Resilient Machine Learning User Guide

Installation

Complete the following steps to install the Python components:

1. Ensure that the environment is up-to-date, as follows:

sudo pip install --upgrade pip

sudo pip install --upgrade setuptools

sudo pip install --upgrade resilient-circuits

1. Run the following command to install pandas:

sudo pip install pandas-0.23.4-cp27-cp27m-linux\_x86\_64.whl

This step is necessary because there is no wheel for pandas on Redhat Enterprise 7 available from pip. Also Redhat Enterprise 7 does not have gcc/g++ installed, so pip cannot build it from source code. A wheel was built locally, using the source code of <https://files.pythonhosted.org/packages/e9/ad/5e92ba493eff96055a23b0a1323a9a803af71ec859ae3243ced86fcbd0a4/pandas-0.23.4.tar.gz>. This is the same source code as the “pip install pandas” downloads.

1. Run the following command to install the package:

sudo pip install --upgrade fn\_machine\_learning-<*version*>.tar.gz

Configure the Python components

The Resilient Circuits components run as an unprivileged user, typically named integration. If you do not already have an integration user configured on your appliance, create it now.

Complete the following steps to configure and run the integration:

1. Using sudo, switch to the integration user, as follows:

sudo su - integration

1. Use one of the following commands to create or update the resilient-circuits configuration file. Use –c for new environments or –u for existing environments.

resilient-circuits config -c

or

resilient-circuits config -u

1. Edit the resilient-circuits configuration file, as follows:
   1. In the [resilient] section, ensure that you provide all the information required to connect to the Resilient platform.
   2. In the [machine\_learning\_predict] section, edit the settings as follows:

model\_dir=path to the folder you are going to save your model files

* 1. In the [machine\_learning] section, edit the settings as follows:

prediction=API\_name\_of\_field\_to\_predict

features=comma-separated incident fields used to formulate algorithm

﻿# Algorithms supported:

# Logistic Regression, Decision Tree, Random Forest, Dummy Classifier

# SVM, SVM with Gaussian kernel, GaussianNB, BernoulliNB,

# K-Nearest Neighbors

algorithm=algorithm\_for\_model

﻿# Ensemble method is optional, it can be Bagging or Adaptive Boosting

method=optional\_ensemble\_method

split=split\_train\_test\_default\_0.5

# Optional limit for number of samples to be used to train a model

max\_count=optional\_limit\_on\_max\_number\_of\_samples\_to\_process

﻿#

# Advanced options

#-----------------

#

# 1. Imbalanced Class

#

# Predicted data could be imbalanced. Uncomment one of the following option

# to handle it

#

#class\_weight=balanced

#imbalance\_upsampling=true

#

# 2. Data Preparation

#

# Some prediction values could be misleading for the machine learning model

# Put those values below. Samples with those values will be filtered out

#

unwanted\_values=None

Please note that these are the algorithms and ensemble methods supported currently.

Algorithms:

* [Logistic Regression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
* [SVM](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)
* [SVM with Gaussian kernel](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)
* [Decision Tree](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)
* [Random Forest](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)
* [GaussianNB](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html)
* [BernoulliNB](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html)
* [K-Nearest Neighbor](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

Ensemble methods:

* [Bagging](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html)
* [Adaptive Boosting](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)

Deploy Customizations to the Resilient platform

This package contains one function definition, and includes one example workflow and a rule that runs this function.

1. Use the following command to deploy these customizations to the Resilient platform:

resilient-circuits customize

1. Respond to the prompts to deploy the function, message destination, workflow, script, and rule.

Run the Integration

To test the integration package before running it in a production environment, you must run the integration manually with the following command:

resilient-circuits run

The resilient-circuits command starts, loads its components, and continues to run until interrupted. If it stops immediately with an error message, check your configuration values and retry.

﻿Build a Machine Learning Model

This package includes a command line tool to build a machine model. It reads the settings from the [machine\_learning] section of the app.config to build the model. This command line tool has five subcommands.

