TELSTRA NETWORK PREDICTION FOR FAULT SEVERITY USING PYTHON PROGRAMMING

By AANU BELLO

TELSTRA NETWORK PREDICTION FOR FAULT SEVERITY USING PYTHON PROGRAMMING

BY

AANU TOSIN BELLO

Data Science Intern, Datalab Nigeria

SUBMITTED TO

DATALAB NIGERIA

(www.datalab.com)

APRIL 2021

Contents

1. OVERVIEW

PROBLEM STATEMENT

A Telecom operator called Telstra has presented a dataset where the goal of the project is to use the dataset to predict Telstra network's fault severity due to different disruptions at a time, at a particular location based on the data available. It is required to find out if the fault is a normal glitch or it is critical and will result in total loss of service.

DATA SOURCE

Original dataset is from Telstra provided by Datalab Nigeria for the purpose of using machine learning models to predict the network fault severity.

TOOLS

Tools used for the project are Python, Jupyter Notebook, Numpy, Pandas, Matplotlib, Scikit-Learn, Machine Learning Models (LogisticRegression, KNeighborsClassifier, DecisionTreeClassifier, GaussianNB).

DATA SET

Five Data Sets were used in this project, namely: event_type, log_feature, resource_type, severity_type, train and all are in (.csv)

2. PROJECT STAGES

i. DATA CLEANING

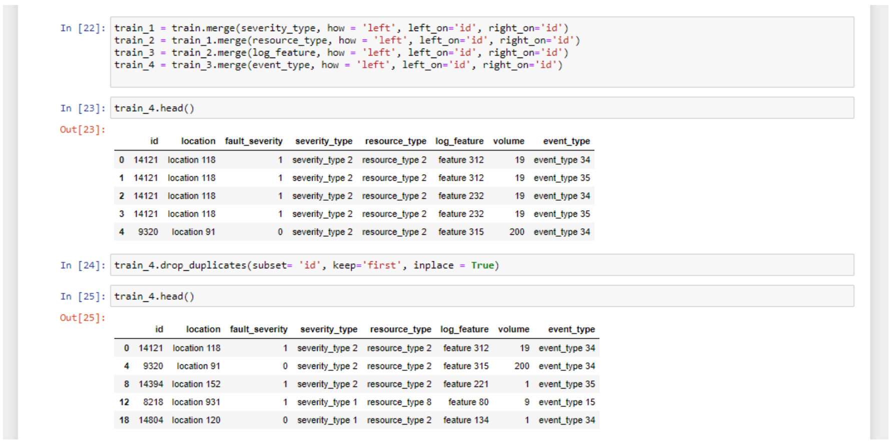
For the project, a fragmented dataset was presented and data cleansing is an essential part of data science, as working with impure data can be detrimental to processes and analysis which can lead to many difficulties. Data cleaning is also an important factor that will determine data quality. Cleaning the data before any process or analysis will help with efficiency, streamlining the margin error and identifying the data type/validity/accuracy.

The platform used is the Jupyter Notebook. It is a data science Integrated Development Environment (IDE) used for data cleaning, data transformation, data visualization, machine learning etc.

Some data cleaning techniques were implemented and they include;

- Data joining
- Getting rid of duplicate values

- Checking for missing values in dataset
- Checking for uniformity
- Avoiding typos and similar errors
- Converting datatype to suitable machine readable form.
- Saving the cleaned dataset for pre-processing.



```
In [22]: train_1 = train.merge(severity_type, how = 'left', left_on='id', right_on='id')
         train_2 = train_1.merge(resource_type, how = 'left', left_on='id', right_on='id')
         train_3 = train_2.merge(log_feature, how = 'left', left_on='id', right_on='id')
         train_4 = train_3.merge(event_type, how = 'left', left_on='id', right_on='id')

In [23]: train_4.head()
Out[23]:
```

