analysisDate);
```

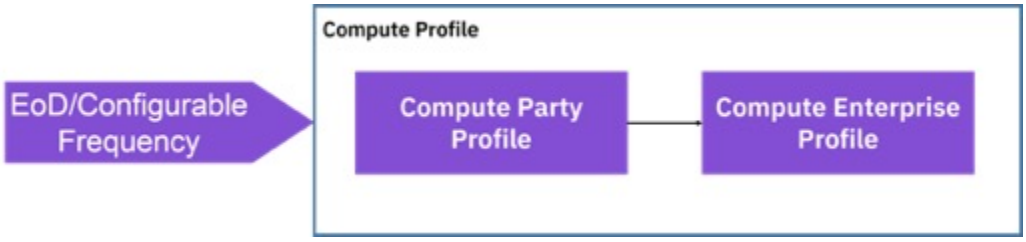
This job performs the following tasks:

1. The job reads the communication data and extracted features from HDFS for the date for which analysis is being done.
2. This job aggregates the data per party and computes the behavior profile, such as the max anger score or the max disgust score, and persists the data in SIFS database.
3. The following risk indicators are computed:
    a. Anger anomaly
    b. Sad anomaly
    c. Negative sentiment anomaly
    d. Inbound anomaly
    e. Outbound anomaly
    f. Confidential anomaly
    g. Unusual mail size
    h. Unusual number of attachments
    i. Unusual communication timings
    j. Unusual number of recipients
    k. Recruit victims
    l. Recruit co-conspirators
    m. Efforts on recruit victims
    n. Efforts on recruit co-conspirators
    o. Intent to use insider information

4. The risk indicators are implemented by using the Spark ML Pipeline Transformer API. To create a new risk indicator, you must write a custom transformer and get the transformer invoked through the pipeline or invoke it independently by using the Transformer transform API.
5. Risk Indicators are persisted to the SIFS database and to Solr through the REST service (/SIFSServices/alertservice/v1/alert/createEvidence).
6. For anomaly based risk indicators, the job computes a profile per party to get the average score for the day and then persists the scores in the SIFS database.
7. The job invokes the inference model—the Party Behavior Risk Model—to detect if the risk evidences have any alert conditions and then persists those alerts in the SIFS database and in Solr and also publishes the alerts to Case Manager.

Profile aggregator job

This job computes the reference profile for each party and then updates the party profile and enterprise profile in the SIFS database.



1. This job computes the profile—MEAN and STD for all parties for a given date. The job expects the date as an input parameter and expects the window parameter to be set in the `sifs.spark.properties` file, where window is the number of days for which the MEAN and STD need to be calculated.
2. This job updates the MEAN and STD values in the PARTY_PROFILE for the profile date. It also inserts the MEAN and STD in the ENTERPRISE_POFILE for the profile date. Surveillance Insight expects that the job is run for a given date only one time. If the same job is run for the same date more than one time, an exception is logged by the job. Only INSERT functions are supported. UPDATE functions are not supported.

Party risk scoring job

This job computes the risk score of a party based on the past alerts for that party.

The job requires the following values:

- PartyRiskDateWindow = 90
- SolrProxyURL = https://localhost:9443/SIFSServices/surveillanceui/v1/index/update

The job reads the past alerts based on the PartyRiskDateWindow value.

Discovery

Discovery makes it possible to rapidly build cognitive exploration applications that unlock actionable insights hidden in unstructured data.

Typically with machine learning models, good results can be achieved with a trained dataset. For training, labels must be used. In the context of e-comm, to determine whether it is a complaint, a training dataset with labels must be used. Such trained datasets can take time to create, however a dictionary-based approach for doing supervision (to detect complaints) can also be used. The objective of the discovery process is to allow the users to prepare the dictionary quickly.

Different datasets will have different terms or keywords. The approach used in Surveillance Insight provides a tool, which can be used against any dataset to discover the key features in that dataset that can then be used for preparing the dictionary or preparing the supervised training dataset with labels. This approach accelerates the process and provides good results.

For more information about the high-level steps to create a discovery model, see the Surveillance Model Design Studio guide.

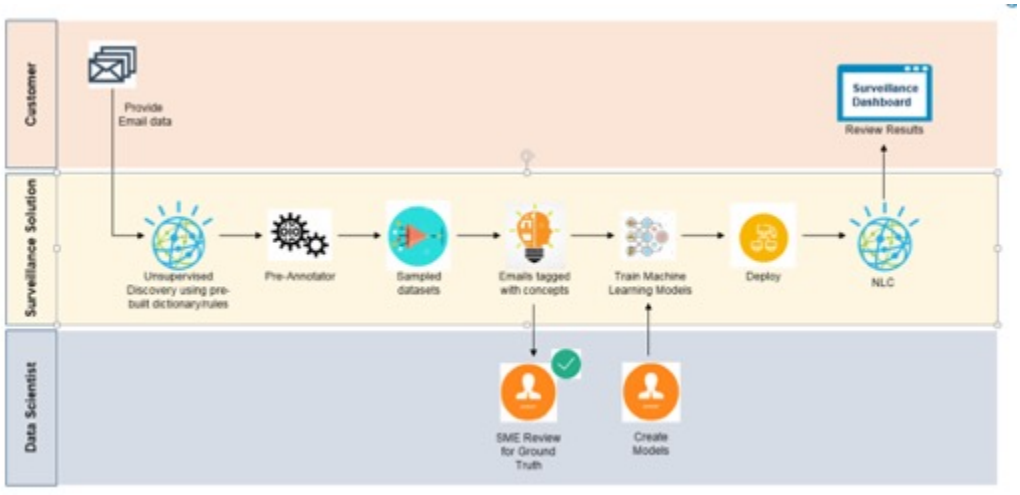


Figure 12: Discovery overview

IBM Electronic Communication Surveillance allows users to run discovery on unstructured data. For example, to classify email as complaints or non-complaint, the following steps must occur:

1. Create a discovery model
2. Run the discovery model for a given dataset
3. Tag the discovered terms and create a dictionary
4. Extract concepts from emails based on the dictionary
5. Execute rules on the extracted concepts and classify the emails as complaint or non-complaint
6. Sample the data and create datasets
7. Perform ground truth labeling for the generated datasets
8. Train the NLC model for the complaints classifier by using the sampled data
9. Publish the trained model
10. Use the published model for document classification

The Discovery job

The Discovery job read email data from HDFS, produces discovery results, and stores the results in CSV format in HDFS.

The job uses the following arguments: the dataset path where the snapshot xml files are stored, the model ID, and the iteration ID.

The job does the following:

1. After the job is initialized, the iteration status is set to “Discovery Started”
2. Email data is read from the HDFS folder and the emails are parsed
3. For each email, invoke the discovery service; for example, analyze method to get the discovery results
4. Store the discovery results in HDFS in CSV format. The discovery results are stored in HDFS folder (incoming dataset path + HDFSDiscoveryPath, as configured in the `sifs.spark.properties` file)
5. After the results are stored, set the iteration status to “Discovery Done”
6. Post that the user can click View to see the discovered entities

The Unsupervised job

The unsupervised job reads email data from HDFS and produces email features and tags for each email and stores the results in CSV format in HDFS.

The job uses the following arguments:

- the dataset path where the snapshot xml files are stored, for example `/user/sifsuser/comm_ds1`
- the dataset path where the email features and tags are stored, for example `/user/sifsuser/comm_ds1/unsupervised`
- the rules file, for example `/home/sifsuser/rules/complaint.rules`

The following is an example of the contents of a rules file:

```
tag=complaints,non-complaints
rule1=(context == 'Customer' || actionverb == 'Rate Lock' || object == 'AppProcessing')
rule2=(context == 'Customer' && (actionverb == 'Rate Lock' || object == 'AppProcessing'))
```

Depending on the lexicons, users can edit the rules file and its outcome.

The job does the following:

1. Reads email data from the HDFS folder and parses the emails
2. For each email, invoke the concept mapper for all published lexicons
3. Invoke the discovery service, for example analyze method to get the discovery results
4. Extract the rules file, with the property name starting with “rule”, and execute all of the rules. If any one rule is true, then the email is tagged. For example, if rule1 is evaluated as true then the email is tagged as “complaints”. If not, the email is tagged as “non-complaints”
5. The email features and tags are stored in HDFS, Db2 and Solr:
   - In HDFS, they are stored in the path provided by the user
   - In Db2 and Solr, the data is stored through a REST service (/createComm)

The sampling job

The email sampling job creates a set of data samples from a corpus of emails. The data samples can be used by Subject Matter Experts to determine the ground truth; that is, whether the emails really are complaints or not. Then, the annotated emails can be used to train the natural language classifier and natural language understanding models.

The job uses the following arguments:

- The HDFS path where the email features are stored, for example, `/user/sifsuser/comm_ds1/unsupervised`
- The number of datasets to be created, for example, 2
- The number of records per dataset, for example, 500
- The class ratio, which defines what tags are used to select emails and also gives the required ratio of each tag that will be used for the samples. For example, "{'complaints':0.5,'non-complaints':0.5}"
- The dataset prefix, for example X01

The job does the following:

1. All emails are cleaned and processed so that only the current email text is used
2. The email class counts are adjusted to reach the desired balance of classes, for examle, complaints=75%, non-complaints=25%
3. The emails are processed by a Latent Dirichlet Allocation algorithm into groups of consistent themes
4. Small datasets are produced that contain representative amounts of all of the themes that are discovered in the previous step
5. The dataset IDs for each sampled email are sent by the REST API and stored in the database as a tag for a given communication ID

The NLC training Job

The NLC training job creates Natural Language Classifier Training files from annotated emails.

The job uses the following arguments:

- The HDFS path where the email features are stored, for example, /user/sifsuser/comm_ds1/unsupervised
- The HDFS path where the sampled data must be stored, for example, /user/sifsuser/comm_ds1/samples
- The list of tags to be used, for example, X01_dataset_0:X01_dataset_3
- The classes to be selected for output, for example, 'complaints,non-complaints'
- The file number to be created, for example, 1

The job does the following:

1. The script reads entries from the SIFS.COMM_FEATURES database table for emails tagged with specific dataset_ids
2. These are then merged with the original email text and used to create training files in the format <email text>,<label>
3. The training files can then be used to train the NLC models

The e-comm pipeline job

The e-comm pipeline job reads email data from HDFS, produces email features, and stores the results in CSV format in HDFS, Db2, and Solr.

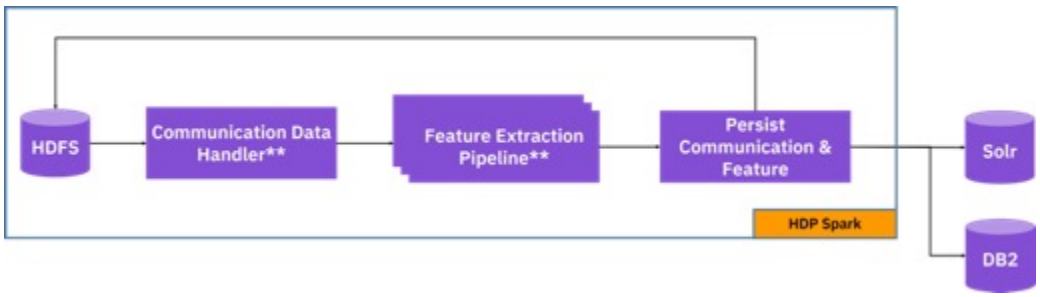


Figure 13: The e-comm pipeline job

The job uses the dataset path where the snapshot xml is stored as an argument.

The processing of this job is the same as the PersistEmail job, except for the source of the emails. In the PersistEmail job, the emails are sourced from Kafka, whereas in this job they are sourced from HDFS.

The job does the following:

1. Read email data from the HDFS folder and parse the emails
2. Get all published NLC models
3. For each email, invoke all of the features that are enabled in the policy
   - Emotion
   - Entity Extractor
   - Lexicons
   - Classifier
4. Store the feature results in HDFS in CSV format
5. The feature results are stored in the HDFS folder (incoming dataset path + HDFSCommFeaturesPath, as configured in the `sifs.spark.properties` file
6. The features are persisted in Db2 and Solr through the REST service (/createComm)

E-Comm Spark job configration

The E-Comm Spark job uses the `sifs.spark.properties` file for the job parameters.

`sifs.spark.properties` contains the following parameters:

Table 9: E-Comm Spark job parameters

Property name	Property value	Description
metadata.broker.list	*<IP>*:6667	The IP address and port number where the Kafka server is running
security.protocol	SASL_SSL	
ssl.truststore.location	/home/sifsuser/security/kafka.client.truststore.jks	The Kafka client SSL truststore location as configured in Kafka
ssl.truststore.password	PltAdmin1PltAdmin1	The SSL truststore password as configured in Kafka
ssl.keystore.location	/home/sifsuser/security/kafka.client.keystore.jks	The Kafka client SSL keystore location as configured in Kafka
ssl.keystore.password	PltAdmin1PltAdmin1	The SSL keystore password as configured in Kafka
ssl.key.password	PltAdmin1PltAdmin1	SSL key password

Table 9: E-Comm Spark job parameters (continued)		
Property name	**Property value**	**Description**
bootstrap.servers	*<IP>*:6667	The IP address and port number where the Kafka server is running
group.id	spark-streaming-notes	Group name to subscribe to the Kafka topic
auto.offset.reset	earliest	Kafka auto offset setting
sparkWarehousePath	file:///home/sifsuser/spark-2.1.1-hadoop2.7/bin/spark-warehouse	
HDFSFilePath	hdfs://*<IP>*:8020/user/sifsuser/	HDFS path
KafkaSSLEnabled	true	If SSL is enabled for Kafka. The value can be true or false.
KafkaEncryptionEnabled	true	If encryption is enabled for Kafka. The value can be true or false.
master	yarn	If the job is running on a yarn cluster
InferenceREST	https://*<IP>*:*<PORT>*/SIFSModelServices/inference/model/predict	The URL where the Inference REST service is hosted
CreateAlertREST	https://*<IP>*:*<PORT>*/SIFSServices/alertservice/v1/alert/createAlert	The REST service URL to create alerts
CreateEvidenceREST	https://*<IP>*:*<PORT>*/SIFSServices/alertservice/v1/alert/createEvidence	The REST service URL to create risk evidences
UpdateAlertREST	https://*<IP>*:*<PORT>*/SIFSServices/alertservice/v1/alert/updateAlert	The REST service URL to update alerts
PartyRiskDateWindow	90	To get past alerts for last 90 days. Used by party risk scoring job
SolrProxyURL	https://*localhost*:9443/SIFSServices/surveillanceui/v1/index/update	The REST service URL to update the party risk scoring job in Solr
db2jdbcurl	jdbc:db2://*<IP>*:50001/SIFS:sslConnection=true;currentSchema=SIFS;	JDBC URL to connect to the SIFS database
db2user	db2inst1	Database user
db2password	{xor}DzMrHjsyNjFu	Database password
db2TrustStore	/home/sifsuser/security/fci_universal_ks.jks	Keystore to connect to the secure database
db2TrustStorePassword	{xor}DzMrHjsyNjFuDzMrHjsyNjFu	Kesytore password
spark.streaming.kafka.consumer.poll.ms	512	Kafka polling time

Table 9: E-Comm Spark job parameters (continued)		
Property name	**Property value**	**Description**
spark.streaming.backpressure.enabled	true	Spark streaming parameter
spark.streaming.receiver.maxRate	20	Spark streaming parameter
spark.streaming.kafka.maxRatePerPartition	20	
ecommTopic	sifs.ecomm.in	Kafka topic to which PersistComm spark job is polling to and accepting communication formatted as JSON
window	30	Number of days used to compute the rolling average by the ProfileAggregator Spark job
riskModelCode	PR	Risk model code to invoke the Party Behavior Risk Model. If you have more than one model, separate each by a comma
kafkaDuration	20	Duration in seconds after which the Kafka topic is polled by the Spark jobs
emailTopic	sifs.email.in	Kafka topic to which the PersistEmail Spark job is polling to and accepting email as XML files
chatTopic	sifs.chat.in	Kafka topic to which the PersistChat Spark job is polling to and accepting chat data as XML files
dictPath	/home/sifsuser/dict	Path where the dictionaries for Emotion and Concept Mapper are copied
rulesPath	/home/sifsuser/rules	Path where the rules for Emotion and Concept Mapper are copied
commFolderPath	comm/	Folder name in HDFS where the communication and extracted features are stored
policyServiceUrl	https://*<IP>*:*<PORT>*/CommServices/ecomm/policy	The REST service URL for querying policies
policyServiceUser	ibmrest1	Policy service user
policyServicePassword	ibmrest@pwd1	Policy service password
CreateCommREST	https://*<IP>*:*<PORT>*/SIFSServices/commservice/v1/createComm	The REST service URL to create comm mapping and entity relationship in the SIFS database

Table 9: E-Comm Spark job parameters (continued)		
Property name	**Property value**	**Description**
entityServiceUrl	https://*<IP>*:*<PORT>*/analytics/models/v1/analyzetext/	The REST service URL to invoke the entity extractor feature
riskScoreThreshold	0.5	Risk score threshold above which the risk evidences are stored in the SIFS database
behSelfThreshold	1.0	Self-threshold for the Behavior Anomaly Risk Indicator
behPopThreshold	2.0	Population threshold for the Behavior Anomaly Risk Indicator
mailSize	2000	Threshold for the size of the communication content
numberOfAttachments	3	Threshold for the number of attachments
windowStartTime	09:00:00	Acceptable communication start time window
windowEndTime	18:00:00	Acceptable communication end time window
numberOfRecipients	4	Threshold for the number of recipients
commVolumeSelfThreshold	1.0	Self-threshold for inbound and outbound anomaly risk indicator
commVolumePopThreshold	2.0	Population threshold for inbound and outbound anomaly risk indicator
recruitSelfThreshold	1.0	Self-threshold for the recruit victims and recruit conspirators for the anomaly risk indicator
recruitPopThreshold	2.0	Population threshold for recruit victims and recruit conspirators for the anomaly risk indicator
commContentSelfThreshold	1.0	Self-threshold for the confidential anomaly risk indicator
commContentPopThreshold	2.0	Population threshold for the confidential anomaly risk indicator
HDFSDiscoveryPath	output	The HDFS folder inside which the discovery results are stored. This folder is relative to the dataset path provided for discovery job
discoveryServiceUrl	https://*<IP>*:*<PORT>*/discovery/v1/analyse	Discovery service url

Table 9: E-Comm Spark job parameters (continued)		
Property name	**Property value**	**Description**
iterationServiceUrl	https://*<IP>*:*<PORT>*/SIFSModelServices/discoverymodel/iterations	Update iteration service url
nlcServiceUrl	https://*<IP>*:*<PORT>*/nlc/v1/models/	NLC service url
nlcMethodUri	/classify/	NLC method uri
HDFSCommFeaturesPath	features	HDFS path where pipeline extracted features are stored. This path is relative to the HDFS path provided as input to e-comm pipeline job
sampleServiceUrl	https://*<IP>*:*<PORT>*/SIFSServices/commservice/v1/communication/{communicationid}/createTag	Service to create tags for a given communication id

End-to-end flow for e-comm processing

The following describes the end-to-end flow for e-comm processing.

1. Create a risk model for party behavior by using the Model Building tool.
   a. Create training data and train the model.
   b. Publish the model.
2. Create a natural language classifier (NLC) by using the Model Building tool.
   a. Create training data and train the classifier model.
   b. Publish the classifier model.
3. Configure the new models for the Spark jobs to consume.
4. Deploy the feature extraction Spark jobs: PersistEmail, PersistChat, PersistComm
5. Deploy the inference job: AnalyzeCommunication
6. Create policy.
7. Ingest the data.
8. Review the alerts in the Surveillance Insight dashboard.

Chapter 4. IBM Trade Surveillance Analytics

IBM Surveillance Insight for Financial Services trade surveillance offers mid and back-office surveillance on market activities and communication to detect and report possible market abuse activity.

The trade component monitors trade data, detects suspicious patterns against the predefined risk indicators, and reports the patterns. The trade data includes order data, trade data, quotes, executions, and end of the day summary data. Also included are the transactions and market reference data from the equity market.

The risk indicators are analyzed by the inference engine. The inference engine uses a risk model to determine whether an alert needs to be created.

The following use cases are provided:

- Pump-and-dump
- Spoofing
- Off-market (equity market)

The following trade-related risk indicators are available in the Surveillance Insight for Financial Services master data:

- Bulk orders
- High order-to-order cancel ratio
- Bulk executions
- Unusual quote price movement
- Pump in the stock
- Dump in the stock
- Deal rate anomaly (equity market)
- Party past alerts

Data ingestion

Market data, such as trade, order, quote, and execution data, are uploaded to the Hadoop file system (HDFS) by using the HDFS terminal. The naming conventions for the files and folder are as follows:

- `/user/sifsuser/trade/Trade_<yyyy-mm-dd>.csv`
- `/user/sifsuser/order/Order_<yyyy-mm-dd>.csv`
- `/user/sifsuser/execution/Execution_<yyyy-mm-dd>.csv`
- `/user/sifsuser/quote/Quote_<yyyy-mm-dd>.csv`
- `/user/sifsuser/EOD/EOD_<yyyy-mm-dd>.csv`
- `/user/sifsuser/transactions/transactions_<yyyy-mm-dd>.csv`
- `/user/sifsuser/marketReference/marketReference_<yyyy-mm-dd>.csv`

The current implementation of the trade use cases expects that there is one file of each type for each day.

The IBM InfoSphere® Streams data loader job monitors the folders. The job reads any new file that is dropped into the folder and sends it for downstream processing.

Trade Surveillance Toolkit

The Trade Surveillance Toolkit helps the solution developers to focus on specific use case development.

The toolkit contains basic data types, commonly used functional operators relevant to trade analytics, and adapters for some data sources.

The Surveillance Base Toolkit includes the following risk indicator operators:

- Bulk orders detection
- High order-to-order cancel ratio
- Bulk execution detection
- Unusual quote price movement
- Deal rate anomaly (equity market)

The risk evidence sink operator is also included.

The Surveillance Base Toolkit includes the following schema type definitions:

- Order
- Quote
- Execution
- Trade
- Transaction
- Market reference
- Risk event
- Trade evidence

For information about the schemas for the types that are defined in the toolkit, see Trade Toolkit data schemas.

Note:

- The risk event and trade evidence schemas are new in this release. All new risk indicator implementations must use these types to create events. The deal rate anomaly risk indicator provides an example of how to use these types. The other risk indicator implementations use the event and event data types, which are deprecated. It is not recommended to use the event and event data types.
- The risk evidence sink operator makes it easier to create risk evidences for downstream consumption. It uses the risk event type events as input. Users of available risk indicators (other than deal rate anomaly), must convert the event type event to the risk event type event before you can push the event to the risk evidence sink operator.

Bulk Order Detection operator

Purpose

Looks at a sliding window of orders and checks if total order volume is over the Bulk Volume Threshold. It is grouped by trader, ticker, and order side (buy/sell). The sliding window moves by 1 second for every slide.

Input

Order Data according to the schema.

Output event contents

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: BULK_ORDER

Trader ID: ID of the trader who is placing the order

Ticker

Event Data

orderQty: total volume of orders in the window for Trader ID

Side: BUY or SELL

maxOrderPrice: maximum order price that was seen in the current window

Configuration

Window Size: time in seconds for collecting data to analyze

Bulk Volume Threshold: Volume threshold that is used to trigger events

High order cancellation operator

Purpose

Looks at a sliding window of window size (in seconds) and checks if total order volume to order cancellation volume for a trader is above the cancellation threshold. It is grouped by trader, ticker, and order side (buy/sell).

Input

Order Data according to the schema.

Output event contents

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: HIGH_CANCEL_RATIO

Trader ID: ID of the trader who is placing the order

Ticker

Event Data

Side: BUY or SELL

Ratio: order volume versus cancellation ratio

Configuration

Window Size: time in seconds for collecting data to analyze

Window Slide: Slide value for the window in seconds

Cancellation Threshold: Volume threshold that is used to trigger events

Price Trend operator

Purpose

Looks at a sliding window of quotes and computes the rise or drop trend (slope) for offer and bid prices. It fires an event if the price slope rises above the Rise Threshold or drops below the Drop Threshold. The former indicates an unusual rise in the quotes and the latter indicates an unusual drop in the quotes. The analysis is grouped by ticker.

Input

Quote Data according to the schema.

Output event contents

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: PRICE_TREND

Trader ID: not applicable

Ticker

Event Data

Side: BID or OFFER

Slope: slope of the bid or offer price

Configuration

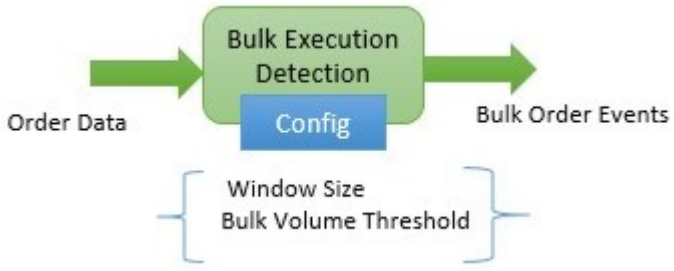
Window Size: time in seconds for collecting data to analyze

Window Slide: Slide value for the window in seconds

Drop Threshold: Threshold that indicates an unusual downward trend in the quotes

Rise Threshold: Threshold that indicates an unusual rise trend in the quotes

Bulk Execution Detection operator



Purpose

Looks at a sliding window of executions and checks if the total executed volume is above the Bulk Volume Threshold. It is grouped by trader, ticker, and order side (buy/sell). The sliding window moves by 1 second for every slide.

Input

Execution Data according to the schema.

Output event contents

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: BULK_EXEC

Trader ID: ID of the trader who is placing the order

Ticker

Event Data

orderQty: total volume of executions in the window for Trader ID

Side: BUY or SELL

TotalExecValue: price * execution quantity for this window. It is grouped by ticker, trader, and side

Configuration

Window Size: time in seconds for collecting data to analyze

Bulk Volume Threshold: The volume threshold that is used to trigger events

Deal Rate Anomaly operator

Purpose

Looks at forex transaction data from the equity market and matches the deal rate against the high and low market values as mentioned in the market reference data at the time of the transaction. If the deal rate falls outside of the high and low range, a deal rate anomaly event is fired.

Input

Transaction data and market reference data.

Output event contents

Note: This operator outputs riskEvent type events. For more information, see “Risk event schema” on page 51.

Id: unique ID for this event

Event Time: time in input data, not system time

Event Type: Transaction Deal Rate Anomaly

Trader ID: ID of the trader who is placing the transaction

Trade evidence data:

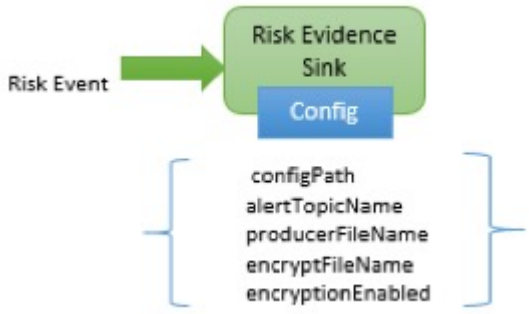
- dataType: transaction
- startTime: ""
- windowSize: 0.0
- id: transaction ID

Configuration

Start time: The time in the input data that corresponds to the first record in the input. This is used to compute the one-minute window durations to read the market reference data.

Risk Indicator Look Up Code: This is the look-up code for Deal Rate Anomaly in the master data in the SIFS database. This value is used to send out the risk indicator to the downstream processes for persisting the risk evidence with the corresponding look-up code.

Risk Evidence Sink operator



Purpose

This operator abstracts the job of creating a risk evidence message, encrypting it, and the passing it on to a Kafka topic for the downstream processes to consume. It takes risk event type as the input and converts it into a risk evidence JSON. It then, optionally, encrypts the JSON and drops it into a pre-configured Kafka topic.

Input

Risk event.

Configuration

configPath: The file system path where the Kafka producer properties file can be found

alertTopicName: The topic name in Kafka where the operator drops the risk evidences

producerFileName: The name of the Kafka producer properties file

encryptFileName: The name of the encryption properties file

encryptionEnabled: Toggles encryption of the risk evidence JSON before it is dropped into the Kafka topic

Ticker price schema