## Download

This subcommand downloads incidents from a Resilient platform and store the data as in CSV file.

|  |  |
| --- | --- |
| -o | Required.  File path to the file to save the incident in CSV file. |

Example:

res-ml download -o incident\_samples.csv

## count\_value

This subcommand downloads incidents from a Resilient platform and store the data as in CSV file.

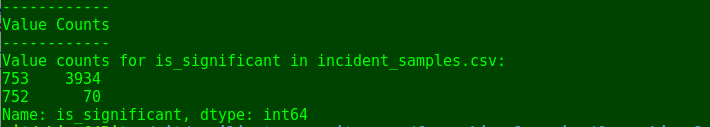
|  |  |
| --- | --- |
| -i | Required.  File path to the file to the saved the incident in CSV file. |
| -f | Required.  Field to count value |

Example:

res-ml count\_value -i incident\_samples.csv -f is\_significant

In this example, we want to check the value count of is\_significant, which is a field of the incident we want to predict.

Sample output:



This command is used to check whether the dataset is imbalanced regarding the field you want to predict. In this example, we can tell from the value count that the dataset is imbalanced. Note that there are approaches one can use to compensate an imbalanced dataset like this.

## Build

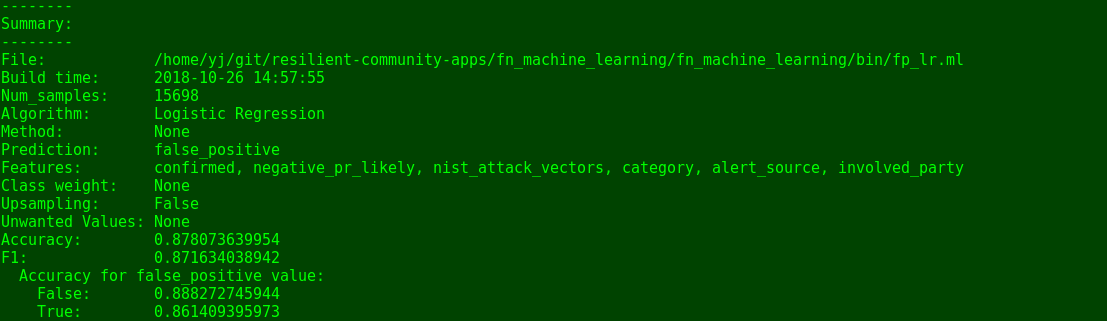
This subcommand is used to build a new model. It takes two flags:

|  |  |
| --- | --- |
| -o | Required.  File path to the file to save the built model. |
| -c | Optional.  File path to a CSV file that contains the samples.  If this flag is absent, the tool downloads incidents from the Resilient platform (specified in app.config) as samples.  If this flag is given, the CSV file is used for samples instead. |

Example:

res-ml build -o fp\_lr.ml

If the model can be built successfully, an output like this is shown:



In this example, we try to predict a field called false\_positive. The output shows what predict field, features, algorithm, optional method. It also shows what if any compensation we selected for the imbalanced class. Please refer to the next section about imbalance class compensation.

As for the measurements, it shows the overall accuracy, and the F1 value. In addition, it shows the accuracy for each value of the predicted field.

## Rebuild

This subcommand is used to rebuild a saved model. It takes two flags:

|  |  |
| --- | --- |
| -i | Required.  File path to the saved model to be rebuilt. |
| -c | Optional.  File path to a CSV file that contains the samples.  If this flag is absent, this tool downloads incidents from the Resilient platform (specified in app.config) as samples.  If this flag is given, the CSV file is used for samples instead. |

Please note that when a model is rebuilt, the predict/features/algorithm/method information is taken from the saved file instead of from app.config. This subcommand is intended for rebuilding/updating a successful model after new samples are available.