	id	location	fault_severity	severity_type	resource_type	log_feature	volume	event_type
0	14121	location 118	1	severity_type 2	resource_type 2	feature 312	19	event_type 34
1	14121	location 118	1	severity_type 2	resource_type 2	feature 312	19	event_type 35
2	14121	location 118	1	severity_type 2	resource_type 2	feature 232	19	event_type 34
3	14121	location 118	1	severity_type 2	resource_type 2	feature 232	19	event_type 35
4	9320	location 91	0	severity_type 2	resource_type 2	feature 315	200	event_type 34

```
In [24]: train_4.drop_duplicates(subset= 'id', keep='first', inplace = True)

In [25]: train_4.head()
Out[25]:
```

	id	location	fault_severity	severity_type	resource_type	log_feature	volume	event_type
0	14121	location 118	1	severity_type 2	resource_type 2	feature 312	19	event_type 34
4	9320	location 91	0	severity_type 2	resource_type 2	feature 315	200	event_type 34
8	14394	location 152	1	severity_type 2	resource_type 2	feature 221	1	event_type 35
12	8218	location 931	1	severity_type 1	resource_type 8	feature 80	9	event_type 15
18	14804	location 120	0	severity_type 1	resource_type 2	feature 134	1	event_type 34

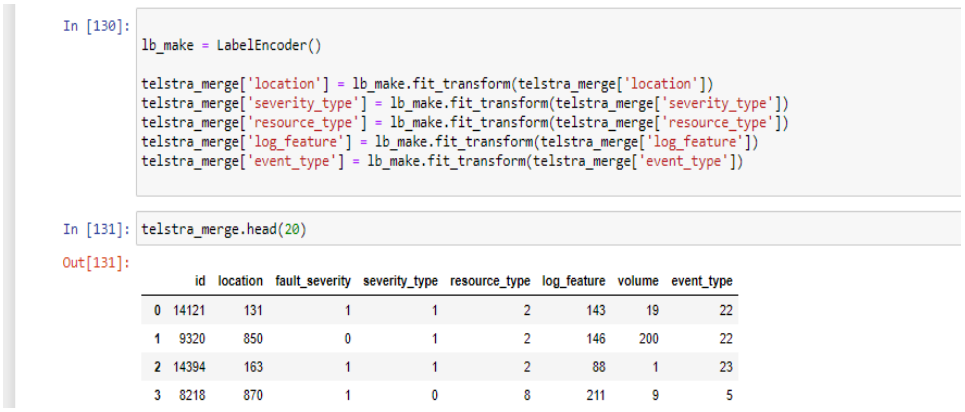
Fig i.1 A figure showing the joined dataset and removing duplicates

ii. DATA TRANSFORMATION

Data transformation is a form of modification that is applied to fine-tune a data. After the data cleaning, some techniques such as normalization also called data pre-processing was applied. Some methods used were

- **Label encoding**: this refers to converting the labels into numeric form so as to convert it into the machine-readable form. It is an important step for a categorical structured dataset.
- **Visualization**: data visualization provides an important suite of tools and techniques for gaining a qualitative understanding and also interpreting data to understand the relationship among variables to determine their relative importance. In this project, box blot and histogram plots were the techniques used for inspecting the underlying frequency distribution of the variables such as outliers, skewness and so on.
- **Outlier detection**: the best practice before proceeding with further analysis on a dataset is to implement an outlier removal phase to rule anomalies. In this project two outlier detection techniques were used. The DBSCAN and the Isolation forest were implemented.