```
symbol,datetime,price
```

Table 10: Ticker price schema

Field name	Field type	Description
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the trade occurred
Price	Float	The unit price of the stocks traded

Execution schema

```
Id, Symbol, Datetime, Brokerid, Traderid, Clientid, effectiveTime, expireTime,
timeInForce, exposureDuration, tradingSession, tradingSessionSub, settlType,
settlDate, Currency, currencyFXRate, execType, trdType, matchType, Side,
orderQty, Price, exchangeCode, refQuoteId, refOrderId
```

For more information about the fields in this schema, refer to the FIX wiki (http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.0SP2%2B)

Table 11: Execution schema

Field name	Field type	Description
Id	String	Unique identifier for the execution
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the trade occurred. The format is yyyy-mm-dd hh:mm:ss
Brokerid	String	The ID of the broker that is involved in this execution
Traderid	String	The ID of the trader that is involved in this execution
Clientid	String	The ID of the client that is involved in this execution
effectiveTime	String	The date and time stamp at which the execution is effective
expireTime	String	The date and time stamp when this execution will expire
timeInForce	String	Specifies how long the order remains in effect. Absence of this field is interpreted as DAY
exposureDuration	String	The time in seconds of a "Good for Time" (GFT) TimeInForce
tradingSession	String	Identifier for a trading session
tradingSessionSub	String	Optional market assigned sub identifier for a trading phase within a trading session
settlType	String	Indicates order settlement period. If present, SettlDate overrides this field. If both SettlType and SettDate are omitted, the default for SettlType is 0 (Regular)
settlDate	String	Specific date of trade settlement (SettlementDate) in YYYYMMDD format

Table 11: Execution schema (continued)		
Field name	**Field type**	**Description**
Currency	String	The currency in which the execution price is represented
currencyFXRate	Float	The foreign exchange rate that is used to calculate SettlCurrAmt from Currencyto SettlCurrency
execType	String	Describes the specific ExecutionRpt (for example, Pending Cancel) while OrdStatus will always identify the current order status (for example, Partially Filled)
trdType	String	Type of trade
matchType	String	The point in the matching process at which this trade was matched
Side	String	Denotes BUY or SELL execution
orderQty	Int	The volume that is fulfilled by this execution
Price	Float	The price per unit for this execution
exchangeCode	String	
refQuoteId	String	The quote that corresponds to this execution
refOrderId	String	Refers to the order corresponding to this execution

Order schema

`Id, Symbol, Datetime, effectiveTime, expireTime, timeInForce, exposureDuration, settlType, settlDate, Currency, currencyFXRate, partyId, orderType, Side, orderQty, minQuantity, matchIncr, Price, manualOrderIndicator, refOrderId, refOrderSource`

For more information about the fields in this schema, refer to the FIX wiki (http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.0SP2%2B)

Table 12: Order schema		
Field name	**Field type**	**Description**
Id	String	Unique identifier for the order
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the order was placed. The format is yyyy-mm-dd hh:mm:ss

Table 12: Order schema (continued)

Field name	Field type	Description
effectiveTime	String	The date and time stamp at which the order is effective
expireTime	String	The date and time stamp when this order will expire
timeInForce	String	Specifies how long the order remains in effect. If this value is not provided, DAY is used as the default
exposureDuration	String	The time in seconds of a "Good for Time" (GFT) TimeInForce
settlType	String	Indicates order settlement period. If present, SettlDate overrides this field. If both SettlType and SettDate are omitted, the default for SettlType is 0 (Regular)
settlDate	String	Specific date of trade settlement (SettlementDate) in YYYYMMDD format
Currency	String	The currency in which the order price is represented
currencyFXRate	Float	The exchange rate that is used to calculate the SettlCurrAmt from Currencyto SettlCurrency
partyId	String	The trader that is involved in this order
orderType	String	CANCEL represents an order cancellation. Used with refOrderId.
Side	String	Indicates a BUY or SELL order
orderQty	Int	The order volume
minQuantity	Int	Minimum quantity of an order to be executed
matchIncr	Int	Allows orders to specify a minimum quantity that applies to every execution (one execution might be for multiple counter-orders). The order can still fill against smaller orders, but the cumulative quantity of the execution must be in multiples of the MatchIncrement
Price	Float	The price per unit for this order

Table 12: Order schema (continued)		
Field name	**Field type**	**Description**
manualOrderIndicator	boolean	Indicates whether the order was initially received manually (as opposed to electronically) or if it was entered manually (as opposed to it being entered by automated trading software)
refOrderId	String	Used with the orderType. Refers to the order that is being canceled
refOrderSource	String	The source of the order that is represented by a cancellation order

Quote schema

```
Id, Symbol, Datetime, expireTime, exposureDuration, tradingSession,
tradingSessionSub, settlType, settlDate, Currency, currencyFXRate, partyId,
commPercentage, commType, bidPrice, offerPrice, bidSize, minBidSize,
totalBidSize, bidSpotRate, bidFwdPoints, offerSize, minOfferSize,
totalOfferSize, offerSpotRate, offerFwdPoints
```

For more information about the fields in this schema, refer to the FIX wiki (http://fixwiki.org/fixwiki/ExecutionReport/FIX.5.0SP2%2B)

Table 13: Quote schema		
Field name	**Field type**	**Description**
Id	String	Unique identifier for the quote
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the quote was placed. The format is yyyy-mm-dd hh:mm:ss
expireTime	String	The date and time stamp when this quote will expire
exposureDuration	String	The time in seconds of a "Good for Time" (GFT) TimeInForce
tradingSession	String	Identifier for a trading session
tradingSessionSub	String	Optional market assigned sub identifier for a trading phase within a trading session
settlType	String	Indicates order settlement period. If present, SettlDate overrides this field. If both SettlType and SettDate are omitted, the default for SettlType is 0 (Regular)

Table 13: Quote schema (continued)		
Field name	**Field type**	**Description**
settlDate	String	Specific date of trade settlement (SettlementDate) in YYYYMMDD format
Currency	String	The currency in which the quote price is represented
currencyFXRate	Float	The exchange rate that is used to calculate SettlCurrAmt from Currencyto SettlCurrency
partyId	String	The trader that is involved in this quote
commPercentage	Float	Percentage of commission
commType	String	Specifies the basis or unit that is used to calculate the total commission based on the rate
bidPrice	Float	Unit price of the bid
offerPrice	Float	Unit price of the offer
bidSize	Int	Quantity of bid
minBidSize	Int	Type of trade
totalBidSize	Int	
bidSpotRate	Float	Bid F/X spot rate
bidFwdPoints	Float	Bid F/X forward points added to spot rate. This can be a negative value
offerSize	Int	Quantity of the offer
minOfferSize	Int	Specifies the minimum offer size
totalOfferSize	Int	
offerSpotRate	Float	Offer F/X spot rate
offerFwdPoints	Float	Offer F/X forward points added to spot rate. This can be a negative value

Trade schema

```
Id, Symbol, Datetime, Brokerid, Traderid, Clientid, Price, Volume, Side
```

Table 14: Trade schema		
Field name	**Field type**	**Description**
Id	String	Unique identifier for the trade
Symbol	String	The ticker corresponding to the trade

Table 14: Trade schema (continued)		
Field name	**Field type**	**Description**
Datetime	String	The date and time at which the trade occurred. The format is yyyy-mm-dd hh:mm:ss
Brokerid	String	The id of the broker involved in the trade
Traderid	String	The id of the trader involved in the trade
Clientid	String	The id of the client involved in the trade
Price	Float	The unit price of the stocks traded
Volume	Int	The volume of stocks traded
Side	String	The BUY or SELL side of the trade

End of day (EOD) schema

```
Id, Symbol, Datetime, openingPrice, closingPrice, dayLowPrice, dayHighPrice,
Week52LowPrice, Week52HighPrice, marketCap, totalVolume, industryCode, div,
EPS, beta, description
```

Table 15: End of day (EOD) schema		
Field name	**Field type**	**Description**
Id	String	Unique identifier for the trade
Symbol	String	The ticker corresponding to the trade
Datetime	String	The date and time at which the trade occurred. The format is yyyy-mm-dd hh:mm:ss
openingPrice	Float	The opening price of the ticker for the date that is specified in the datetime field
closingPrice	Float	The closing price of the ticker for the date that is specified in the datetime field
dayLowPrice	Float	The lowest traded price for the day for this ticker
dayHighPrice	Float	The highest traded price for the day for this ticker
Week52LowPrice	Float	The 52-week low price for this ticker
Week52HighPrice	Float	The 52-week high price for this ticker
marketCap	Float	The market cap for this ticker

Table 15: End of day (EOD) schema (continued)		
Field name	**Field type**	**Description**
totalVolume	Int	The total outstanding volume for this ticker as of today
industryCode	String	The industry to which the organization that is represented by the ticker corresponds to
Div	Float	
EPS	Float	
Beta	Float	
Description	String	The description of the organization that is represented by the ticker

Market reference schema

```
symbol, periodDate, periodNumber, periodStart, periodEnd, open, high, low,
close, mid
```

Transaction schema

```
transactionID, linkedOrderID, tradeDate, timeExecuted, valueDate, productType,
dealerCode, portfolioCode, counterpartyCode, counterpartyName,
counterpartyLocation, channel, broker, buyCCY, sellCCY, dealRate, buyCCYAmount,
sellCCYAmount, transactionStatus, traderId, symbol
```

Risk event schema

```
rstring id, rstring description, rstring eventType, rstring startTime, float64
windowSize, rstring traderId, rstring symbol, float64 score,
list<tradeEvidence> evidenceData ;
```

Trade evidence schema

```
rstring dataType, rstring startTime, float64 windowSize, list<rstring> id;
```

Event schema

```
id, eventType, startTime, windowSize, traderId, symbol, data
```

Table 16: Event schema		
Field name	**Field type**	**Description**
id	String	System generated id for the event
eventType	String	The type of the event
startTime	String	The system time when the event occurred

Table 16: Event schema (continued)		
Field name	**Field type**	**Description**
windowSize	Float	The size (in seconds) of the data window that the operator used while looking for events in the input data stream.
traderId	String	The trader id associated with the event
symbol	String	The symbol associated with the event
data	List of event data	Event specific data list. See Event Data schema

Event data schema