Example:

res-ml build -i lg\_adaboost.ml

If the model can be rebuilt successfully, an output like this is shown:



## View

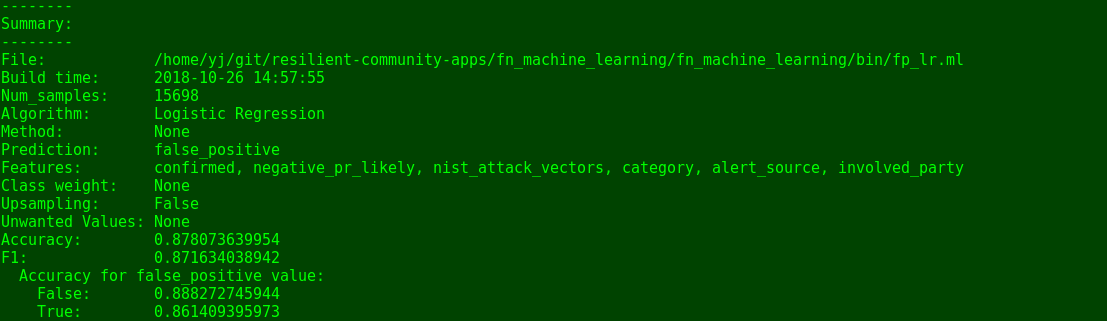
This subcommand is used to view a saved model. It takes one flag:

|  |  |
| --- | --- |
| -i | Required.  File path to the saved model to be rebuilt. |

Example:

res-ml view -i fp\_lr.ml

It shows the summary of the saved model. This is the same as the output you can see when you build a model. The information is saved into the model file, so you can view it later.



## Logging

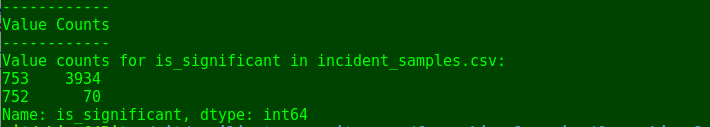
The res-ml command line utility creates a log file called res-ml.log in the current folder you run res-ml. It also generates a CSV file called resilient\_incidents.csv to store all the samples. To enable debug level logging, put a “-v” flag in front of the subcommand.

Example

res-ml -v view -i fp\_lr.ml

Handling imbalanced dataset

An imbalanced dataset has disproportionate ratio of observations in each class.



As shown in the value count output of the above example, there are in total 4004 samples (incidents). Only 70 (1.75%) of them have is\_significant=752 (True). This is normally referred as minority class. 98.25% of the samples have is\_significant=753 (False). This is the majority class.

When a dataset is imbalanced, a machine learning algorithm normally pays most of its attention to fitting the majority class. As a result, the accuracy for the minority class is not so satisfactory.

If the customer cares more about the accuracy of the majority class (is\_significant=False here), then there is no need to compensate the imbalanced dataset.

But if the customer cares about the accuracy of the minority class (is\_significant=True), then compensation approaches can be selected from the app.config file. Right now there are two compensation approaches we support:

* Upsampling. Uncomment the following line in app.config to enable this.

﻿﻿imbalance\_upsampling=true

Note this approach add copies of the minority samples into the dataset until its count is the same as the majority samples. In the above example, about 56 copies of each minority sample will be added. This will force the machine learning algorithm to pay more attention to fitting the minority samples.

* Balanced class weight. Uncomment the following line in app.config to enable this.

class\_weight=balanced

This approach sets the class\_weight flag of an machine learning algorithm to “class\_weight”. This essentially sets a higher weight factor to the minority samples. It can effectively force the machine learning algorithm to pay more attention to fitting the minority samples.

Filtering unwanted samples

Sometimes there are incidents you doing want to use to train a machine model. For example, if you want to predict is\_significant, and some incidents might have blank value for this field. These incidents in general can confuse the machine learning model, and it is a good practice to remove them.

A blank value is read as None. So in app.config, put a line like this to ignore those incidents with is\_significant as blank.

unwanted\_values=None

Value “Unknown” normally can confuse the model as well. It is a good practice to filter them as well.