```
name, value
```

Table 17: Event data schema		
Field name	**Field type**	**Description**
name	String	The name of the event property
value	String	The value of the event property

Pump-and-dump use case

The solution provides a pump-and-dump use case, which carries out structured analysis of trade, order, and execution data and unstructured analysis of email data. The result is a daily score for the pump-and-dump indication.

The pump-and-dump score is distributed daily among the top five traders. Top five is determined based on the positions that are held by the traders.

Triggering the pump-and-dump rules

Ensure that the following folders exist on the Hadoop file system. The folders are:

- /user/*sifsuser*/trade/
- /user/*sifsuser*/order/
- /user/*sifsuser*/execution/
- /user/*sifsuser*/quote/
- /user/*sifsuser*/EOD/
- /user/*sifsuser*/sifsdata/ticker_summary/ticker_summary/
- /user/*sifsuser*/sifsdata/position_summary/
- /user/*sifsuser*/sifsdata/positions/
- /user/*sifsuser*/sifsdata/pump_dump/
- /user/*sifsuser*/sifsdata/trader_scores/

Both structured market data and unstructured email data are used for pump-and-dump detection. For accurate detection, ensure that you load the email data before you load the structured data. After structured data is pushed into Hadoop, the pump-and-dump implementation processes this data and

automatically triggers the inference engine. The inference engine considers evidences from both email and structured data analysis to determine the risk score.

Understanding the pump-and-dump analysis results

When the data is loaded into Surveillance Insight for Financial Services, the pump-and-dump rules are triggered and the following files are created on the Hadoop file system:

- Date-wise trade summary data, including moving averages, is created in `/user/sifsuser/sifsdata/ticker_summary/ticker_summary_<date>.csv`
- Date-wise position summary data is created in `/user/sifsuser/sifsdata/positions/top5Positions_<date>.csv` and `/user/sifsuser/sifsdata/position_summary/position_summary_<date>.csv`
- Date-wise pump-and-dump score data is created in `/user/sifsuser/sifsdata/pump_dump/pump_dump_<date>.csv`
- Date-wise trader score data is created in `/user/sifsuser/sifsdata/trader_scores/trader_scores_<date>.csv`

The Spark job for pump-and-dump evidence collection is run for each date. This job collects all of the evidences for the day from Hadoop and populates the following tables in the SIFS database:

- Risk_Evidence
- Evidence_Ticker_Rel
- Evidence_Involved_Party_Rel

The Spark job also runs the inference engine, which applies a risk model and detects whether an alert needs to be generated for the evidence. Based on the result, either a new alert is generated, an existing alert is updated, or no action is taken. The alert information is populated to the following tables:

- Alert
- Alert_Ticker_Rel
- Alert_Involved_Party_Rel
- Alert_Risk_Indicator_Score
- Alert_Evidence_Rel

After the evidence and alert tables are updated, the pump-and-dump alert appears in the dashboard.

Pump-and-dump alerts are long running in that they can span several days to weeks or months. The same alert is updated daily if the risk score does not decay to 0.

The following rules explain when an alert is generated versus when an alert is updated:

1. If no evidence of pump-and-dump activity for a ticker from either structured or unstructured analysis exists, or if the risk score is too low, then no alerts are created.
2. If the inference engine determines that an alert must be created, then an alert is created in the Surveillance Insight database against the ticker. The top 5 traders for the day for that ticker are also associated with the alert.
3. After the alert is created, the alert is updated daily with the following information while the ticker remains in a pump or dump state:
   - New alert risk indicator scores are created for each risk indicator that is identified on the current date.
   - The alert end date is updated to the current date.
   - The alert score is updated if the existing score is less than the new score for the day.
   - The new evidences for the day is linked to the existing alert.

- New parties that are not already on the alert are linked to the alert. New parties would be the top 5 parties for the ticker for the current date.

4. After the alert is created, if the ticker goes into an undecided state, the risk score will start decaying daily. If the score is not 0, the alert is updated as indicated in step 3. For an undecided state, the alert has no pump or dump evidences for the date.

Spoofing detection use case

The spoofing detection use case implementation analyzes market data events and detects spoofing patterns.

A spoofer is a trader who creates a series of bulk buy or sell orders with increasing bid or decreasing ask prices with the intention of misleading the buyers and sellers in a direction that results in a profit for the spoofer. The spoofer cancels the bulk orders before they are completed and then sells or buys the affected stocks at a favorable price that results from the spoofing activity. By analyzing the stock data that is streaming in from the market, the spoofing detection use case detects spoofing activity in near real time.

Triggering the spoofing rules

The spoofing use case implementation requires order, execution, and quote data to detect the spoofing pattern. Loading the data triggers the spoofing rules and updates the alert and score tables in the database.

Understanding the spoofing results

The spoofing use case leverages the Trade Surveillance Toolkit to detect spoofing. It analyzes the market data by looking at the events that are fired by the toolkit and generates alerts if a spoofing pattern is detected. The evidence is then used to determine whether an alert needs to be generated. This decision is made by the inference engine. The alert and the evidence are stored in the Surveillance Insight database by using the REST services.

Spoofing user interface

A spoofing alert appears in the **Alerts** tab.

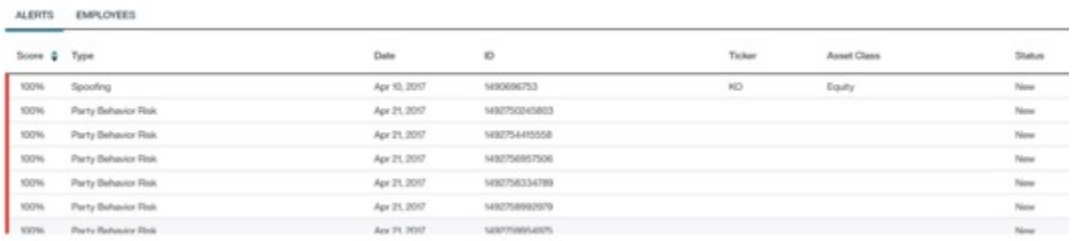


Figure 14: Spoofing alert

Click the alert to see the alert overview and reasoning.

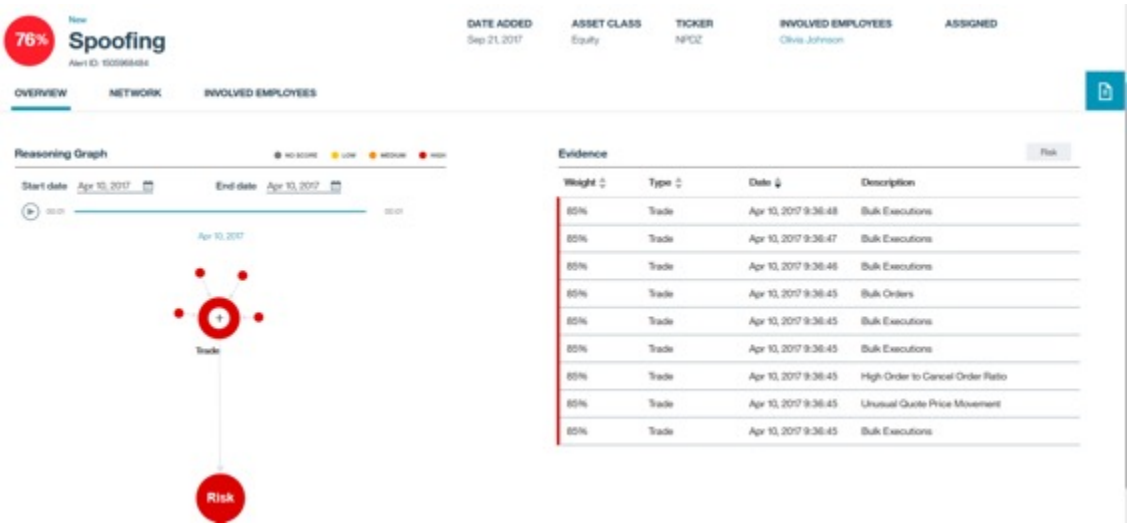


Figure 15: Spoofing overview page

The evidence shows the spoofing pattern where in the bulk orders, unusual quote price movement, and high ratio of orders to cancellation are followed by a series of bulk executions. These evidences contribute to the overall risk as shown in the reasoning graph. In this example, all of the evidences have a 99% weight. This is because for spoofing to happen, each of the events, represented by the risk indicators, should necessarily happen. Otherwise, the pattern would not qualify for spoofing.

Off-market use case

The off-market use case works on forex transaction data and detects transactions that have a deal rate that are outside of the market reference data range at the time of the transaction. The use case combines this evidence with the history of alerts of the trader that are involved in the transaction and decides whether an alert needs to be generated.

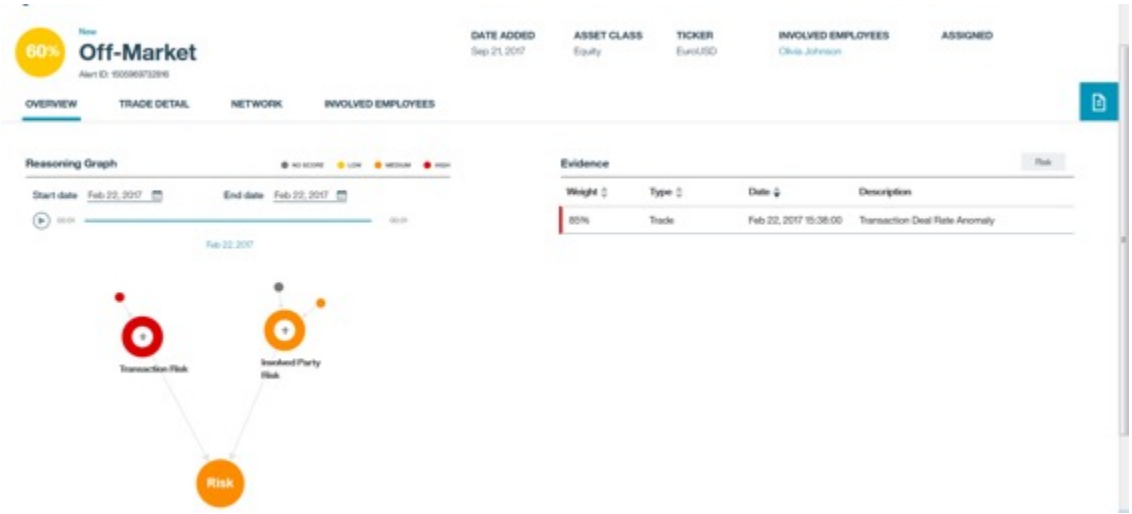
Triggering the off-market use case

The off-market use case implementation requires forex transaction data and market reference data—high and low deal rates for every minute—from the equity market. Loading the data into HDFS triggers the off-market detection.

Understanding the off-market results

An off-market alert appears in the Surveillance Insights dashboard as follows:

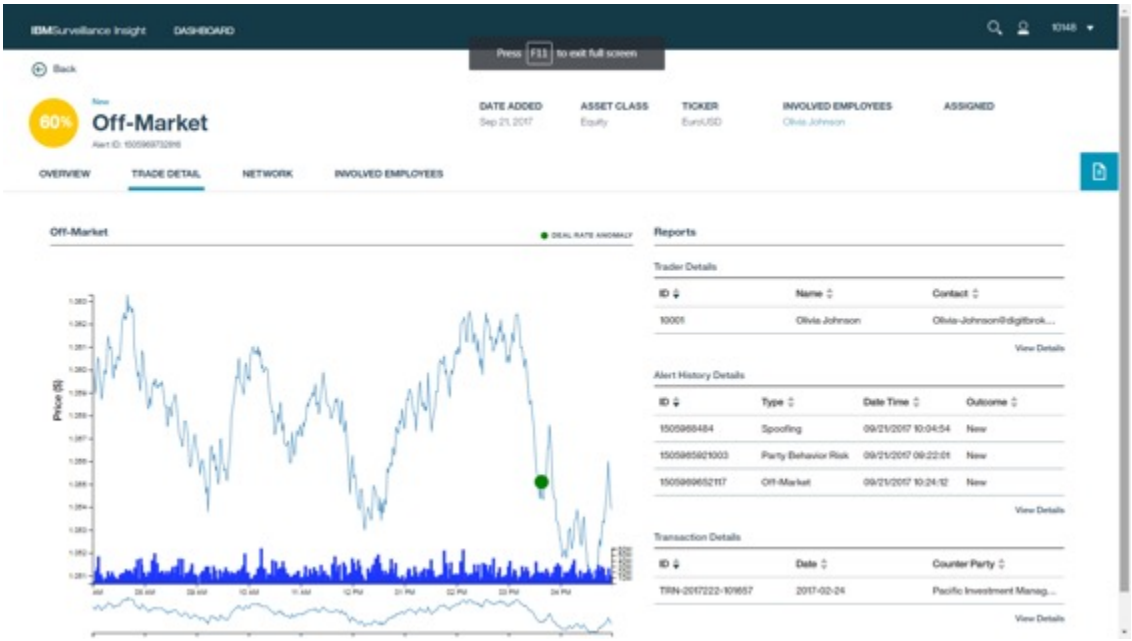
Click the alert to view the off-market alert overview.



As shown in the diagram, the reasoning chart contains two categories of risk indicators: transaction risk and party risk. Each category has one risk indicator:

1. Deal rate anomaly, which indicates that there is an anomaly in the transaction deal rate.
2. Past Alert History, which indicates that the involved party has a history of alerts in the past.

The off-market alert also contains the trade details page that shows the following trade chart:



The trade chart contains the price and volume chart of the transactions and also the point at which the deal rate anomaly occurred, which is indicated by a circle. Further details about the alert history, the transaction, and the party appear in the details sections of the trade report that is displayed on the right side.

Front running use case

The front running use case is designed to detect potential cases where an employee deals ahead of a client. Front running occurs when an employee takes advantage of advance knowledge of large pending orders from a client that has a potential to impact the market price significantly.

Configuring Surveillance Insight to use FTR order data

The front running use case can pull order data from the Financial Transaction Repository (FTR) if the order data is loaded into FTR. Order data is loaded into FTR by using CSV formatted files. To enable reading data from FTR, you must update the following properties in the `/home/sifsuser/lib/sifs.spark.properties` file that is on all of the nodes in the Hadoop cluster.

```
TradeDataSource=FTR
FTRWarehouseDirectory=/apps/hive/warehouse/
FTRHiveMetastoreUris=thrift://<IP>:<PORT>
FTRHiveAddress=jdbc:hive2://
<FTR_IP>:<FTR_PORT>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=<HIVE_SERVERNAME>
```

To access data from Surveillance Insight without going through FTR, set the following property:

```
TradeDataSource=SI
```

Before you can access data from FTR, FTR must be installed and configured properly and the front running data must be loaded into FTR. For more information about ingesting data into FTR, see the FTR documentation.

Triggering the front running use case

The front running risk model is a combination of e-comm and trade risk indicators. The e-comm pipeline must be run as a pre-requisite before you can run the trade pipeline, but the e-comm pipeline is optional. If you do not run it, the e-comm risk indicators are not generated. For more information on the e-comm surveillance pipeline, see Chapter 3, “IBM Electronic Communication Surveillance Analytics,” on page 23.

The following is a list of the front running risk indicators:

- Bulk order
- Front order
- Order proximity
- Intent to use insider information (E-comm)
- Event proximity (E-comm)

The front running use case works on order data and employee data.

1. Run the Trade Data Processor job (`TradeDataProcessor_SI.sh`) to generate the necessary risk indicators for front running.

   Ensure that the order data for the date that needs to be processed is loaded into the following folder in HDFS:

   `/user/sifsuser/order/Order_<yyyy-mm-dd>.csv`

   The schema for the data is as follows:

   - ClOrdID
   - Symbol
   - TransactTime
   - OrderType
   - OrderQty
   - Price
   - Side
   - PartyID

   Refer to the New Single Order specification in FIX 4.4 for a description of the above fields.

Ensure that the `employee.csv` that contains the list of traders who are employees of the brokerage firm in context is in the following location: `/user/sifsuser/trade/employees.csv`

The file contains one column that is named TraderId that contains the traders who are employees.

2. Run the Front Running Inference job (`FrontRunningInference.sh`).

   This job creates the front running alerts, which are displayed in the interface.

Viewing the alerts

Figure 16: Front running alerts

Click an alert to display the details. The following image shows the reasoning graph and the risk evidences that are associated with the alert.

Figure 17: Front running alert details

Click one of the trade evidences or the **Trade Detail** tab to show the trade chart.

Figure 18: Front running trade detail chart

The trade detail chart shows the price and volume chart for the duration of the alert. The annotations on the chart show the front order and bulk order risk indicators that are relevant to this instance of front running. The report section of this page shows the trader details, their alert history, and the actual transactions (orders) that are involved in the current instance.

Extending Trade Surveillance

Solution developers can use the solution's architecture to develop new trade surveillance use cases.

The Surveillance Insight platform is built around the concept of risk indicators, evidences, and alerts. A use case typically identifies a certain type of risk in terms of risk indicators.

One of the first things for any use case implementation on the platform is to identify the risk indicators that are to be detected by the use case. After the risk indicators are identified, the kinds of evidence for the risk must be identified. For example, the indicators might be trade data or email data that showed signs of risk during analysis.

A risk model must be built that uses the evidence so that the inference engine can determine whether an alert must be generated.

The type of alerts that are generated by the use case must also be identified.

The identified risk indicators, the model, and the alert types are then loaded into the The Surveillance Insight database:

- The risk indicators must be populated in the RISK_INDICATOR_MASTER table.
- The risk model must be populated in the RISK_MODEL_MASTER table.
- The alert type must be populated in the ALERT_TYPE_MASTER table.

End-to-end implementation

The following diagram shows the sequence of steps that are involved in developing a new Surveillance Insight for Financial Services (SIFS) trade use case:

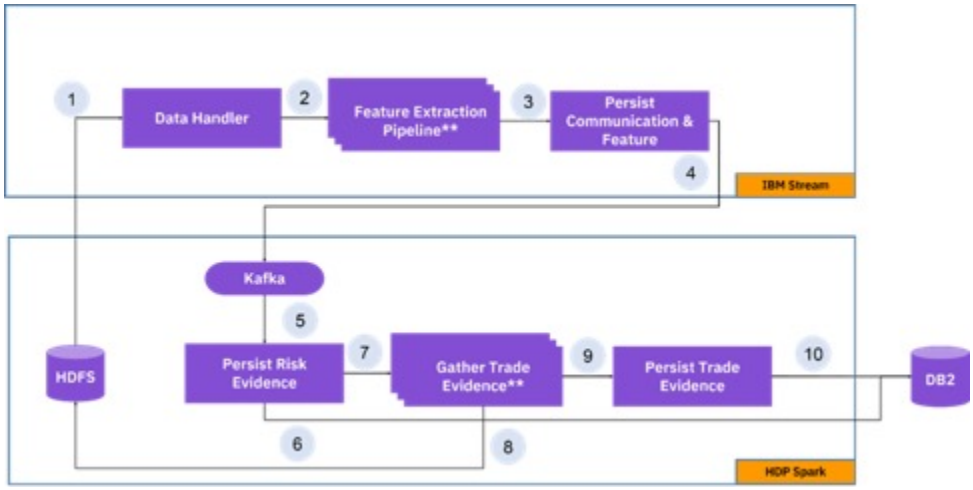


Figure 19: End-to-end implementation for a new use case

1. Data Handler: The DataLoader Streams job that is part of the Surveillance Insights installation monitors certain folders in HDFS. When data is loaded into these folders, the DataLoader picks the content and make it available to the use case implementations.
2. Implementing Risk Indicators: Typically a new use case uses one or more of the existing risk indicators or you must implement new indicators. New risk indicators can be implemented in Streams or in Spark. This decision is made based on what data the risk indicator needs to work on. Typically all indicators that need to work on market data are implemented as Streams operators. They tap into the data that is coming from the DataLoader and perform the necessary risk detection. Operators that need to work on non-market data such as party alert history or proximity of risk indicators should to be

implemented in Spark, and then integrated into the end of the day job that uses all of the evidences and invokes the inference engine.

Implementing a new risk indicator involves implementing the core logic that is involved in reading and analyzing the market data for patterns of interest based on the use case requirements.

a. Understand the event-based approach that is needed to perform trading analytics.

   One of the fundamental principles on which the Trade Surveillance Toolkit operates is generating events that are based on stock market data. The toolkit defines a basic event type that is extensible with event-specific parameters. Different types of stock data, such as orders, quotes, executions, and trade data, are analyzed by the operators in the toolkit. Based on the analysis, these operators generate different types of events.

   The primary benefit of the event-based model is that it allows the specific use case implementation to delegate the basic functions to the toolkit and focus on the events that are relevant. Also, this model allows the events to be generated one time and then reused by other use cases. It also drastically reduces the volume of data that the use case must process.

b. Identify the data types and analytics that are relevant to the use case.

   Identify what data is relevant and what analytics need to be performed on the data. These analytic measures are then used to identify the events that are of interest to the use case.

c. Identify Trading Surveillance Toolkit contents for reuse.

   Map the data types and events that are identified to the contents in the toolkit. The result of this step is a list of data types and operators that are provided by the toolkit.

d. Design the Streams flows by leveraging the toolkit operators.

   This step is specific to the use case that you are implemented.

   In this step, the placement of the Trading Surveillance Toolkit operators in the context of the larger solution is identified. The configuration parameter values for the different operators are identified. Also, data types and operators that are not already present in the toolkit are designed.

e. Implement the use case and generate the relevant risk indicators.

3. Send out a riskEvent to the RiskEvidenceSink operator. This task takes care of generating a risk evidence and dropping the evidence JSON files into a configured Kafka topic.

   It is important to understand the structure and contents of the risk event so that the right information is populated into the SIFS database. For more information about the schema, see “Risk event schema” on page 51.

   Every risk event contains an evidence data list. Each item in the list if of the type tradeEvidence. This field is used by the TradeEvidencePersistence spark job to decide what evidence data needs to be populated in the SIFS database so that the trade charts and evidences can be shown in the Surveillance Insights dashboard.

   For example, a risk indicator that requires evidences of type orders and quotes to be shown in the dashboard requires two evidence data records in the risk event.

   The following is an example of one evidence data record with the type of quote:

```
Datatype  : "quote"
  startTime : indicates from where to start reading the quote.csv file in HDFS
  windowSize : startTime + windowSize (in seconds) is the duration for which records
      will be fetched from the quotes.csv in HDFS.
  list<> id : if there are specific quotes to be shown in the trade charts, provide the
      list of ids here. In this case, the startTime and windowSize need not be provided.
      They will be ignored.Provide empty string for startTime and 0.0 for windowSize.
```

   The second evidence data record is similar, but the type is "`order`".

4. The Trade Evidence Persistence Spark job that is part of the SIFS installation waits for risk evidences on a configured topic. When it receives evidences, it persists the evidence in the SIFS database. It also brings in additional trade evidences that might be required to generate the trade charts in the Surveillance Insights dashboard.

5. Create an end-of-day Spark job that reads all of the relevant evidences for the use case (from the SIFS database) and invokes the inference engine which is a REST API call.

   The inference engine applies the risk model for the use case and determines whether a new alert must be created for the identified risk evidences. If the engine returns a positive result, it create an alert in the Surveillance Insight database by invoking the createAlert REST service.

Summary

A new trade use case requires the following elements:

1. Implementing a Streams job with the required risk indicators and wiring them to the data loader job exports and the risk evidence sink operator.
2. Implementing any risk indicators that are not directly dependent on market data. This is done by using Spark APIs.
3. Implementing an end-of-day Spark job that uses all of the risk evidences and invokes the inference engine. If necessary, the job also creates an alert in the SIFS database.

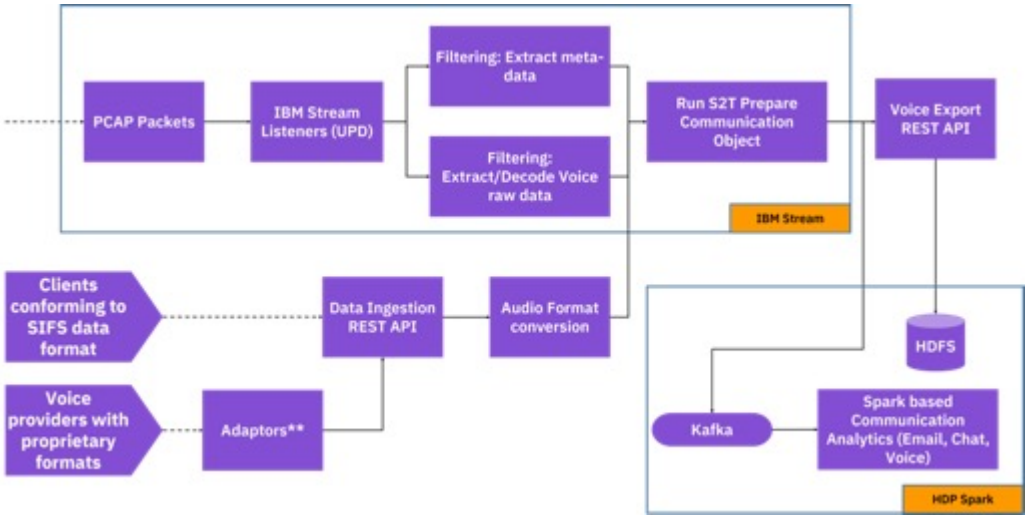
Surveillance Insight for Financial Services provides the following components to help you build your own use cases:

- Surveillance Base Toolkit with reusable types and operators
- Data loader to abstract HDFS folder monitoring
- Evidence persistence job in Spark
- Inference engine APIs
- Alert APIs

Chapter 5. IBM Voice Surveillance Analytics

IBM Voice Surveillance Analytics solution helps identify various risk indicators and alerts associated with voice communication. The voice data files can be directly ingested into voice surveillance system in WAV format. Additionally, the voice surveillance is also capable of reading the data from voice network in PCAP format. The captured voice data is further processed using IBM Watson Speech-to-Text toolkit to generate voice transcripts in plain text format. The generated transcript is then evaluated against various features and different risk indicators are calculated. The risk indicators are then analyzed by the inference engine to detect alarming conditions and generate alerts if needed.

The following diagram shows the different data flow for IBM Voice Surveillance Analytics.



In IBM Voice Surveillance Analytics, the voice data can either be fed through network packets or through the voice data ingestion services.

Flow 1: PCAP format processing obtains the voice data directly from voice network packets and fetches the metadata from Bluewave APIs.

Flow 2: The Voice Ingestion Service and WAVAdaptor processing work in conjunction to process the voice data. For audio files, if the audio file or metadata format is different an adaptor must be built to invoke the voice data ingestion service.

- The Speech to Text operators translate the voice data into transcripts with speaker diarisation.
- The voice artifacts can be optionally exported via the voice data service export interface. It further helps to store metadata, voice transcripts, and audio file into HDFS for both of the above-mentioned flows.
- After the Speech to Text transcript is done, a communication object is then published to the downstream analysis pipeline.
- The generated voice transcript can further be associated with notes, annotations, tags, and evidences from the interface.

Voice Ingestion service

IBM Voice Surveillance Analytics solution processes voice data.

The following formats are supported:

- WAV file in uncompressed PCM, 16-bit little endian, 8 kHz sampling, and mono formats
- PCAP files and direct network port PCAP

The voice ingestion service accepts multipart requests from the user. The multipart requests should contain the following parts:

- A part name "metadata" containing the metadata JSON
- A part name "audiofile" containing the audio binary data

The voice metadata JSON is parsed to get the call start date and gcid values. These values are used to store the audio binary data on the HDFS, where the converted audio file is persisted. The voice metadata JSON is published to the Kafka topic for further processing by the Voice Streams application. The incoming audio file can be WAV, MP3, or OPUS formats. MP4, M4A, M4P, M4B, M4R, and M4V are not supported.

The audio file is converted by using the ffmpeg utility to an uncompressed PCM, 16-bit little endian, 8 kHz sampling mono format WAV file. For example, if an audio file named call001.mp3 is passed to the Voice Ingestion service, the file is converted to cal001.wav and persisted on HDFS.

A sample voice dataset consisting of audio and metadata JSON is provided to help ingest voice files. Use the following command to run the script:

```
./processvoice.sh
```

The following is a sample voice ingestion service multipart request:

```
HEADERS

Content-Type: multipart/form-data; boundary=--------------------
--------e73b4c199aee


BODY

------------------------------e73b4c199aee
Content-Disposition: form-data; name="metadata"; filename="meta.
json"
Content-Type: application/octet-stream

{
    "Initiator": {
        "ContactID": "(+1)-204-353-7282",
        "Name": "Chris Brown",
        "DeviceID": "dev004",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26"
    },
    "Participants": [{
        "ContactID": "(+1)-687-225-8261",
        "Name": "Jaxson Armstrong",
        "DeviceID": "dev002",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26"
    }, {
        "ContactID": "(+1)-395-309-9915",
        "Name": "Henry Bailey",
        "DeviceID": "dev003",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26"
    }],
    "ContactIDType": "phone",
    "AudioFileName": "File1.wav",
    "CallStartTimeStamp": "2017-04-13 11:18:20",
    "CallEndTimeStamp": "2017-04-13 11:19:26",
    "GlobalCommId": "gcid100906524390995"
}

------------------------------e73b4c199aee
Content-Disposition: form-data; name="audiofile"; filename="File
_2.mp3"
Content-Type: application/octet-stream

ID3......vTSS....Logic 10.1.0COM..h.engiTunNORM. 00000284 000002
87 00004435 000043F8 00005A00 00005A00 00007D9C 00007E47 0000715
E 0000715E.COM....engiTunSMPB. 00000000
.........................
.............................................................
------------------------------e73b4c199aee--
```

Voice data services

IBM Voice Surveillance Analytics provides REST API services to allow persisting of voice artifacts and retrieval of audio streams required for playback.

Export REST API service

The Export REST service facilitates the voice streams application to export voice metadata, transcript, and audio file (in WAV format). The default implementation persists these onto HDFS.

The Export REST service accepts a multipart request. The multipart request should contain the following parts:

- A part name "metadata" containing the voice metadata. This part is mandatory.
- A part name "audiofile" containing the audio data. This part is optional.
- A part name "transcript" containing the voice transcript data. This part is optional.

A sample export service multipart request is shown below:

```
Content-Type: multipart/form-data; boundary=--------------------
--------84989e398b28

------------------------------84989e398b28
Content-Disposition: form-data; name="metadata"; filename="metad
ata.json"
Content-Type: application/octet-stream

{
    "Initiator": {
        "ContactID": "(+1)-204-353-7282",
        "Name": "Chris Brown",
        "DeviceID": "dev004",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26"
    },
    "Participants": [{
        "ContactID": "(+1)-687-225-8261",
        "Name": "Jaxson Armstrong",
        "DeviceID": "dev002",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26"
    }, {
        "ContactID": "(+1)-395-309-9915",
        "Name": "Henry Bailey",
        "DeviceID": "dev003",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26"
    }],
    "ContactIDType": "phone",
    "AudioFileName": "File1.wav",
    "CallStartTimeStamp": "2017-04-13 11:18:20",
    "CallEndTimeStamp": "2017-04-13 11:19:26",
    "GlobalCommId": "gcid100906524390995"
}

------------------------------84989e398b28
Content-Disposition: form-data; name="audiofile"; filename="File
1.wav"
Content-Type: application/octet-stream

RIFF.=..WAVEfmt ........@....>......data.<..INFOISFT....Lavf57.7
1.100.data.<..................................................
..............................................................
..............................................................
..............................................................
..............................................................
..............................................................
.............................$.......0.3.'.....!. .*.....$...
------------------------------84989e398b28
Content-Disposition: form-data; name="transcript"; filename="tra
nscript.txt"
Content-Type: text/plain
```

```
{
    "allUtterances": [{
        "utterances": [{
            "SpeakerId": 0,
            "Utterance": "Hello"
        }, {
            "SpeakerId": 1,
            "Utterance": "Hiebert boss Mister mac was calling"
        }, {
            "SpeakerId": 2,
            "Utterance": "Hey Bob how are you"
        }, {
            "SpeakerId": 3,
            "Utterance": "How's the wife"
        }],
        "utteranceStart": 0.83,
        "utteranceEnd": 7.86,
        "utteranceConfidence": 0.7305149015323721
    }, {
        "utterances": [{
            "SpeakerId": 1,
            "Utterance": "She was asking about your day"
        }, {
            "SpeakerId": 2,
            "Utterance": "Ask"
        }, {
            "SpeakerId": 1,
            "Utterance": "You anyway came across some interesting research about cement company"
        }],
        "utteranceStart": 7.86,
        "utteranceEnd": 14.91,
        "utteranceConfidence": 0.6774061718072295
    }, {
        "utterances": [{
            "SpeakerId": 1,
            "Utterance": "Store very good money get into one discuss with you any interest"
        }],
        "utteranceStart": 14.93,
        "utteranceEnd": 20.25,
        "utteranceConfidence": 0.5302127616382294
    }, {
        "utterances": [{
            "SpeakerId": 2,
            "Utterance": "Sure you want to launch we can talking details on"
        }],
        "utteranceStart": 20.26,
        "utteranceEnd": 24.95,
        "utteranceConfidence": 0.7640036944982442
    }, {
        "utterances": [{
            "SpeakerId": 1,
            "Utterance": "Sure we can go for lunch at noon today"
        }],
        "utteranceStart": 24.98,
        "utteranceEnd": 27.07,
        "utteranceConfidence": 0.9037086843769477
    }]
}

------------------------------84989e398b28--
```


Data retrieve REST API service

The Data Retrieve REST service is primarily used by the UI to retrieve voice metadata, the URL of the converted audio file (in WAV format), and the voice transcript from HDFS.

The Data retrieve REST service accepts a communication ID. For a given communication ID, the service fetches the available voice metadata, the voice transcript, and the URL of the converted audio file from HDFS and sends a JSON response to the caller.

The service response JSON data contains:

1. Voice metadata under the field "metadata".
2. The field attribute "audiourl" contains the URL for the audio file.

3. Voice transcript under the field attribute "transcript".

A sample Data Retrieve service multipart response is shown below:


```
{
 "metadata": {
        "Initiator": {
            "ContactID": "(+1)-204-353-7282",
            "Name": "Chris Brown",
            "DeviceID": "dev004",
            "CallStartTimeStamp": "2017-04-13 11:18:20",
            "CallEndTimeStamp": "2017-04-13 11:19:26"
        },
        "Participants": [
            {
            "ContactID": "(+1)-687-225-8261",
            "Name": "Jaxson Armstrong",
            "DeviceID": "dev002",
            "CallStartTimeStamp": "2017-04-13 11:18:20",
            "CallEndTimeStamp": "2017-04-13 11:19:26"
            },
            {
            "ContactID": "(+1)-395-309-9915",
            "Name": "Henry Bailey",
            "DeviceID": "dev003",
            "CallStartTimeStamp": "2017-04-13 11:18:20",
            "CallEndTimeStamp": "2017-04-13 11:19:26"
            }
        ],
        "ContactIDType": "phone",
        "AudioFileName": "File_2.wav",
        "CallStartTimeStamp": "2017-04-13 11:18:20",
        "CallEndTimeStamp": "2017-04-13 11:19:26",
        "GlobalCommId": "gcid100906524390995"
    },
  "audiourl":"/SIFSVoiceDataService/voice/v1/2017-04-13/
c1cd03c1869f73b079ec6e151ba89d04c6b7452f/audio",
"transcript": [
        {
           "Speech": "I",
           "endtime": 22.25,
           "starttime": 2.19,
           "Speaker": "Speaker 0"
        },
        {
           "Speech": "This is Bob joy wells on the call",
           "endtime": 22.25,
           "starttime": 2.19,
           "Speaker": "Speaker 1"
        },
        {
           "Speech": "Merry here",
           "endtime": 22.25,
           "starttime": 2.19,
           "Speaker": "Speaker 0"
        },

        {
           "Speech": "Merry one of you want to take us off strategy",
           "endtime": 35.96,
           "starttime": 22.57,
           "Speaker": "Speaker 1"
        },
        {
           "Speech": "Sure about Stacy I'll play small part orders increase the protocol by
ratio for the strike price of twenty two with an expiration date of five thirty one",
           "endtime": 35.96,
           "starttime": 22.57,
           "Speaker": "Speaker 2"
        }

    ]
}
```

Audio Streaming REST API Service

The Audio Streaming API is used by the UI to allow audio playback of voice evidences in an alert context. The resultant URL can be played through a standard web browser or an audio player that supports audio streaming.

Voice Surveillance Toolkit metadata schema

Table 18: Metadata schema

Field name	Description
Initiator.ContactID	Call initiator's Contact ID. This can be a phone number or a login name.
Initiator.Name	Call initiator's name
Initiator.DeviceID	Device ID from which the call was initiated by initiator.
Initiator.CallStartTimeStamp	The date and time when the call is initiated by the initiator. The value should be in YYYY-MM-DD HH:MM:SS format.
Initiator.CallEndTimeStamp	The date and time when the call is left by initiator. The value should be in YYYY-MM-DD HH:MM:SS format.
Participants.ContactID	Participant’s Contact ID. This can be a phone number, IP address, or a login name.
Participants.Name	Name of the participant.
Participants.DeviceID	Device ID from which the call was initiated by Participant.
Participants.CallStartTimeStamp	The date and time when the call is joined by the participant. The value should be in YYYY-MM-DD HH:MM:SS format.
Participants.CallEndTimeStamp	The date and time when the call is left by participant. The value should be in YYYY-MM-DD HH:MM:SS format.
ContactIDType	Allowed values are "loginname" and "phone". The value "phone" is set when a user is identified by their phone number. The value "loginname" is set when a user is identified by their login names, for example, while they are using Cloud9 Trader.
AudioFileName	Audio file name that needs to be processed.
CallStartTimeStamp	The date and time when the call is initiated. The value should be in YYYY-MM-DD HH:MM:SS format.
CallEndTimeStamp	The date and time when the call is ended. The value should be in YYYY-MM-DD HH:MM:SS format.
GlobalCommId	Unique global communication ID attached to this audio.

WAV adaptor processing

IBM Voice Surveillance Analytics solution processes WAV files based on the metadata trigger that is received through pre-defined Kafka topics. The WAV adaptor reads the data from the Kafka topic, decrypts the Kafka message, parses it, and fetches the voice audio file location. The audio content is then passed to the SpeechToText (S2T) toolkit operator for translation. All of the utterances and the speaker diarization are aggregated. The aggregated conversation text is then converted to a communication object and then published to the Kafka topic.

Also, if an export URL is configured, the voice artifacts—the metadata, utterances, and the audio binary—are sent to the export service.

The export capability allows you to export individual voice artifacts to different endpoints. You can specify the following parameters when you submit the Streams job:

- To export all of the voice-related artifacts to the HDFS on hostname1:

```
EXPORTALLURL=https://<hostname1>:<port>/SIFSVoiceDataService/voice/v1/export
```

- To export the voice metadata-related artifacts to the HDFS on hostname2:

```
EXPORTMETADATAURL=https://<hostname2>:<port>/SIFSVoiceDataService/voice/v1/export
```

- To export the voice metadata and transcript to the HDFS on hostname3:

```
HDFS.EXPORTTRANSCRIPTURL=https://<hostname3>:<port>/SIFSVoiceDataService/voice/v1/export
```

- To export the voice metadata and audio data to the HDFS on hostname2 and hostname4:

```
EXPORTAUDIOURL=https://<host2>:<port>/SIFSVoiceDataService/voice/v1/export;https://
<hostname4>:<port>/SIFSVoiceDataService/voice/v1/export
```

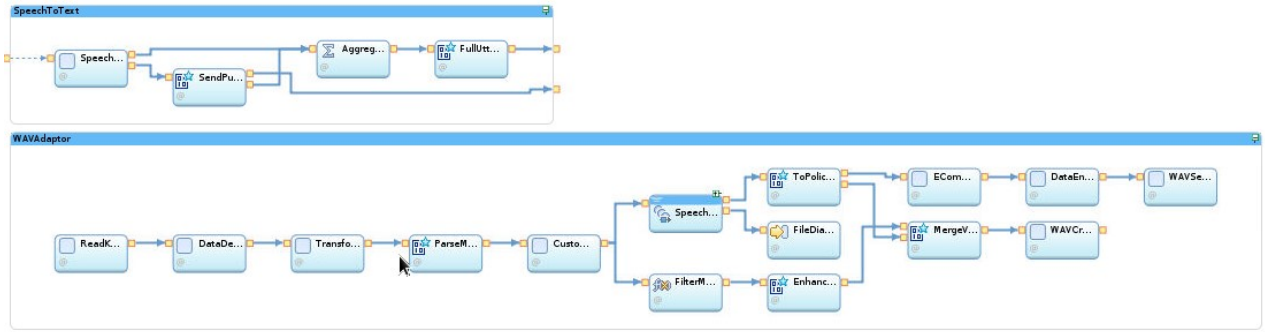


Figure 20: WAVAdaptor Streams Job

PCAP format processing

Processing voice data from network involves speech extraction from network packets and IPC-based call metadata extraction.

Networktap job

This job sniffs the network packets from the defined network interface, and then takes the received packets and transfers them to the downstream job. The Standalone job connects to the network interface card by using the IBM Streams Network Toolkit's PacketLiveSource operator. This operator puts the network interface into promiscuous mode to enable gathering of all network packets. The packets are then forwarded to the downstream PCAP Adaptor job by using the IBM Streams Standard Toolkit's TcpSink operator.

Figure 21: Networktap Streams job

PCAP Adaptor job

The PCAP Adaptor job parses PCAP data from a network port. The raw packet data is also exported to the IPC job. Packets are filtered based on IP addresses, subnets, or by login names. The filtered RTP packets are processed and all of the audio packet data that is collected for a given call is exported to the RouteSpeech job. Certain call attributes, such as the callid, channel_id, source, and destination port, are exported to the CorrelateCallMetadata job

Figure 22: PCAP Adaptor job

IPC job

The IPC metadata extraction jobs consists of two SPLs: IPC and CorrelateCallMetadata. The IPC job receives raw socket data from the PCAP Adaptor job. It identifies the SIP Invite messages of new user logins to their turret devices. It then parses the XML data packets to fetch the device ID and session ID that corresponds to the handsets and stores it in an internal monitored list. This is done to avoid monitoring audio data from speaker ports. After the SIP ACK messages are received, it verifies that the device ID from the ACK is present in the monitored list. It then emits the device ID and the destination port.

Figure 23: IPCS Streams job

CorrelateCallMetadata job

The CorrelateCallMetadata job uses Bluewave's (BW) LogonSession API to prepare a list of all of the users who are logged on to the voice network. From the LogonSession response XML, certain attributes about the users, such as their IP address, loginname, userid, zoneid, zonename, loginname, firstname, lastname, and emailid are extracted and cached. Subsequently, for users who log in , their corresponding device IDs are sent by the IPC job. For the incoming device ID, the LogonSession details are fetched from the BW API and the user list in the cache is updated.

Any access to BW needs an authentication token. After an authentication token is created, it is refreshed at regular intervals. Also at regular intervals, a call is made to get the communication history for the last 5

seconds. This is compared with the call records that are extracted from the RTP packets based on the loginname and call start and end times. If the call timing of the communication history record and of RTP packets are within a tolerable deviation, then that communication history record is assigned as the metadata record for the call that is identified in the RTP packets. The identified metadata record is then exported to the RouteSpeech job.

Figure 24: CorrelateCallMetadata Streams job

RouteSpeech job

The RouteSpeech receives audio packets and metadata as tuples. In an organization's voice network, calls emanate from different departments and each department may have vocabulary that is specific to its business. As a result, calls must be routed through specific Speech to Text language models that are based on the source of the call, for example, calls from the Foreign Exchange department might be routed through a S2T language model that was developed specifically for Foreign Exchange whereas calls from the Equity team are routed through a S2T language model that was developed for Equity. This allows a better accuracy rate in speech recognition. Based on the department of the loginname that is associated with the call, the raw speech files are created in a directory that is assigned to a route. Metadata tuples are updated with the partyid and exported to ProcessMetadata job.

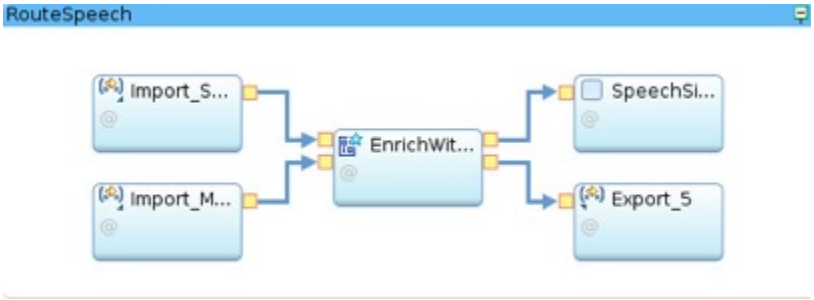


Figure 25: RouteSpeech Streams job

PCAPSpeech job

This job contains the SpeechToText operators for processing audio data from the raw speech files that are created in the RouteSpeech job. After the S2T is complete, it checks if metadata for the concerned call is available. If metadata is available, it is correlated with the converted text from the call and a CommData object is created and published to Kafka. Also, if an export URL is configured, the voice artifacts—the metadata, utterances, and the audio binary—are sent to the export service. The default export service persists the artifacts to the HDFS file system. If the metadata is not available, the audio binary and utterances are persisted to HDFS.

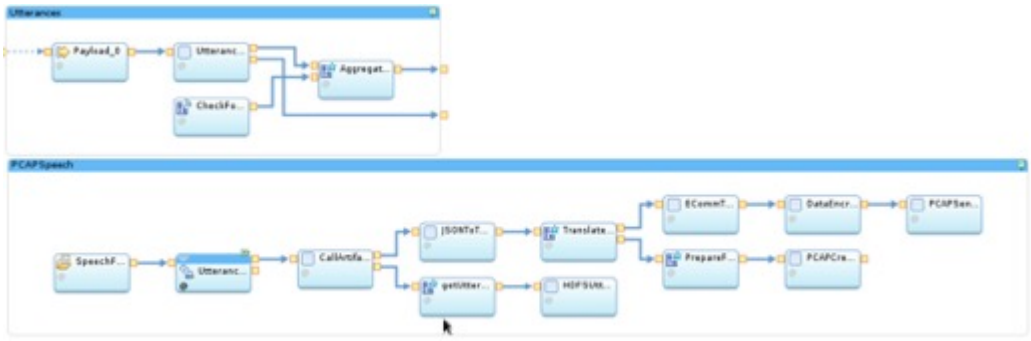


Figure 26: PCAPSpeech Streams job

ProcessMetadata job

The ProcessMetadata job consumes the metadata tuples that are sent by the RouteSpeech job. It checks if transcripts for the concerned call is available. If a transcript is available, it is correlated with metadata and a CommData object is created and published to Kafka. Also, if an export URL is configured, the voice artifacts—the metadata, utterances, and the audio binary—are sent to the export service. The default export service persists the artifacts to the HDFS file system. If a transcript for the call is not available, the metadata is persisted to HDFS.

Figure 27: ProcessMetadata Streams job

Chapter 6. IBM Complaints Analytics

IBM Complaints Analytics provides the components to build a solution that analyzes customer complaints and identifies trends in the complaints with respect to the domain-specific theme of the complaint, the product, and sub-product to which the complaint corresponds, the geography, and the age of the complaining customer.

The key components of the conduct surveillance solution are:

- The Analysis Pipeline
- Trend Analysis Component
- Complaints Dashboard and Supporting Data Services

Figure 28: Conduct Surveillance workflow

The raw data is analyzed through a pipeline of machine learning models that output the features of each complaint, such as the theme, sub-theme, product, and sub-product, to which the complaint belongs. These features, along with the customer age and geography, are then fed as inputs to the trend analysis component that identifies trends across the various combinations of these features. The results of the trend analysis are published to a web-based dashboard for the compliance officers to review.

Data ingestion

Raw complaints data is typically in CSV format. The schema of this data is not restricted by IBM Surveillance Insight for Financial Services. The analysis pipeline implementation must provide a schema transformation stage to map the input fields to the complaints features schema that is expected by the trend analysis component.

The sample analysis pipeline that is provided with IBM Surveillance Insight for Financial Services uses a specific schema.

The raw data is expected to be in loaded into the HDFS for the analysis pipeline to consume as a Spark data set. To leverage the pipeline stages and API that are provided by Surveillance Insight, the data must be loaded into Surveillance Insight and read as a Spark data set. However, this is not a constraint if a non-Spark based pipeline implementation is the preferred way to implement the pipeline. As long as the complaints features are generated in HDFS in the required schema, the pipeline can be implemented using other technologies.

Analysis pipeline

The analysis pipeline processes complaint text through a series of machine learning models that analyze the textual content and identify whether it is a complaint, what keywords of interest are present in the complaint, and what theme (or category) of interest that the complaint belongs to.

Input to the pipeline is raw data and the output are complaint features that are stored in the database. The schema or the complaint features are detailed in the complaint features section.

IBM Complaints Analytics also provides an API, that is a wrapper for the Spark ML APIs, to create custom pipelines and a sample implementation of a pipeline that uses the publicly available CFPB data.

Note: IBM Complaints Analytics does not provide machine learning models as part of the product. These models must be created as part of the specific engagement.

Create an analysis pipeline

Complaints pipeline API

IBM Surveillance Insight for Financial Services provides a wrapper API to create Spark pipelines. Typical usage of the API is as follows:

- Create a pipeline:

```
public ComplaintsPipeline(SparkSession spark, HashMap<String,String> propertiesMap)
```

  The propertiesMap is expected to hold the following properties that are read from the `sifs.spark.properties` file:

```
HDFSComplaintsPath=complaints
WatsonNLCThemeClassifierREST=https://gateway.watsonplatform.net/natural-language-
classifier/api/v1/classifiers/<THEME_CLASSFIER_MODEL_ID>/classify
WatsonNLCComplaintsClassifierREST=https://gateway.watsonplatform.net/natural-language-
classifier/api/v1/classifiers/<COMPLAINTS_CLASSIFIER_MODEL_ID>/classify
WatsonNLCCredentials=<nlc_username>:<nlc_password>
```

```
WatsonNLUREST=https://gateway.watsonplatform.net/natural-language-understanding/api/v1/
analyze?version=2017-02-27
WatsonNLUModelID=alchemy
WatsonNLUCredentials=<nlu_username>:<nlu_password>
```

```
SolrREST=https://<IP>:8984/solr/complaints/update?commit=true
```

```
ComplaintTrendsREST=https://<IP>:9443/complaintsservices/surveillance/v1/ complaint/
createTrend
ComplaintThreshold=0.5
ComplaintSentenceModel=/home/sifsuser/models/en-sent.bin
keywordsServiceUrl=https://<IP>:<PORT>/analytics/models/v1/get_keywords/
```

- Add stages to the pipeline:

```
public boolean addStage(Transformer pipelineStage)
```

  A pipeline stage is implemented as a Spark Transformer and passed to the API.

- Run the pipeline:

```
public Dataset<Row> run(Dataset<Row> complaintsDS)
```

  Calling the run method starts the execution of the pipeline with the configured stages.

For more information about creating transformers by using the Spark API, see the Spark documentation.

Reusable pipeline stages

IBM Surveillance Insight for Financial Services provides the following Spark transformers that you can use as stages in an analysis pipeline:

- ComplaintsNLUStage
  - Provides an implementation that calls the Watson Natural Language Understanding (NLU) REST service to identify entities of interest in the complaint text for each complaint. This implementation expects the NLU model to return the complaint and process the entities by using the naming convention "Complaint_xxxx" and "Process_yyy", where xxx and yyy are the category and process mapping for the complaint. The implementation also interprets relationships between a category and a process. This is expected to appear in the "relation" part of the Watson NLU service response.
  - Input: Complaint features dataset with the raw complaint text and with the IS_COMPLAINT flag set.

  - Output: Complaint features dataset with the NLU_Response field set to the response from the NLU service.
- ComplaintClassifierStage
  - Provides an implementation that calls the Watson Natural Language Classifier (NLC) REST service to identify whether a piece of text is a complaint. This implementation breaks the complaint text into chunks of 1024 characters and passes them to the NLC service. The first chunk that qualifies to be a complaint (that is, crosses the configurable ComplaintThreshold parameter), stops the processing of further chunks. This implementation expects the NLC to return a "Complaint" or "Non-Complaint" class.
  - Input: Complaint features dataset with the raw complaint text and the IS_COMPLAINT flag set (see schema section for details). The model that is required to perform the classification is expected to be created and configured as input to this stage. The model is not provided as part of the product.
  - Output: Complaint features dataset with the THEME field set to the response from the NLC service.
- KeywordsDetectionStage
  - Reads the complaint text and extracts the necessary keywords from the text. The keyword machine learning service is explained in the Machine Learning section
  - Input: Complaint features dataset with the raw data
  - Output: Complaint features dataset with the KEYWORDS field set to the response from the keyword service

Reusable utilities

The following persistence components are available for reuse for implementing a pipeline:

- ComplaintsPersistence: This component provides methods for persisting the complaint features to HDFS and the Complaints table in Db2.
- ComplaintsSolrDataLoader: This component provides a method to persist the complaint features to Solr. This is to enable the search of complaint features through the complaints dashboard. The complaints Solr schema is explained in the data schemas section.

Sample pipeline implementation

IBM Surveillance Insights product provides a sample implementation of an analysis pipeline that uses the following stages, in order:

1. Schema Adaptor
2. Complaint Classifier
3. Complaints NLU
4. Keyword Extraction

The sample implementation provides an example of a typical pipeline design for complaints analysis. It contains a schema adaptor that maps the contents of the raw data to the complaint features schema. This step creates a dataset that passes through all the stages in the pipeline. Each stage then completes the necessary fields based on its purpose.

In the case of the sample implementation, the Complaint Classifier runs an NLC model that classifies the text as complaint or non-complaint and sets the IS_COMPLAINT flag in the output dataset.

The NLU stage populates the results from the Watson NLU service into the NLU_Response field in the incoming dataset and also the Theme and Process fields.

The keyword extraction step extracts keywords from the complaint text and populates the results in the KEYWORDS field.

The persistence step saves the outcome of the pipeline, which is the complaint features dataset, to the HDFS. It also persists part of the response to the complaint table in Db2. The complaint text and some other related information are persisted to Solr. The schemas are detailed in the schema section. The persistence step is not implemented as a pipeline stage.

The header for the raw data used for the sample pipeline is as follows:

```
COMPLAINT_DATE,PRODUCT,SUB_PRODUCT,COMPLAINT_TEXT,GEO,REFERENCE_ID,CUSTOMER_ID
```

The data itself is taken from the publicly available CFPB database. The sample data contains 3 months of complaints for the Mortgage product.

An NLC model must be created for the Watson Bluemix NLC Service by using the sample CFPB data. The model ID must be configured in the `sifs.spark.properties` file. The same is true with the NLU service. The output results are saved to HDFS in the path that is configured in the `sifs.spark.properties` file appended with the path that the user provided while running the pipeline. Default path where features data is stored in HDFS is `/user/sifsuser/complaints/complaint_features`.

Complaint features

The complaint features file is persisted to the HDFS as a CSV file. It contains the following data for each record that is processed through the pipeline. Every record may or may not be a complaint.

The complaint feature file is in CSV format and it contains the following data for each record that was processed through the pipeline. Each record might or might not be a complaint.

- complaint_id — system generated id
- reference_id — id of the complaint in the source file
- complaint_date — date mentioned in the complaint or date email was received
- theme_confidence_score — as returned by NLC model
- theme — as returned by NLC model
- sub-theme — as returned by the NLC model
- product — as returned by the NLU model
- sub-product — as returned by the NLU model
- product_ref — product reference in raw data
- sub-product_ref — sub-product reference in raw data
- geo — geo reference in raw data
- channel — channel reference in raw data
- customer_id — customer id reference in raw data
- customer_age — customer age reference in raw data
- customer_gender — customer gender reference in raw data
- customer_race — customer race reference in raw data
- is_complaint — true/false as returned by the NLC model
- is_complaint_score — as returned by the NLC model
- NLU_Response — as-is response of the NLU model
- Subject — subject of the complaint from raw data
- from_mailid — valid only for email complaints
- to_mailid — valid only for email complaints
- Tone_Analyzer_Response — not in use
- Keywords — as is response from the keywords service

Trend analysis

A trend is a change in the normal number of expected complaints. A trend is a consistent upward or downward movement, out of the ordinary range.

For more information about trends, see “Trend detection” on page 79.

Chapter 7. Machine learning

The solution offers pre-built custom libraries for some machine learning tasks.

The following pre-built libraries are provided:

- Trend detection
- On-line trend detection
- Extracting Keywords
- Parsing e-mails
- Sampling e-mails
- Unsupervised learning
- Natural Language Classifier model
- Natural Language Understanding model
- Discovery
- Entity Extraction

The solution uses open source frameworks and libraries, such as tensorflow and Keras.

Trend detection

For the purposes of Surveillance Insight complaint trend detection, a trend is a defined as a significant and prolonged change in the number of complaints per day that you would expect to see. The trend detection algorithm identifies all such shifts in the complaint counts, either up or down, and presents these in an easy to read user interface.

The following diagram shows one example of a trend: the number of complaints per day for one complaint theme (or complaint type) has been plotted for 30 days. This is trending because the number of complaints started increasing rapidly after June 4th, over and above the normal historic variability.

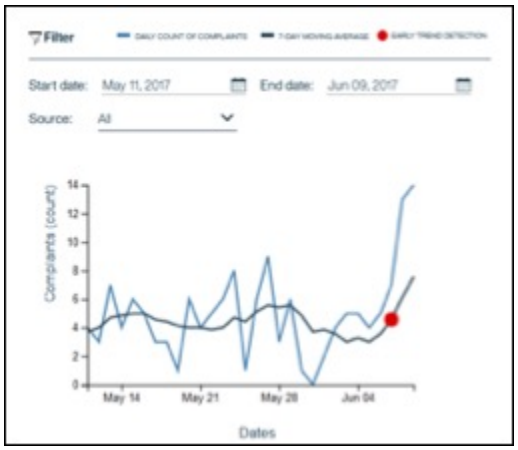


Figure 29: Trend detection example

The lighter blue line is the raw amount of complaints per day, and is very jagged (noisy) due to the number of complaints varying between weekdays and weekends. To make it easier to understand, Surveillance Insights plots a smoothed line (dark blue line) by taking the average over the previous 7 days on each day. This removes the noise caused by weekdays/weekends. The red dot indicates the day on which the detected trend started.

Business problem

Financial institutions require a method of identifying unusual increases in complaints, and in identifying specific causes of those complaints outside of the usual internal methods. In particular, a method is required for finding sudden increases in the number of complaints that are related to a specific theme that may help in the early detection of a business problem or a customer pain-point.

Approach to solving the business problem

Surveillance Insight uses a combinatorial and statistical approach to systematically check and find groups of related trends that show trending behavior.

All complaints are classified by the Natural Language Classifier and Natural Language Understanding models, and the results are combined with customer data. The result is a set of data points for each complaint such as category, process, product, theme, sub-product, age, zip code, state, gender etc.

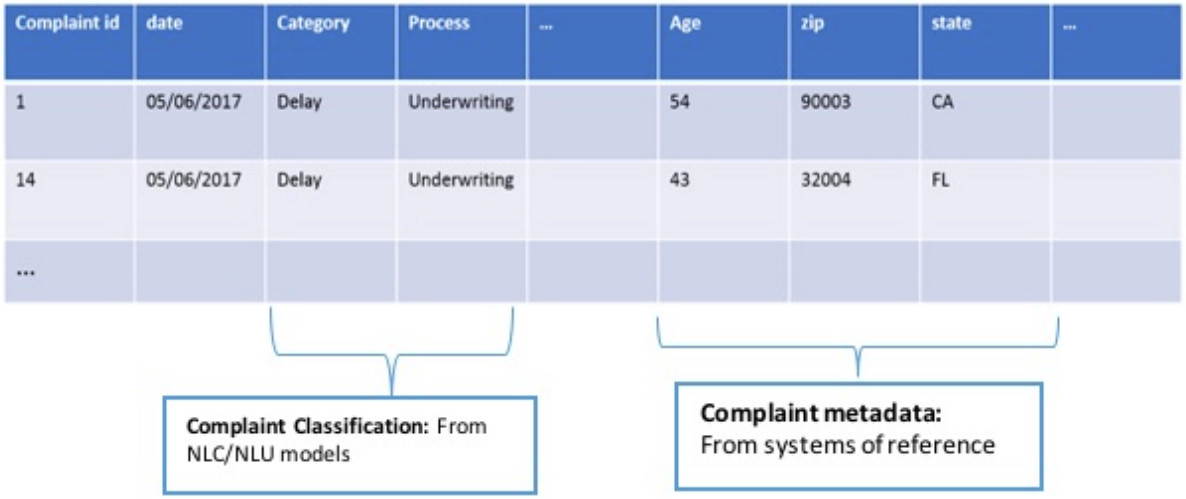


Figure 30: Grouped complaints

The trend detection algorithm groups all of the complaints according to these features. Then the algorithm calculates all of the combinations of these features, and then checks each one to see if a trend can be detected.

For example, the trend detection algorithm derives all of the combinations of these fields:

- complaint category
- complaint category + process
- complaint category + process + zip code

This forms a hierarchy of complaint types: the top level is very generic (complaint category only) but more specific details appear as further columns are added. For example:

- complaint category = ‘burdensome request’, ...
- complaint category = ‘burdensome request’ + process = ‘Mortgage Application”,
- complaint category = ‘burdensome request’ + process = ‘Mortgage Application” + zip code = 90410

The trend detection algorithm then counts the complaints in each combination of feature-value pairs over periods of 30, 60, and 90 days. Each count is evaluated to see if a trend can be found using a trend detection rule. This method enables the detection of cross-feature trends that would otherwise be hard to detect; for example, complaints about loan applications for customers younger than 30 in the mid-western states.

Trend detection rule

Different statistical methods are used to identify trends:

- A Mann-Kendall score is a statistical assessment of whether there is a consistent upward or downward trend in the count over time.
- A Poisson Distribution Model is a statistical model that is used to evaluate how unusual a point in a time-series is when it is compared to previous points. It helps identify sudden increases (or decreases) in the number of complaints that may help indicate the presence of a trend.

These statistical measures are combined to determine if a trend is occurring:

- a significant and sustained increase (or decrease) in the count
- at least one day where the count is unexpectedly large or small given the preceding counts (using a Poisson distribution model)
- at least one day greater than the average count
- a clearly defined start and end
- for an increasing trend, the end-count must be greater than the start-count
- for a decreasing trend, the end-count must be less than the start-count

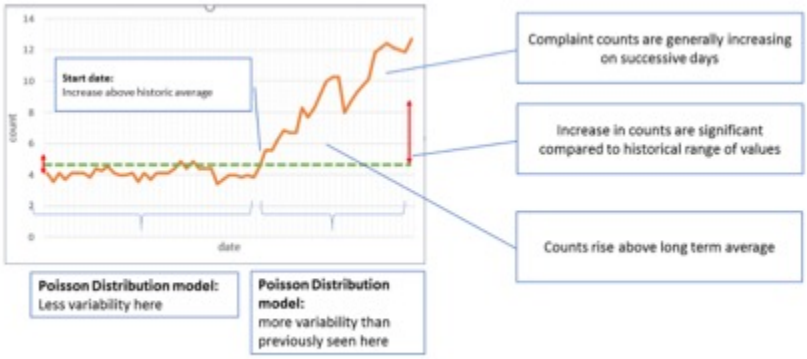


Figure 31: Diagram showing the trend detection rule

The parameters that Surveillance Insight uses to detect trends can be tuned for each customer in the configuration files. The configurable parameters include:

- Relative importance of rate of increase or decrease in a trend
- Relative importance of absolute count of complaint
- Relative importance of consistency of trend: once a trend starts, does it only increase or does it fluctuate?
- Duration of trend

Early trend detection

The trend detection algorithm also shows the date on which a time-series began to show a trend. This is called the “early trend detection” date. An example is shown by the red dot in the following diagram.

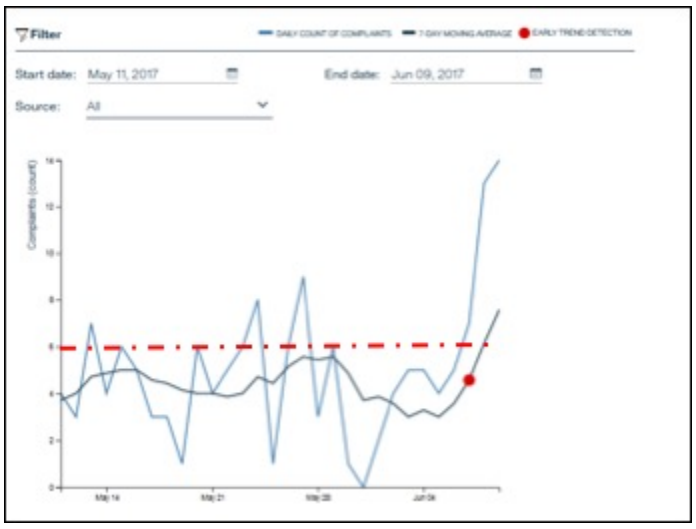


Figure 32: Early trend detection

To explain this better, the chart above also shows a horizontal dotted red line which shows the average number of complaints per day for this combination for the whole 30-day period. The dotted line does not appear in the Surveillance Insight, it is only used for explanation here.

The red dot shows the earliest date on which a trend was detected because:

- All counts after this date are increasing: for example, there is a trend
- The red dot shows the first date (in raw count) with a count above the 30-day average
- Early Trend Detection is a way of indicating “the trend started on this date...”

Assumptions

Trends are derived based on aggregate data from the complaints meta-data.

The NLC and NLU models which create the meta-data (complaint type, process etc.) have accurately classified the complaints in most of the cases.

Accuracy and limitations

There is no absolute definition of a trend.

Trend detection is a compromise between detecting significant (but possibly small) trends, and ignoring noise due to natural (i.e. random) variability.

Trends may vary widely in terms of number of complaints per day, speed of increase in count per day and other factors. It is difficult to find a “one-size fits all” method that will discover trends for themes at a high level general view (e.g. a countrywide perspective) and also for low level highly specific characteristics. For example, there may tens of thousands of complaints related to ‘delay’ while the count of complaints related to ‘poor customer service for female customers over 50 in New Mexico’ may only contain tens of complaints.

On-demand trend detection

The on-demand trend algorithm integrates with the Surveillance Insight Complaints Explore page and is also available as a stand-alone REST API.

A user can determine features of interest, such as complaint category, process, product, customer age, and geography, and set their corresponding values. When sent to the REST API, the on-demand service

produces a time-series of counts of complaints per day for the date range given and other key statistics useful in determining whether a trend is present.

Business problem

The Surveillance Insight trend detection algorithm runs on a periodic basis to determine if trends are occurring across all dimensions and at all levels of granularity, and then displays all of the discovered trends in the trend detection UI. Aside from this, users may want to independently query the trends database to check for trends. This might be for several reasons:

- To visually determine the normal baseline level of complaints for a given set of complaint features
- To combine complaint features in arbitrary combinations that are not checked as standard
- To confirm that a previously detected trend has now ceased to trend for example because remedial action has been taken
- To determine the effect of complaint and entity labeling in between scheduled trend-detection runs.

Approach to solving the business problem

A lightweight, on-demand version of the trend detection algorithm is provided via a REST API. The main difference between the trend detection algorithm and the on-demand trend detection algorithm is that the on-demand trend detection is designed to allow a user to investigate one time-series of complaint counts that may or may not show a trend. The following table summarizes the differences.

Table 19: Trend detection vs On-demand trend detection

Trend detection algorithm	**On-demand trend detection**
Automatically generates all combinations of available features	User manually inputs one set of features of interest
Algorithm finds only those combinations of features that are trending, and ignores non-trending combinations	Returned time-series may be trending or non-trending
Variable number of trends may be detected, depending on actual number of trends present in data, sensitivity of trend detection parameters in configuration etc.	Always returns one time-series corresponding to the selection criteria (provided there are entries in the database)

Using the REST service

Starting the REST Service