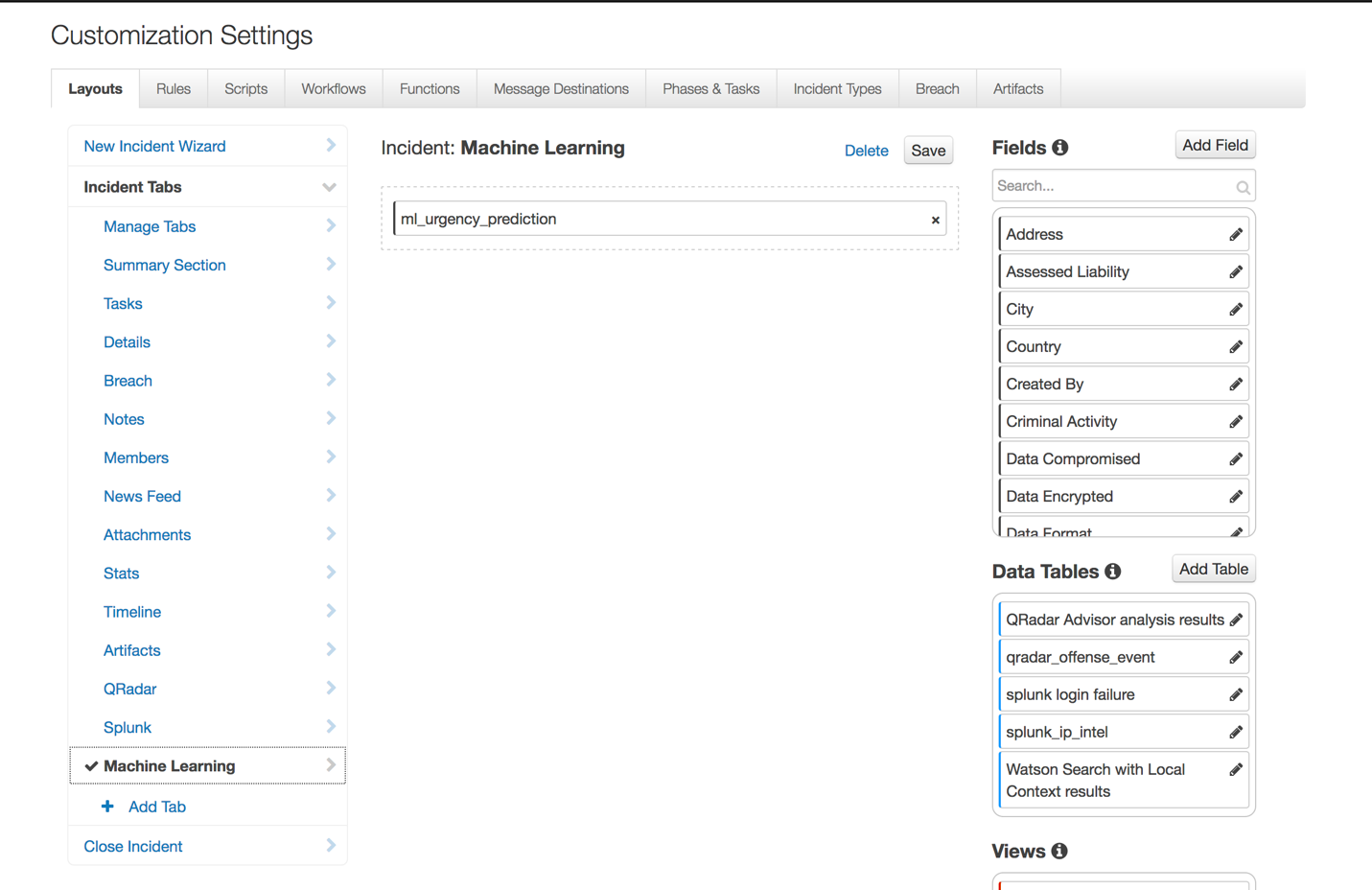
Use a ML Model to Predict

Copy the model you want to use for prediction to the model\_dir specified in the app.config. Then you can refer to it from your pre-process script in your function.