```
python3 onDemandTrendDetectionRestAPI.py
```

Sample input

```
THEME = Complaint Delay, CUSTOMER_AGE = 30-50, GEO": TX and FL
```

Sample response

Table 20: Sample response example		
Date	**Complaint count**	**Smooth complaint count**
2018-03-08	14	14.0
2018-03-09	16	15.0
...	...	...
2018-03-15	27	18.3

Service details

The service allows users to determine if a given set of complaint features is trending.

Method
POST

URL
/analytics/models/v1/on_demand_trend/

Input
JSON payload

Output
JSON response

The following is an example CURL command to POST:

```
curl -k -H 'Content-Type: application/json' -X POST --data-binary @query.json https://
ip_address:port/analytics/models/v1/on_demand_trend/
```

The following code is an example JSON payload:

```
{"query" : {"THEME": ["Complaint_Delay"], "CUSTOMER_AGE": ["30-40", "40-50"], "GEO":
["TX","FL"]}}
```

Note: Dates are optional.

Either set the start-date and end-date directly by including the parameters in the JSON payload, for example, “STARTDATE”: “2017-05-01”, “ENDDATE”: “2017-06-12”.

Or, you can send no dates, in which case the end-date is set to the current date, and the start-date is set to be 365 days prior (using configuration parameter time periods.)

The response is a JSON structure that gives the complaint counts per day and other trend statistics. The following code is an example response:

```
{"trends":{"description": "CUSTOMER_AGE:30-40&THEME:Complaint_SalesPractice", "complaintcount":
{"trendinglast30days": "+4.6%", "30dayavg": 21.3, "7dayavg": 19.1, "earlytrend": null,
"trendinglast7days": "+0.0%", "dates": [{"date": "2018-03-08", "smoothcount": 14.0,
"actualcount": 14}, {"date": "2018-03-09", "smoothcount": 15.0, "actualcount": 16}, {"date":
"2018-03-10", "smoothcount": 15.3, "actualcount": 16}, {"date": "2018-03-11", "smoothcount":
15.7, "actualcount": 23}, {"date": "2018-03-12", "smoothcount": 16.1, "actualcount": 19},
{"date": "2018-03-13", "smoothcount": 16.4, "actualcount": 20}, {"date": "2018-03-14",
"smoothcount": 17.3, "actualcount": 27}]}, "product": "all", "riskscore": 24.5, "timeperiod":
7, "theme": "Complaint_SalesPractice", "trendtypeid": 3, "geo": "all", "trendattributes":
[{"MAX_AGE": 40}, {"THEME": "Complaint_SalesPractice"}, {"MIN_AGE": 30}], "trendstartdate":
null}}
```

Assumptions

The NLC and NLU models that create the meta-data (complaint type, process, etc.) have accurately classified complaints in the majority of cases.

Accuracy and limitations

On-demand trend detection does not work for keywords.

Keyword extraction

The e-communication keyword extraction service is used to extract key words and phrases from text, such as an email or chat. The algorithm parses the text into sentences and removes the most frequent but least useful words for determining meaning (stop-words). It then applies various statistical and frequency methods to determine the most significant key words and phrases.

Business problem

Business texts, such as emails, can be long and wordy. It is useful to see a list of key words and phrases to quickly assess the validity, subject, and themes of a text and its classification.

Approach to solving the business problem

This service uses an implementation of the RAKE Automatic Keyword Extraction from Individual Documents Algorithm. This is a domain-independent method for automatically extracting keywords, as sequences of one or more words, from individual documents. RAKE can be applied to individual documents and does not need to see the whole corpus, unlike term-frequency or inverse document frequency, for example.

For more information on this algorithm, see https://www.researchgate.net/publication/227988510_Automatic_Keyword_Extraction_from_Individual_Documents.

You can configure the algorithm with the following parameters:

- The maximum word length (in characters) for any word
- The number of key words or phrases required for each text. For example, the default value is 10 keywords/phrases per text
- The maximum length of any key phrase

Assumptions

The algorithm currently works only for English text.

The algorithm uses punctuation and sentence structure to determine keywords so the text should not have been cleaned of these before use.

A list of stop-words is used to remove the most common words that do not act as keywords. This is provided as part of Surveillance Insight, and it may be necessary to amend and supplement this list.

Using the REST service

The service allows users to derive keywords from a text.

Starting the REST Service

```
python3 keywordsRESTAPI.py &
```

Sample input

```
Just a short message to apprise every one of the cleaning results performed on the filter
separator from the Transwestern M/S. Wipe tests taken of the interior portions of the
separator prior to cleaning revealed PCB results of: 360, 500 and 520 microgram. This was
not a standard wipe taken of a specific area. These wipes were taken over large areas to
determine presence/absence of PCB's within the separator. After the cleaning operation was
completed, three wipe samples were taken of the interior portions of the separator. All
results showed non- detect at less than 1.0 microgram. I feel we can be fairly certain that
the separator is free of residual PCBs. Jeff, would you pass this information on to your
counterpart for PG&E?
```

Sample response

```
cleaning revealed pcb results, cleaning results performed, results showed, pcb, cleaning
operation, wipe tests, wipe samples, standard wipe, specific area, short message
```

Service details

Table 21: get_keywords service details

Method	URL	Input	Output
POST	/analytics/models/v1/get_keywords	JSON payload	JSON response

The following is an example CURL command to POST:

```
curl -k -H 'Content-Type: application/json' -X POST --data-binary @text.json https://
<ip_address>:<port>/analytics/models/v1/get_keywords/
```

The following code is an example JSON payload:

```
{"text”: “Just a short message to apprise everyone of the cleaning results performed on the
filter separator from the Transwestern M/S. Wipe tests taken of the interior portions of the
separator prior to cleaning revealed PCB results of: 360, 500 and 520 microgram. This was not a
standard wipe taken of a specific area. These wipes were taken over large areas to determine
presence/absence of PCB's within the separator. After the cleaning operation was completed,
three wipe samples were taken of the interior portions of the separator. All results showed non
detect at less than 1.0 microgram. I feel we can be fairly certain that the separator is free
of residual PCBs. Jeff, would you pass this information on to your counterpart for PG&E?"}
```

The following code is an example response:

```
{"keywords": ["cleaning revealed pcb results", "cleaning results performed", "results showed",
"pcb ", "cleaning operation", "wipe tests", "wipe samples", "standard wipe", "specific area",
"short message"]}
```

Accuracy and limitations

It is difficult to determine the accuracy of this keyword extraction method since most existing approaches focus on the manual assignment of keywords by professional curators who may use a fixed taxonomy. Alternatively, keywords are produced by the author of the text, and rely on the author's judgment to provide a representative list.

This is a heuristic method based on punctuation, stop-words, and frequency counts to determine key words and phrases. It does not use an NLP model to determine the meaning of the text.

The algorithm uses a list of English stop-words.

Although there is no actual limitation on the size of input text, the number of keywords per text is defined in the configuration as a global constant, for example, 10. Overall the algorithm will give more representative keywords for shorter texts.

Parsing emails

Many of the natural language processing steps require relatively clean texts to produce good results. However, email text can be messy, and may include irrelevant replies, forwards, spam, signatures, disclaimers, and other artifacts. Due to the wide range of email formats and mail protocols in use, email texts can also include a range of characters that are problematic for analysis, including HTML tags, unusual characters, and corrupted or encoded sequences of random characters. As a first step of most NLP pipelines, this service is used to remove redundant characters and text.

The e-communication email parsing process is used to clean up email texts before further processing. There are two components:

- a python mailparse library that is called directly by other python machine learning code
- a standalone mailparse REST API that can be used as an on-demand email cleaning service

Approach to solving the business problem

This service uses python email libraries and regular expressions to split emails into their component parts.

Assumptions

Email content is derived from the current email only; replies and forwarded emails (the email history trail) are not used to determine email content or context.

Regular expressions are good enough to identify most of e-mail sections. These are configurable and can be extended as necessary.

Input is a file of email texts in MIME or similar structure.

Capitalization and punctuation are not used later as part of the NLC model. In other words, there is no downstream NLP model that relies on accurate character level input to determine features.

Using the REST service

Starting the REST Service

```
python3 mailparseRESTAPI.py &
```

Sample input

```
Message-ID: <29665600.1075855687895.JavaMail.evans@thyme>\nDate: Tue, 26 Sep 2000 05:11:00
-0700 (PDT)\nFrom: phillip.allen@enron.com\nTo:
    cindy.cicchetti@enron.com\nSubject: Re: Gas Trading Vision meeting\nMime-Version:
1.0\nContent-Type: text/plain; charset=us-ascii\nContent-Transfer-Encoding:  7bit\nX-From:
Phillip K Allen\nX-To: Cindy Cicchetti\nX-cc: \nX-bcc: \nX-Folder: \
\Phillip_Allen_Dec2000\\Notes Folders\\'sent mail\nX-Origin: Allen-P\nX-FileName: pallen.nsf
\n\nNymex expiration is during this time frame.  Please reschedule.
```

Sample response

```
Nymex expiration is during this time frame.  Please reschedule.
```

parse_email Service details

The service allows users to parse a single email into its component parts.

Table 22: parse_email service details			
Method	**URL**	**Input**	**Output**
POST	/analytics/models/v1/parse_email	JSON payload	JSON response

The following is an example CURL command to POST:

```
curl -k -H 'Content-Type: application/json' -X POST --data @msg_29.json https://ip address:port/
analytics/models/v1/parse_email/
```

The following code is an example JSON payload:

```
{"message": "Message-ID: <29665600.1075855687895.JavaMail.evans@thyme>\nDate: Tue, 26 Sep 2000
05:11:00 -0700 (PDT)\nFrom: phillip.allen@enron.com\nTo:
    cindy.cicchetti@enron.com\nSubject: Re: Gas Trading Vision meeting\nMime-Version:
1.0\nContent-Type: text/plain; charset=us-ascii\nContent-Transfer-Encoding:
    7bit\nX-From: Phillip K Allen\nX-To: Cindy Cicchetti\nX-cc: \nX-bcc: \nX-Folder: \
    \Phillip_Allen_Dec2000\\Notes Folders\\'sent mail\nX-Origin: Allen-P\nX-FileName: pallen.nsf
\n\nNymex expiration is during this time frame.  Please reschedule."}
```

The following code is an example response:

```
{"history": null,"body": "Nymex expiration is during this time frame. Please reschedule.",
"header": {"from": "phillip.allen@enron.com", "date": "Tue, 26 Sep 2000 05:11:00 -0700 (PDT)",
"to": "cindy.cicchetti@enron.com", "subject": "Re: Gas Trading Vision meeting", "X-from":
"Phillip K Allen", "X-to": "Cindy Cicchetti
```

clean_email Service details

This function cleans the email body text by removing special characters so that only alpha-numeric characters remain.

Table 23: clean_email service details			
Method	**URL**	**Input**	**Output**
POST	/analytics/models/v1/clean_email	JSON payload	JSON response

The following is an example CURL command to POST:

```
curl -k -H 'Content-Type: application/json' -X POST --data @msg_cln_1.json https://ip
address:port/analytics/models/v1/clean_email/
```

The following code is an example JSON payload:

```
{"message":"Allen.xls Enclosed is the preliminary proforma for the Westgate property is Austin
that we told you about. ... number of things she does everyday. Fortunately, it looks as if she
will be ok in the long run. George W. Richards Creekside Builders"}
```

The following code is an example response:

```
{"message": "Allen xls Enclosed is the preliminary proforma for the Westgate property is Austin
that we told you about .. number of things she does everyday Fortunately it looks as if she
will be ok in the long run George W Richards Creekside Builders"}
```

Accuracy and limitations

The algorithm uses lists of python regular expression patterns to identify each of the key parts of an email: signatures, forwarded emails, disclaimers, etc. In some cases, these sections may be missed or misidentified. The regular expressions can be amended if new disclaimers or sign-offs are found.

Currently only emails in English are parsed.

Sampling emails

The e-communication sampling process is used to create small sample datasets of emails that represent the main types of themes that are found in the whole corpus of emails. These samples can be used by Subject Matter Experts (SMEs) as templates for identifying whether the emails really are complaints and what entities are contained in the emails. Subsequently, you can use these annotated emails to train the natural language classifier and natural language understanding models

Business problem

The sampling process is required to solve the restrictions around the number of emails that can be accurately labeled manually by SMEs. Since hand labeling is time-consuming and resource-intensive, not all complaints can be chosen. Instead, a sample is taken by using a theme detection algorithm combined with a stratified sampling method. Themes are discovered in the email corpus, and the sampling method replicates the proportion of each discovered theme in the sampled datasets. This should ensure that a wide and representative set of training examples is created.

The following steps are carried out:

1. All of the emails are cleaned and processed so that only the current email text is used.
2. The email class counts are adjusted to reach the desired balance of classes, for example, complaints=75%, non-complaints=25%.
3. The emails are grouped into consistent themes.
4. Small datasets are produced containing representative amounts of all of the themes that are discovered in the previous step.
5. The dataset IDs for each sampled email are sent by a REST AP to be included in the email meta-data.

You can also choose random sampling by changing the sampling method parameter. If you choose this option, the algorithm carries out the following steps:

1. All emails are cleaned and processed so that only the current email text is used.
2. The email class counts are adjusted to reach the desired balance of classes, for example, complaints=75%, non-complaints=25%.
3. Emails are chosen at random from the proportions of classes that are used in step 2.

If a random sample is taken, then it is possible that some categories of complaints or entities might not be included due to chance.

Approach to solving the business problem

For this problem, we assume that most complaints can be grouped into a number of categories, and that this number is in the range of 5 to 50. A Latent Dirichlet Algorithm (LDA) is used to determine which breakdown of topics would best divide the complaints into such groups. This is done by determining the set of keywords that best describe each complaint theme, and the number of themes that best describe all of the complaints.

Although the themes and the keywords of each theme are derived, they are not used directly by Surveillance Insight. They are, however, visible in the algorithm logs. Instead, the approach is taken of grouping the complaints by similar keywords, and then ensuring that all of the groups are included in the samples in the relative proportions that they occur.

The following parameters are used for this algorithm:

- Upper limit on email length: Occasionally very long emails can be included in the dataset. To prevent this, emails with lengths that are more than a number of standard deviations above the mean are excluded.
- Minimum number of words in an email: This parameter is used to exclude emails that are too short to be of use.
- Part of speech tags for lemmatization: This parameter controls which parts of speech are considered for analysis of themes. The default setting is 'NOUN', 'ADJ', 'VERB','ADV'. All other parts of speech are not considered.
- Latent Dirichlet allocation grid search parameter for number of topics: The algorithm must determine the best number of topics that represent all of the emails. A range of topics can be entered. However, a large set of numbers can substantially increase the processing time.
- Latent Dirichlet allocation grid search parameters for number for learning decay: This parameter is also used to determine the best number of topics that represent all of the emails. You can enter a range of learning decay coefficients. However, a large set of numbers can substantially increase processing time.

Assumptions

There is a large corpus of emails: in the range 100 thousand - 10 million.

Of all of the emails in this corpus, approximately only 2 - 3% are complaints, that is, in total there are between 2000 - 20,000 actual complaints.

Due to time and staffing constraints, SMEs are able to manually label only a few hundred emails at most, for example, 300.

Accuracy and limitations

The algorithm must determine the number of themes that are present in all of the emails by using a grid search algorithm. This is done using a configuration to suggest a range of different numbers, and the algorithm determines which works best. However, if the range of numbers is too great, then the processing time may increase substantially.

Natural language classifier

A long short term memory (LSTM) based deep learning model is used to classify text into two categories.

For training the model, annotated text data in CSV files is used. Each text is assigned a class by an SME. For cleaning the training data, the emailparse library is used.

Approach to solving the business problem

Surveillance Insight uses an LSTM-based deep learning network to classify the mails into two classes.

Assumptions

Surveillance Insight uses a word embedding method to train word embedding as per the training data.

Starting the REST server

```
python3 RestApi.py
```

Get all NLC models

The service allows to get all the model details at present.

Method
GET

URL
/nlc/v1/v1/models/

Input
none

Output
JSON response

The following is an example CURL command:

```
curl -X GET -k –I 'https://HOST:PORT/nlc/v1/models/'
```

The following code is an example response:

```
{"models": [{"creator": "creator name", "last trained": "NA", "dataset": "NA", "f1": "NA",
"description": "description of the model", "id": "1529472109688", "status": "draft", "labels":
null, "p": "NA", "name": "name of the model", "r": "NA"}]}
```

Create a model

This service allows to create a model with the given JSON payload.

Method
POST

URL
/nlc/v1/v1/models/

Input
JSON Payload

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip address:port/nlc/v1/models/' --data '{"description":"description
of the model","name":"name of the model","creator":"creator name"}'
```

The following code is an example JSON payload:

```
{"description":"description of the model", "name":"name of the model", "creator":"creator name"}
```

The following code is an example response:

```
{"id": "1529472385051"}
```

Train a model

This function trains a model with the appropriate dataset. The dataset must be uploaded by using the Design Studio. This function supports input and datasets from HDFS.

Method
POST

URL
/nlc/v1/v1/models/<model-id>/train/

Input
JSON Payload

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip address:port/nlc/v1/models/1529472385051/train/' --data
' {"dataset":"uploaded_dataset_name.csv"}
```

The following code is an example JSON payload:

```
{"dataset":"uploaded_dataset_name.csv"}
```

The following code is an example JSON payload for an HDFS dataset.

```
{"hdfs_path":"path/to/hdfs/dataset/file/uploaded_dataset_name.csv"}
```

The following code is an example response:

```
{
    "status": {
        "code": 200
    },
    "f1": 0.XX,
    "precision": 0.XX,
    "recall": 0.XX
}
```

Get details

This function allows you to get details of the already created models.

Method
GET

URL
/nlc/v1/v1/models/<model-id>/details/

Input
None

Output
JSON response

The following is an example CURL command:

```
curl -X GET -k -i 'https://ip address:port/nlc/v1/models/1529472385051/details/
```

The following code is an example response:

```
{"creator": "creator name", "last trained": "NA", "dataset": "NA", "f1": "NA", "description":
"description of the model", "id": "1529472385051", "status": "draft", "labels": null, "p":
"NA", "name": "name of the model", "r": "NA"}
```

Publish model

Use this function to publish a model after you train it.

Method
POST

URL
/nlc/v1/v1/models/<model-id>/publish/

Input
None

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip address:port/nlc/v1/models/1529472385051/publish/
```

The following code is an example response:

```
{"creator": "creator name", "last trained": "last trained time", "dataset": "dataset name",
"f1": "xx", "description": "description of the model", "id": "1529472385051", "status":
"published", "labels": [label names], "p": "xx", "name": "name of the model", "r": "xx"}
```

Delete model

This function cleans the email body text by removing special characters so that only alpha-numeric characters remain.

Method
DELETE

URL
/nlc/v1/v1/models/<model-id>/delete/

Input
None

Output
JSON response

The following is an example CURL command:

```
curl -X DELETE -k -i 'https://ip address:port//nlc/v1/models/1529472385051/delete/'
```

The following code is an example response:

```
{"id": "1529472385051"}
```

Sample of training tuple

```
"The recent approaches that we took has turned out to be an abject failure. It
was disgusting to see him smiling on my failure." ,class-x
```

Accuracy and limitations

The algorithm uses LSTM deep learning network to classify given text into one of the classes. In some cases, the text can be misclassified as there can be similar text in multiple classes.

Currently only text in English can be classified.

Natural language understanding

Natural language understanding models can tag the words of a sentence with given tags, and to tag two custom entities and then use the two tagged entities to extract a relationship between the entities in the text. This integrated service is implemented in `NLUIntegrated.py`.

Approach to solving the business problem

Surveillance Insight uses a long short term memory (LSTM) based deep learning network to tag the words into multiple tags according to the training data tags. The LSTM deep networks can learn and remember long sequences and can relate among the sequences.

Starting the REST service

```
python3 NLUIntegrated.py
```

The service provides the following APIs:

- /nlu/v1/models/' (GET)
- '/nlu/v1/models/' (POST)
- '/nlu/v1/models/<model_id>/train/' (POST)
- '/nlu/v1/models/<model_id>/details/' (GET)
- '/nlu/v1/models/<model_id>/delete/' (DELETE)
- '/nlu/v1/models/<model_id>/analyze/' (POST)

Get all NLU models

The service gets all of the model details.

Method
GET

URL
/nlu/v1/v1/models/

Input
none

Output
JSON response

The following is an example CURL command:

```
curl -X GET -k –I 'https://HOST:PORT/nlu/v1/models/'
```

The following code is an example response:

```
{"models": [{"creator": "creator name", "last trained": "NA", "dataset": "NA", "f1": "NA",
"description": "description of the model", "id": "1529472109688", "status": "draft", "labels":
null, "p": "NA", "name": "name of the model", "r": "NA"}]}
```

Create model

This service creates a model with a given JSON payload.

Method
POST

URL
/nlu/v1/v1/models/

Input
JSON payload

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip address:port/nlu/v1/models/' --data '{"description":"description
of the model","name":"name of the model","creator":"creator name"}'
```

The following code is an example payload:

```
{"description":"description of the model", "name":"name of the model", "creator":"creator name"}
```

The following code is an example response:

```
{"id": "1529472385051"}
```

Train a model

This function trains a model with an appropriate dataset. The model must already exist, and the dataset must be uploaded by using the Surveillance Insight Design Studio.

Method
POST

URL
/nlu/v1/v1/models/<model-id>/train/

Input
JSON payload

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip address:port/nlu/v1/models/1529472385051/train/' --data
' {"dataset":"uploaded_dataset_name.csv"}'
```

The following code is an example payload:

```
{"dataset":"uploaded_dataset_name.csv"}
```

The following code is an example response:

```
{
    "status": {
        "code": 200
    },
    "f1": 0.XX,
    "precision": 0.XX,
    "recall": 0.XX
}
```

Get details

Use this function to get details of an already created model.

Method
GET

URL
/nlu/v1/v1/models/<model-id>/details/

Input
None

Output
JSON response

The following is an example CURL command:

```
curl -X GET -k -i 'https://ip address:port/nlu/v1/models/1529472385051/details/
```

The following code is an example response:

```
{"creator": "creator name", "last trained": "NA", "dataset": "NA", "f1": "NA", "description":
"description of the model", "id": "1529472385051", "status": "draft", "labels": null, "p":
"NA", "name": "name of the model", "r": "NA"}
```

Publish model

This function publishes a model after it is trained.

Method
POST

URL
/nlu/v1/v1/models/<model-id>/publish/

Input
None

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip address:port/nlu/v1/models/1529472385051/publish/
```

The following code is an example response:

```
{"creator": "creator name", "last trained": "last trained time", "dataset": "dataset name",
"f1": "xx", "description": "description of the model", "id": "1529472385051", "status":
"published", "labels": [label names], "p": "xx", "name": "name of the model", "r": "xx"}
```

Delete model

This function cleans the email body text by removing special characters so that only alpha-numeric characters remain.

Method
DELETE

URL
/nlu/v1/v1/models/<model-id>/delete/

Input
None

Output
JSON response

The following is an example CURL command:

```
curl -X DELETE -k -i 'https://ip address:port//nlu/v1/models/1529472385051/delete/'
```

The following code is an example response:

```
{"id": "1529472385051"}
```

Sample of training tuples

```
the brief visit , mr. de villepin 's first foreign trip since becoming prime
minister in may , had another purpose : to repair the always edgy relationship
between britain and france that became even edgier at the
<entity1>european_union<entity1> summit meeting in <entity2>brussels<entity2>
last month . /location/location/contains(european_union,brussels)
```

Accuracy and limitations

The algorithm uses LSTM deep learning network to tag the entities and it uses GRU RNN to extract the relationship between the chosen entities. In some cases, the words can be tagged with wrong labels and the detected entities can be wrong which may lead to a wrong relation as there can be similar text in multiple places and they might have different entities tagged for relations.

Currently only text in English can be tagged and relation extraction is also supported for English only.

Discovery services

The discovery services provide two REST APIs.

Analyze service

This service takes a complaint as input and returns the JSON for all of the contexts, action verbs, emotions, and objects.

Method
POST

URL
/discovery/v1/analyse

Input
Text

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip_address:port/discovery/v1/analyse' --data 'Hello how are you
doing today?'
```

The following code is an example payload:

```
Hello how are you doing today?
```

The following code is an example response:

```
{"entities": [{"context": "you", "actionverb": "are doing", "emotion": "na", "object": "na",
"is_negative": "False"}], "status": {"message": "Success", "code": 200}}
```

Entities service

This service takes input from HDFS and from Db2 if you have tagged some data in the previous iteration. It performs some data processing and returns the most frequently occurring words for contexts, action verbs, emotions, and objects along with the tags.

Method
POST

URL
discovery/v1/entities

Input
JSON

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip_address:port/discovery/v1/entities' --data '{
"path":"/user/sifsuser/comm_raw/",
"model_id":1,
"iteration_id":41,
"is_negative": "true"
}'
```

The following code is an example of a JSON payload for a negative action verb:

```
{
"path":"/path/to/data /",
"model_id":1,
"iteration_id":41,
"is_negative": "true"
}
```

The following code is an example of a JSON payload for a positive action verb:

```
{
"path":"/path/to/data /",
"model_id":1,
"iteration_id":41,
"is_negative": "false"
}
```

The following code is an example of a JSON payload for all action verb:

```
{
"path":"/path/to/data /",
"model_id":1,
"iteration_id":41
}
```

Entity extraction services

The entity Extraction service has one REST API, which returns a list of entities (organization and person) in a given email body.

Analyze text service

This service takes a complaint as input and returns the JSON for all the contexts, action verbs, emotions, and objects.

Method
POST

URL
/analytics/models/v1/analyzetext/

Input
Text

Output
JSON response

The following is an example CURL command:

```
curl -X POST -k -i 'https://ip_address:port/analytics/models/v1/analyzetext/' --data 'I work at Company X'
```

The following code is an example payload:

```
I work at Company X
```

The following code is an example response:

```
{"entities": [{"name": "Company X", "type": "organization"}], "status": {"message": "Success", "code": 200}}
```

Emotion detection library

The Emotion Detection library uses Apache UIMA Ruta (RUle based Text Annotation) and a custom scoring model to detect emotions and sentiment in unstructured data, such as text from emails, instant messages, and voice transcripts.

The library detects the following emotions from the text:

- Anger
- Disgust
- Joy
- Sadness
- Fear

It assigns a score from 0-1 for each emotion. A higher value indicates a higher level of the emotion in the content. For example, an Anger score of 0.8 indicates that the anger is likely to be present in the text. A score of 0.5 or less indicates that anger is less likely to be present.

The library also detects the sentiment and indicates it as positive, negative, or neutral with a score of 0-1. For example, a positive sentiment score is 0.8 indicates that positive sentiment is likely expressed in the text. A score of 0.5 or less indicates that positive sentiment is less likely expressed in the text. The sentiment score is derived from the emotions present in the text.

How the library works

The solution uses dictionaries of emotions and rules to detect the emotions in text and a scoring model to score the emotions.

The dictionaries are contained in the `anger_dict.txt`, `disgust_dict.txt`, `fear_dict.txt`, `joy_dict.txt`, and `sad_dict.txt` files. Each dictionary is a collection of words that represent emotion in the text.

The rule file is based on Ruta Framework and it helps the system to annotate the text based on the dictionary lookup. For example, it annotates all the text that is found in the anger dictionary as Anger Terms. The position of this term is also captured. All the inputs are fed into the Scoring model to detect the sentence level emotions and also the document level emotion. The document level emotion is returned as the overall emotion at the document level.

The following code is an example of a rule definition.

```
PACKAGE com.ibm.sifs.analytics.emotion.types;

" Sample Rule
" load dictionary
WORDLIST anger_dict = 'anger_dict.txt';
WORDLIST joy_dict = 'joy_dict.txt';

" Declare type definitions
DECLARE Anger;
DECLARE Joy;

" Detect sentence
DECLARE Sentence;
PERIOD #{-> MARK(Sentence)} PERIOD;

MARKFAST(Anger, anger_dict, true);
MARKFAST(Joy, joy_dict, true);

" Same for other emotions
```

The emotion detection dictionary

Emotion detection is a java-based library and is available as a JAR. Currently, it is used in the Real-time Analytics component to detect the emotions in real time and score the emotions in the incoming documents.

As shown in the following diagram, it offers two functions:

- Initialize, which initializes the Emotion library by loading the dictionary and the rules. This function needs to be called only once, and must be started when dictionaries or rules are changed.
- Detect Emotion, which takes text as input and returns a JSON string as a response.

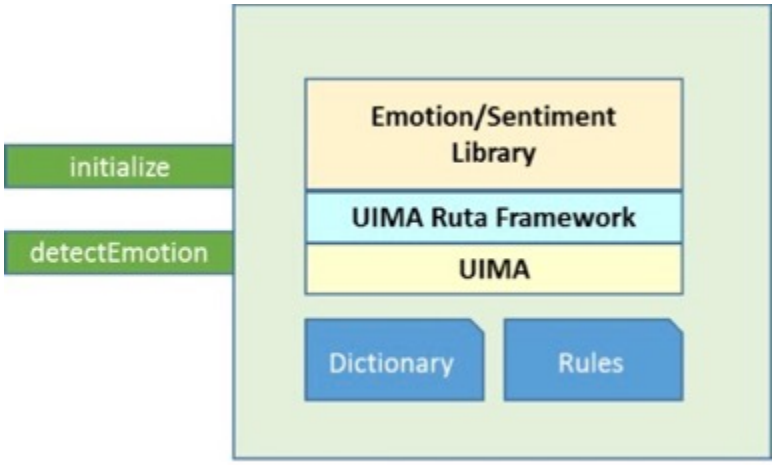


Figure 33: Emotion detection library

Definitions

```
public static void initialize(String dictionaryPath, String rulePath) throws Exception
```

```
public static String detectEmotion(String text)
```

Sample input

```
The investment you made with ABC company stocks are doing pretty good. It has increased
50 times. I wanted to check with you to see if we can revisit your investment portfolios for
better
investments and make more profit. Please do check the following options and let me know. I can
visit you at your office or home or at your preferred place and we can discuss on new business.
Market is doing pretty good. If you can make right investments now, it can give good returns on
your retirement.
```

Sample response

```
{
    "sentiment": {
        "score": 1,
        "type": "positive"
    },
    "emotions": {
        "joy": "1.0",
        "sad": "0.0",
        "disgust": "0.0",
        "anger": "0.0",
        "fear": "0.0"
    },
    "keywords": {
        "negative": [],
        "joy": ["pretty", "retirement", "good", "profit"],
        "sad": ["retirement"],
        "disgust": [],
        "anger": [],
        "fear": ["retirement"]
    },
    "status": {
        "code": "200",
        "message": "success"
    }
}
```

Starting the module

```
// Initialize Module (ONLY ONCE)
EmotionAnalyzer.initialize(<path to dictionaries>, <path to rule file>);

// Invoke the library (For every incoming document)
String resultJSON = EmotionAnalyzer.detectEmotion(text);
```

Note: Errors or exceptions are returned in the JSON response under the Status element with a code of 500 and an appropriate message. Success messages return as shown in the following example.

```
"status": {
        "code": "200",
        "message": "success"
    }
```

Concept mapper library

The Concept Mapper library uses Apache UIMA Ruta (RUle based Text Annotation) to detect the concepts in unstructured text such as emails, instant messages, or voice transcripts.

The library detects the following concepts from the text.

- Tickers—Stock Symbol
- Recruit Victims—Evidence of a trader who is trying to get clients to invest in a specific ticker. This activity is indicated as "Recruit Victims."

- Recruit Conspirators—Evidence of a trader who is collaborating with other traders to conduct a market abuse activity such as "pump/dump." This activity is indicated as "Recruit Conspirators" in the surveillance context.

Note: If there is more than one ticker in the text, all the tickers are extracted and returned as a comma-separated string.

How the library works

The solution uses a dictionary of tickers, keywords, or phrases that represent Recruit Victims and Recruit Conspirators, and concepts and rules to detect the concepts in the text. The dictionaries include the `dict.txt` for each concept, such as `recruitvictims`, `recruitconspirators`, and so on. Each dictionary is a collection of words that represent different concepts in the text.

The rule file is based on Ruta Framework and it helps the system to annotate the text based on the dictionary look up. For example, it annotates all of the text that is found in the Recruit Victims dictionary as Recruit Victims Terms. The position of this term is also captured.

The following code is an example of a rule.

```
PACKAGE com.ibm.sifs.analytics.types;

WORDLIST dict = 'dict.txt';

DECLARE Concept;

MARKFAST(Concept, dict, true);
```

The Concept Mapper dictionary

Concept Mapper is a java-based library and it is available as a JAR. It is used in the Real-time analytics component to detect the concepts in real time from the incoming text. As shown in the following diagram, it offers the following functions:

- Initialize, which initializes the library by loading the dictionary and the rules. This function needs to be called only once, and must be started when dictionaries or rules are changed.
- Detect concepts, which take text as input and returns a JSON string as a response.

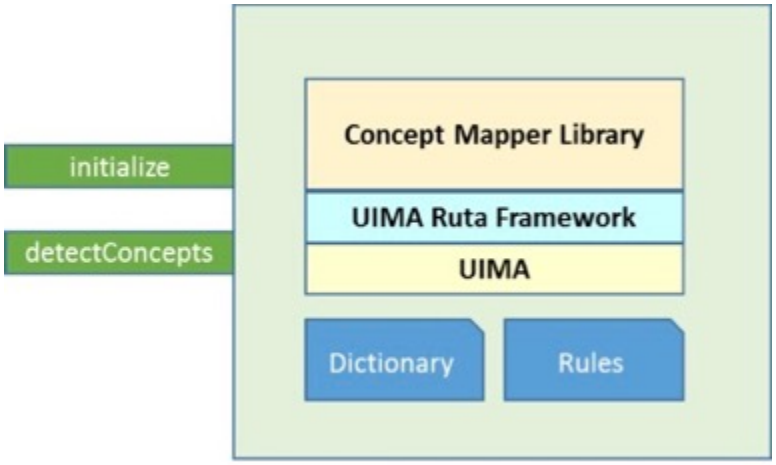


Figure 34: Concept mapper library

Definitions

```
public static void initialize(String dictionaryPath, String rulePath) throws Exception
```

```
public static String detectConcepts(String text)
```

Sample input

```
I wanted to inform you about an opportunity brought to us by an insider, Mr. Anderson,
from ABC Corporation. They specialize in manufacturing drill bits for deep-sea oil rigs. Mr.
Anderson
owns about 35% of the float and would like us to help increase the price of his company’s stock
price.
If we can help increase the price of the stock by 150%, we would be eligible for a substantial
fee and
also 1.5% of the profit Mr. Anderson will make disposing the shares at the elevated price.
Would you be
interested in joining our group in helping Mr. Anderson?
```

Detecting the concept

```
// Invoke the library (For every incoming document) String resultJSON =
ConceptMapper.detectConcepts(<path of dictionaries>, <path to rule file>,text);
```

Note: Errors or exceptions are returned in the JSON response under the Status element with a code of 500 and an appropriate message. Success messages return as shown in the following example.