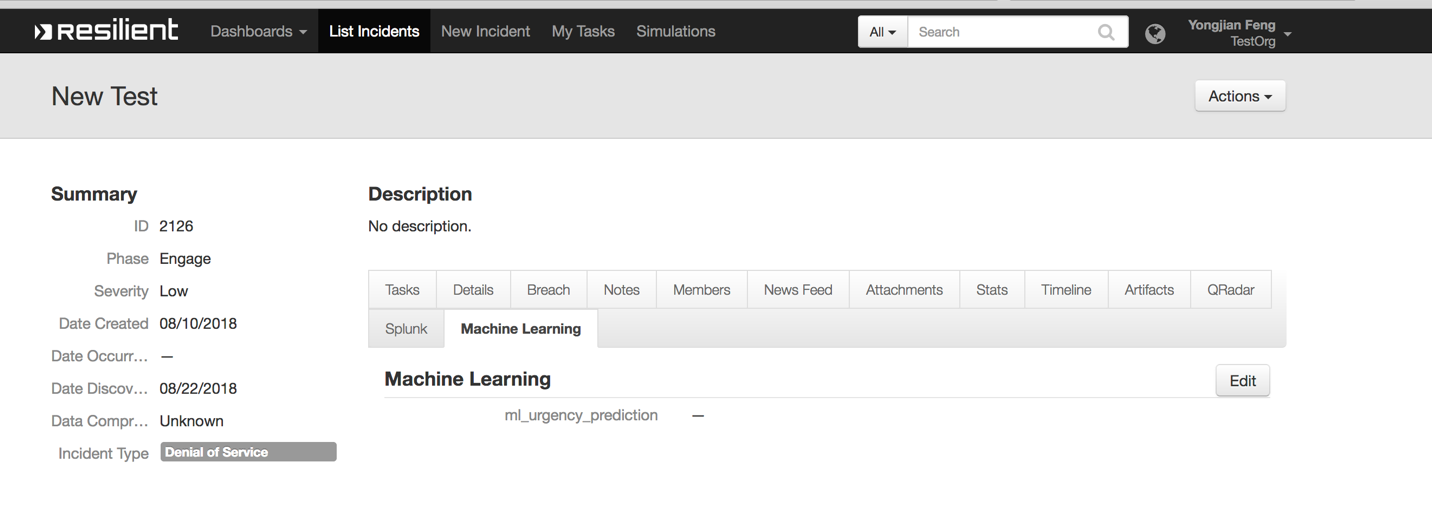
The sample workflow included in this integration is called “ML predict urgency”. A custom field called “ml\_urgency\_prediction” is added to your Resilient platform when you run the following command:

resilient-circuits customize

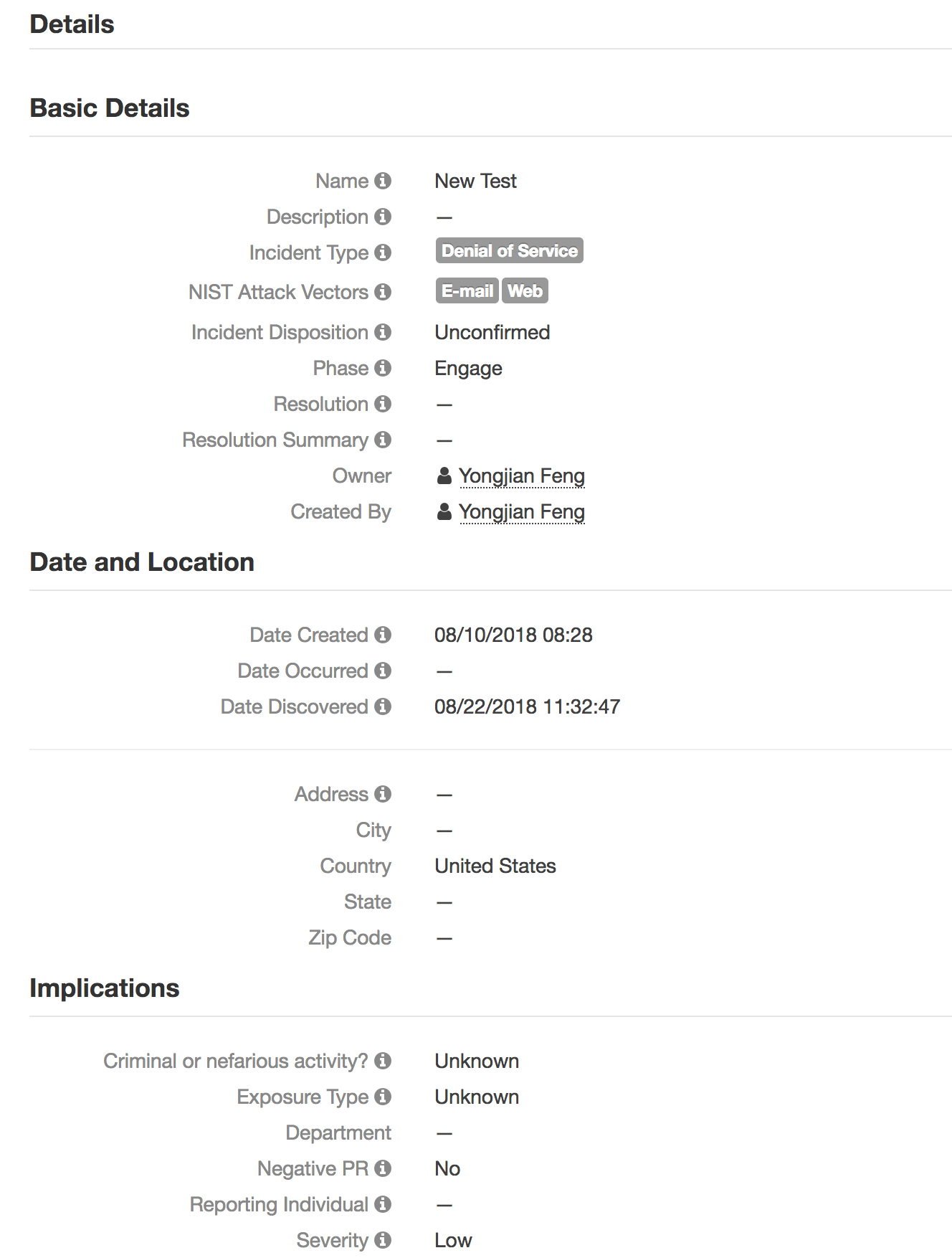
Go to Customization Settings page and create a new incident tab called “Machine Learning”. Then add this custom field into the newly created tab.



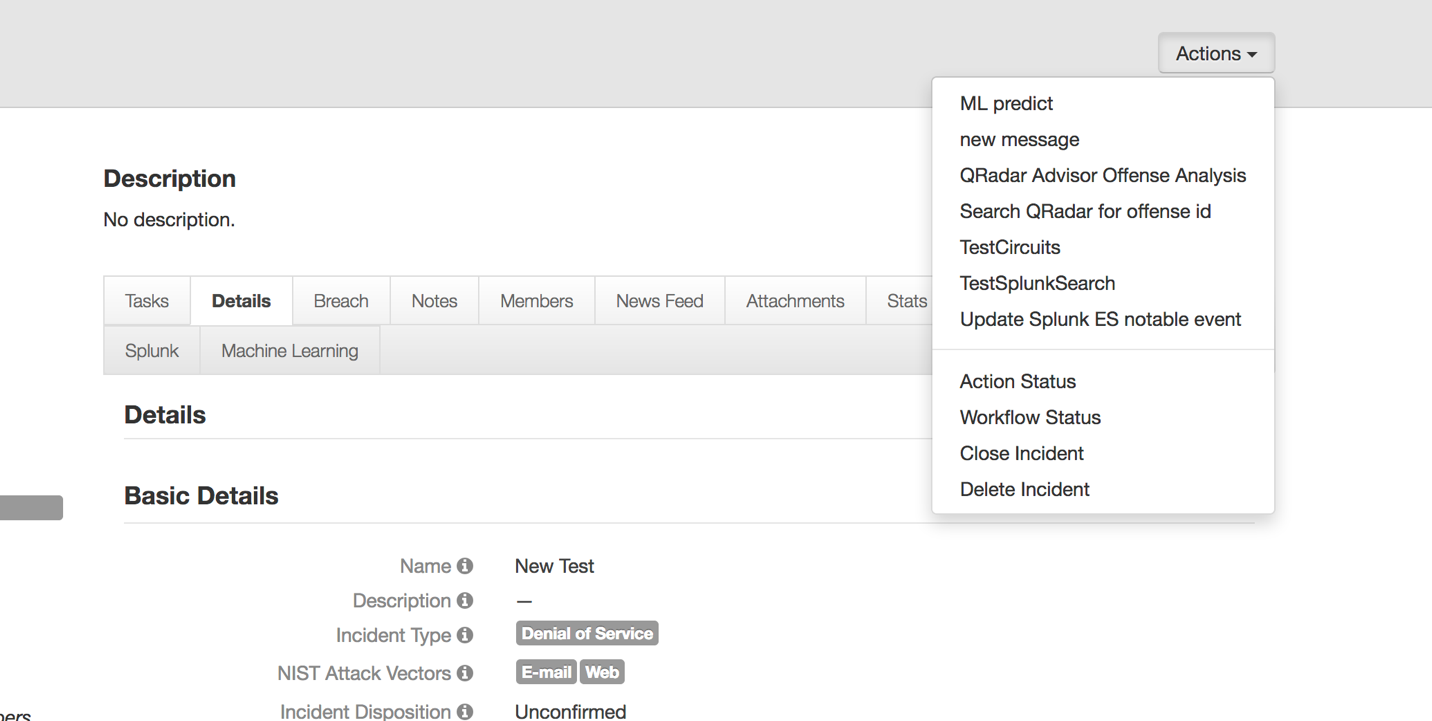
The tab and custom field are added to the incident page



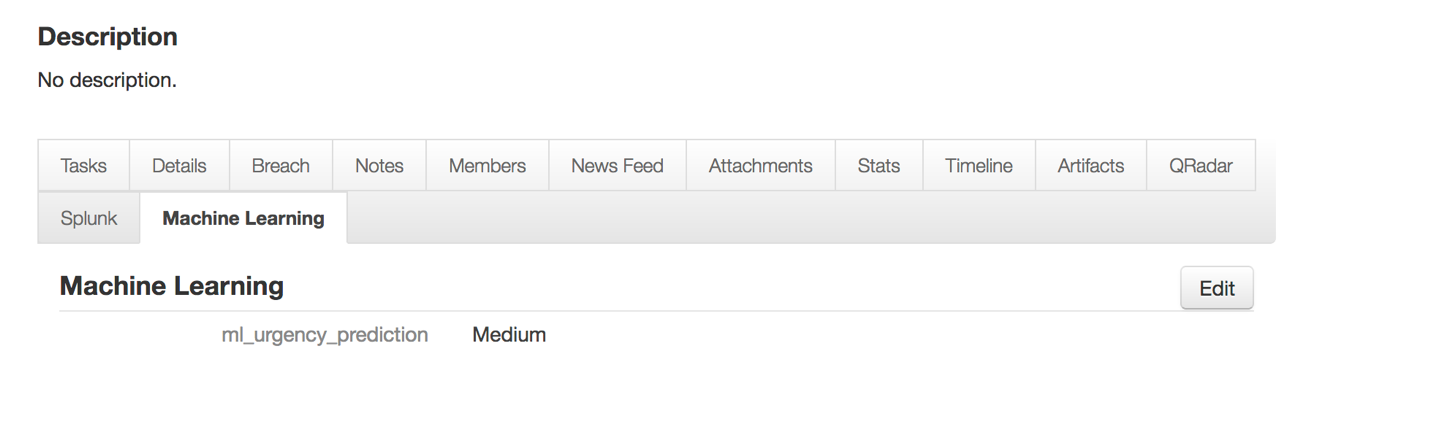
Go to the “Details” tab to make sure that all the incident field features have values. In this particular example, the features are “NIST Attack Vectors”, “Incident Type”, “Incident Disposition”, and “Negative PR”. A prediction will be made using these values.