```
"status": {
        "code": "200",
        "message": "success"
    }
```

Chapter 8. Dashboards

IBM Surveillance Insight for Financial Services solution provides two web-based dashboards that provide pictorial views of the analysis from processing structured and un-structured data.

Both of the applications can be accessed by typing the following URL into your web browser and providing appropriate log-in credentials:

https://<web-server-host>:<port>/

After logging in, card views are displayed where you can choose the application of interest to you.

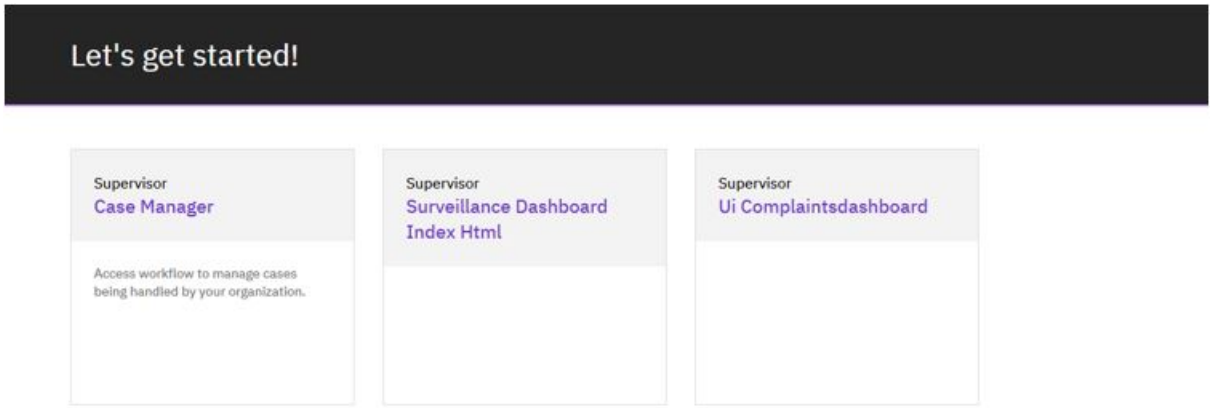


Figure 35: Card views

The Surveillance dashboard application is accessible to users with one of the following roles; supervisor, investigator, or analyst. The Complaints dashboard application is accessible only to users with the role of supervisor.

Surveillance dashboard

The Surveillance dashboard is a web-based application that shows the complaints trend analysis results.

Health Check dashboards

The Health Check dashboards give you the ability to view the health of different analytical processes.

You can access the view by clicking the **Health Check** tab on the **Surveillance** dashboard. There are two types of dashboards: Ecomm and Voice. By default, the **Ecomm – Ingestion** dashboard is displayed when you open the **Health Check** tab.

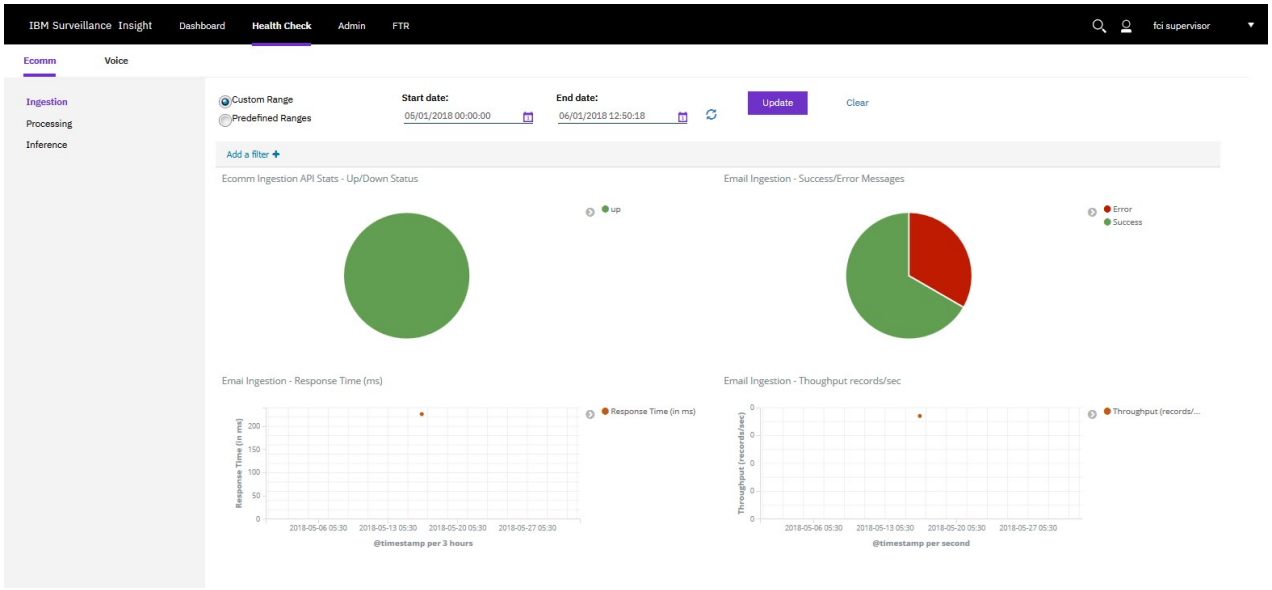


Figure 36: Default Health Check user interface

Both display data based on the selected date ranges. There are two types of date ranges that you can select; **Custom Range** and **Predefined Ranges**.

Custom Range

The **Custom Range** option contains a **Start date** and an **End date**. By default, the **Start date** is the current day and the **End date** is one year after the **Start date**. You can search for any **Start date** or **End date**.

Predefined Ranges

The **Predefined Ranges** option is used to select dates quickly. From the drop-down lists for **Start date** and **End date**, you can select from a number of options: Today, Yesterday, This Week, This Month, and This Year.

After you select a date range, click **Upload** to display data for the date range in the dashboards. By default, date ranges start from today.

Ecomm dashboards

There are three Ecomm dashboards: Ingestion, Processing, and Inference.

Ingestion

The **Ingestion** dashboard has four visualizations:

- **Ecomm Ingestion API Stats – Up/Down Status**. This chart displays the **Up/Down status** ratio for the Ecomm ingestion service during a specified time interval.
- **Ingestion API Stats – Success/Error Messages**. This chart displays the **Success/Error Messages** ratio for a specified time interval.
- **Ingestion API Stats – Response Time (ms)**. This chart shows the average response time of ingested records per millisecond for a specified time interval.
- **Ingestion API Stats – Thoughput records /sec**. This chart shows the average number of records that are ingested per second.

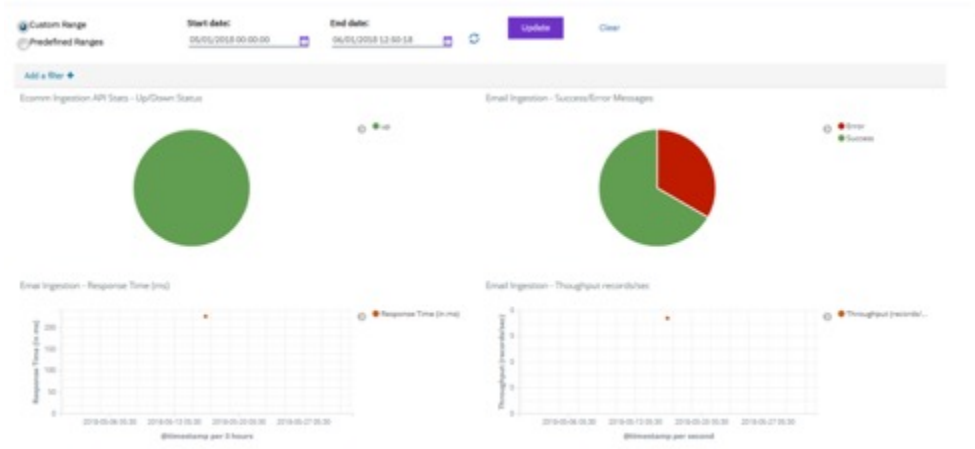


Figure 37: Ecomm Ingestion dashboard

Processing

The **Processing** dashboard has eight visualizations.

- **Ecomm Spark Job – Persist Email**. This chart shows the **Up/Down status** ratio about the job which persists email data ingested during a specified time interval. This job runs though out the day.
- **Ecomm Spark Job – Persist Chat**. This chart shows the **Up/Down status** ratio about the job which persists chat data ingested during a specified time interval. It runs for 2 to 5 minutes a day and has a higher percentage of down status.
- **Ecomm Spark Job – Persist Comm**. This chart shows the **Up/Down status** ratio about the job which persists communications data during a specified time interval. This job runs though out the day.
- **Ecomm Processing – No of Communication Processed**. This metric is the total number of emails that are processed by a **Persist Email** spark job for a specified time interval.
- **Ecomm Dashboard – Ingestion vs Processing**. This chart shows the real-time count of ingestion throughput and processing throughput, and the backlog (Ingestion – Processing) counts for a specified time interval. In the chart:
  - Green represents ecomm ingestion.
  - Blue represents ecomm processing.
  - Orange represents the backlog.
- **Ecomm Processing – GCID Errors**. This metric is the number of GCID errors for a specified time interval.
- **Ecomm Processing – parse email data errors**. This metric is the number of parse email errors for a specified time interval.
- **Ecomm Processing GCID Source Error Message**. This data table contains source Spark log messages of all GCID errors that are captured.
- **Ecomm Processing Parse Email Data Errors Source Message**. This data table contains source Spark log messages of all parse email data errors captured.

Figure 38: Ecomm Processing dashboard

Inference

The **Inference** dashboard has three visualizations.

- **Ecomm Processing – Inference Alerts bar chart**. This chart shows the number of inference alerts for a specified time interval.
- **Ecomm Processing – No of Communications Processed in Inference**. This metric is the sum of all communications that were processed in inference for a specified time interval.
- **Ecomm Processing – Inference Last Run Time**. This metric is the last run time of an inference spark job. The inference spark job generally runs once nightly in a 24-hour period.

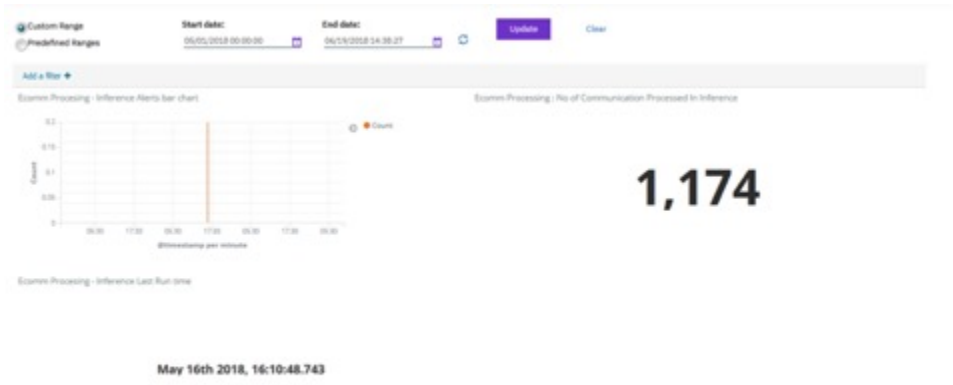


Figure 39: Ecomm Inference dashboard

Voice dashboards

The Voice health check dashboard has a single Ingestion Services view.

Ingestion Services

This view presents health data for voice ingestion services. The **Ingestion Services** dashboard has four visualizations:

- **Voice Ingestion API Stats – Uptime Status**. This chart displays the **Up/Down status** ratio for the Voice ingestion service during a specified time interval.
- **Voice Ingestion API Stats – Success/Error Messages**. This chart displays the **Success/Error Messages** ratio for a specified time interval.

- **Voice Ingestion API Stats – Response Time(in ms)**. This chart shows the average response time of ingested records per millisecond for a specified time interval.
- **Voice Ingestion API Stats – Thoughput Records /sec**. This chart shows the average number of records that are ingested per second.

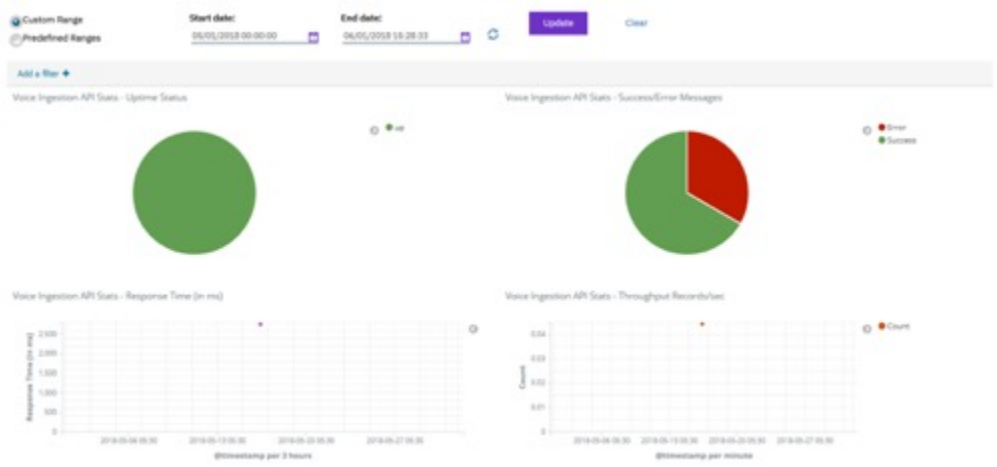


Figure 40: Voice Ingestion Services dashboard

Admin

The Surveillance dashboard provides a way to submit a Spark job using the Admin page.

You can access the page by clicking the **Admin** tab in the Surveillance dashboard. This page is primarily used for unsupervised learning during the discovery process.

Complaints dashboard

The Complaints dashboard is a web-based application that shows the complaints trend analysis results.

The Complaints dashboard shows the key trends in the analyzed complaints in a table. It also shows a bubble chart with further details on the trends. Each bubble corresponds to a single trend in the trend table. The size of the bubble corresponds to the volume of complaints that contributed to the trend. The color of the bubble corresponds to the level of risk that is associated with the complaint.

Figure 41: Complaints dashboard

Click a row in the trend table or a bubble in the bubble chart to display the **Theme Details** page.

Theme Details

The **Theme Details** page shows the complaints that contributed to the trend. It also shows a trend analysis chart with the complaint count and the date on the Y and X axis. Click one of the complaints in the table to show the evidence details page.

Figure 42: Theme Details page

Evidence Details

The **Evidence Details** page shows the actual complaint text with the customer information. The complaint text is annotated with entities of interest that were identified by the NLU model. Users can also add or remove tags that correspond to the emotions that are expressed in the complaint text.

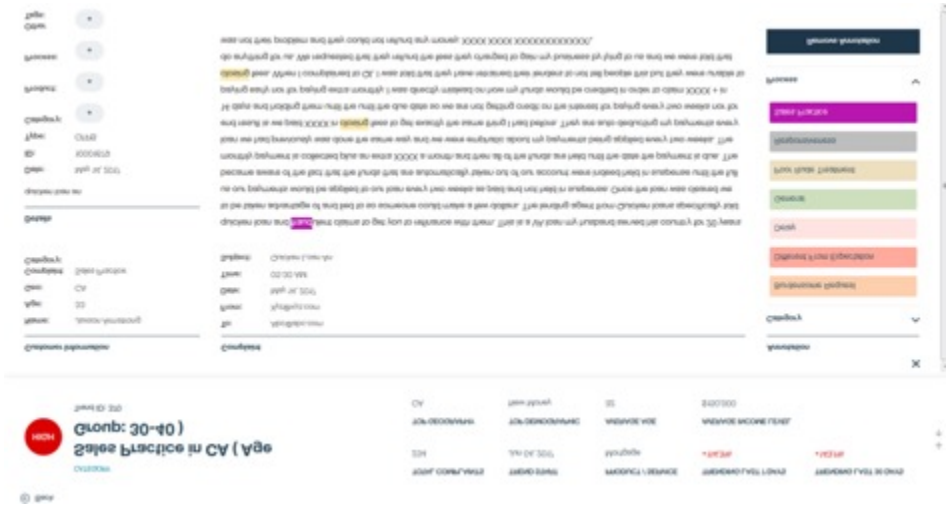


Figure 43: Evidence Details page

The complaints home page also contains the **Explore** option that you can use to view the complaint counts across the categories, products, processes, geographical areas, and age groups. It also has the keyword word counts for different combinations of the explore parameters. The following diagram shows the explore view.

Figure 44: Explore view

The side pane displays the various filter parameters that you can use to filter the insights. The complaints part of this view shows the list of actual complaints.

Figure 45: Explore view

Click a complaint from this list to display the complaint details similar to the one show in the **Evidence Details** page.

Chapter 9. GDPR compliance

Under the General Data Protection Regulation (GDPR) security requirements, an important requirement is to have a facility to remove the personal information (PI) related to a user from the solution's data store. Surveillance Insight provides this functionality by using a REST API service. The REST service removes the PI data of a user from IBM Db2 and from the Solr data store.

Removing party personal information

To remove party personal information, you run the right to forget service from the command line. On Linux operating systems, you can use the following curl command:

```
curl -k -H 'auth:digest' -X PUT --user ibmrest1:ibmrest@pwd1 --digest -v https://<host>:<port>/
righttoforget/v1/removeparty/<employeeid>
```

Ensure that you replace the `host`, `port`, and `employeeid` values with values that are appropriate for your environment.

`--user` *`ibmrest1:ibmrest@pwd1`* is a sample. Ensure that you use the correct user credentials.

On Microsoft Windows operating systems, you can use a tool such as Postman to call the service.

Data that is not considered by the service

In cases when the removed user is a supervisor, a reference might appear in a REPORTING_TO field for other users. The service does not modify this reference. To modify the reference, you must do so manually with the help of a DBA.

Surveillance Insight does not remove communication evidences that are related to the user.

Surveillance Insight does not remove any previous alerts or related entities that are generated for the user.

Surveillance Insight does not remove the risk models that are created by the user.

Chapter 10. Verifying audit records

IBM Surveillance Insight for Financial Services captures each request and response for auditing purposes. Admin users can view and verify audit details by using a REST API service.

URL

```
https://<audit_service_host_ip_address>:9233/security/audit/search?start_idx=0&count=5000
```

You can modify the `count` value to get more records.

Method

POST

Headers

```
Content-type: application/json
x-access-token: <token>
```

The `x-access-token` must be for a user with an admin role.

Admin users can get an `x-access-token` by going to the following product URL:

```
https://kubernetes.master:3000/security-auth/api/v1.0/login/ldap?
username=fciadmin&password=password
```

The *kubernetes.master* value is the fully qualified domain name or IP address for the Kubernetes master node computer. The *username* and *password* are the admin user's LDAP credentials.

Request body

```
{
"field":"productId",
"operator":"equal",
"value":"3"
}
```

The `productID` value for Surveillance Insight is 3.

Notices

This information was developed for products and services offered worldwide.

This material may be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. This document may describe products, services, or features that are not included in the Program or license entitlement that you have purchased.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Software Group
Attention: Licensing
3755 Riverside Dr.
Ottawa, ON
K1V 1B7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

IBM Surveillance Insight for Financial Services includes Brat (v 1.3) from the following source and licensed under the following agreement:

- http://weaver.nlplab.org/~brat/releases/brat-v1.3_Crunchy_Frog.tar.gz
- https://creativecommons.org/licenses/by-sa/3.0/legalcode

IBM Surveillance Insight for Financial Services includes spaCy Models (v 2.0.0) from the following source and licensed under the following agreement:

- https://spacy.io/models/en (en_core_web_sm 2.0.0)
- https://creativecommons.org/licenses/by-sa/3.0/legalcode

Trademarks

IBM, the IBM logo and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at " Copyright and trademark information " at www.ibm.com/legal/copytrade.shtml.

Index

V