From the Actions button in an incident, click “ML predict”.



Wait for the integration to finish its job. The prediction is shown.



Getting started

For beginners, there are the steps for getting started.

## Download samples

Once you have your app.config created, you can download incidents and save them into a CSV file. These are the samples to be used to train a machine learning model.

res-ml download -o incident\_samples.csv

## Check value counts

Now you need to make decision on what you want the machine model to predict. One example is “severity\_code”. This is the severity of an incident. Use count\_value command to check if you have an imbalanced dataset.

res-ml count\_value -i incident\_samples.csv -f severity\_code

## Pick features

For this step, you need to pick fields for features. They are the inputs for the machine model, while the predicting field is essentially the output from the machine model. Here you are providing relevant information for the machine learning model. So think about what can affect the predicting field.

Enter all the features into app.config. For example:

Features=incident\_type\_ids, confirmed, negative\_pr\_likely, nist\_attack\_vectors

## Pick an algorithm

Pick an algorithm from 9 supported algorithms. Set this in app.config

algorithm=Logistic Regression

Note there is one special algorithm called “Dummy Classifier”. This shall be used only for baseline comparison. It basically uses only statistics to predict. It is not a machine learning model because it does not attempt to learn the relationship between the features and the predicting. It is used as a baseline, because a successful machine learning model shall do better than this Dummy Classifier.

At this point, the customer can try first the Dummy Classifier to get the baseline.

Then he can try several algorithms. If a model can learn something from the dataset, it shall have its F1 value larger than the one of the Dummy Classifier.

Also different algorithm will give different F1 value, the overall accuracy, and the accuracy for each value. Customer needs to make a decision on which one fits best his need.

## Pick compensation approach

If the customer has an imbalanced dataset, he can enable on of the two supported approach to compensate the dataset.

These approaches can affect the result.

## View a saved model

Once you built and saved several models, you can use the view command to view a summary of a saved model, and select the one you want to use to predict.

## Predict

Move the saved model to the model\_dir folder specified in the app.config, so the Resilient function can use it to predict.

Rename it to the same as the one in the input tab of the workflow if necessary.

## Rebuild

A machine learning model is trained by historical data. Once you accumulate more data, you can rebuild the model. Normally this can be done as a weekly task.

Use the rebuild command to rebuild a model. You just need to specify which saved model you want to rebuild, the saved model will be updated.

Note

In the sample workflow, the prediction is stored in a custom field, ml\_urgency\_prediction. The user needs to manually copy it over to severity\_id (which is the predict field).

If user wants to write the prediction value into severity\_id directly, do this in the post-process script of the sample workflow.

Also, user wants to predict something other than severity\_id, change the post-process script to write to the desired field.

Limitations

Unicode is not supported yet. All features and prediction need to be in ASCII format.

Only classification is implemented. Regression is not yet supported.

Only select fields can be used as features. Free string inputs need Nature Language Processing, which is not supported yet.

Troubleshooting

These are common errors encountered during machine learning:

## The least populated class in y has only 1 members, which is too few

One particular predict value has only one sample. Say for example, 1000 incidents are used as samples for training to predict “severity\_code”. If out of those 1000 samples, only one has severity\_code = high, then this error will be shown.

To verify this, use the count\_value command to check the count values of the field you want to predict.

As for resolution for this, more samples are needed. Specifically, more samples with severity\_code = high are needed.

## This solver needs samples of at least 2 classes in the data, but ….

This means all the samples use to train the model carry the same predict value. For example, 1000 incidents are used for training to predict “severity\_code”. If all of those 1000 incidents have severity\_code = low, then this error will be shown.

To verify this, use the count\_value command to check the count values of the field you want to predict.

This means this is nothing to learn from these samples for predicting that particular field.

## Not samples to train a model

Note that in the pre-process steps, we get rid of all the samples with missing/blank features. As a result, even if you might have a large number of incidents to start with, after filtering those with missing features, you will end up with less number of or even 0 incidents.

This means you need to select features carefully, or even need to fill in the missing features first if possible.