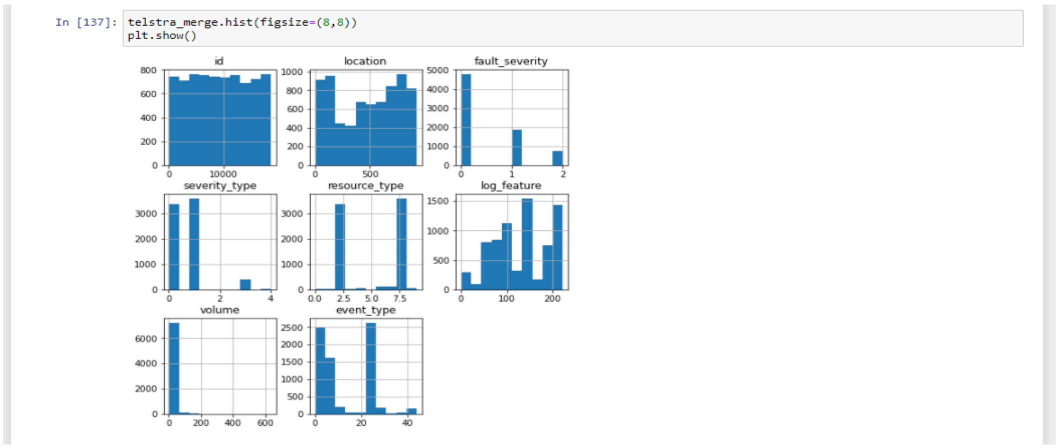
The best performing detection with an accurate ground truth was the Isolation forest.



```
In [130]:
lb_make = LabelEncoder()

telstra_merge['location'] = lb_make.fit_transform(telstra_merge['location'])
telstra_merge['severity_type'] = lb_make.fit_transform(telstra_merge['severity_type'])
telstra_merge['resource_type'] = lb_make.fit_transform(telstra_merge['resource_type'])
telstra_merge['log_feature'] = lb_make.fit_transform(telstra_merge['log_feature'])
telstra_merge['event_type'] = lb_make.fit_transform(telstra_merge['event_type'])

In [131]: telstra_merge.head(20)
Out[131]:
```

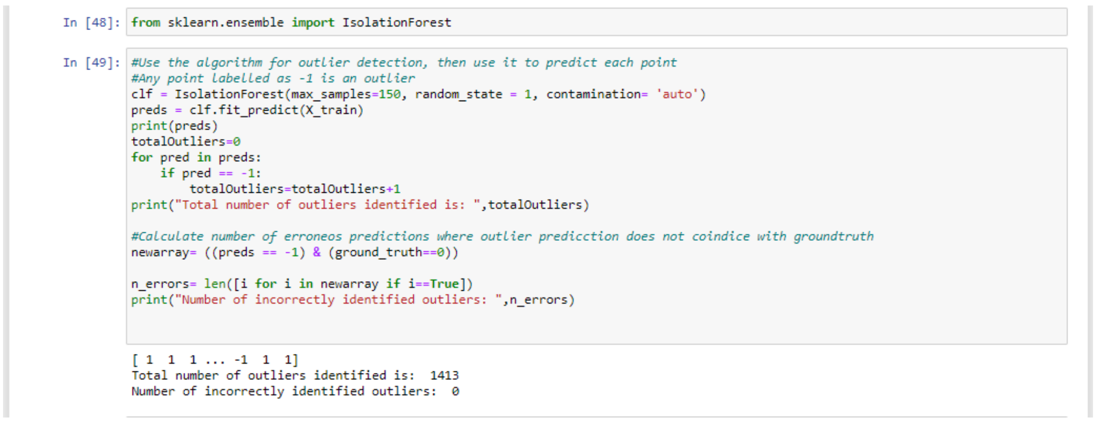
	id	location	fault_severity	severity_type	resource_type	log_feature	volume	event_type
0	14121	131	1	1	2	143	19	22
1	9320	850	0	1	2	146	200	22
2	14394	163	1	1	2	88	1	23
3	8218	870	1	0	8	211	9	5

Fig ii.1 Figure showing label encoded columns



```
In [137]: telstra_merge.hist(figsize=(8,8))
          plt.show()
```

Fig ii.2 A histogram plot of variables

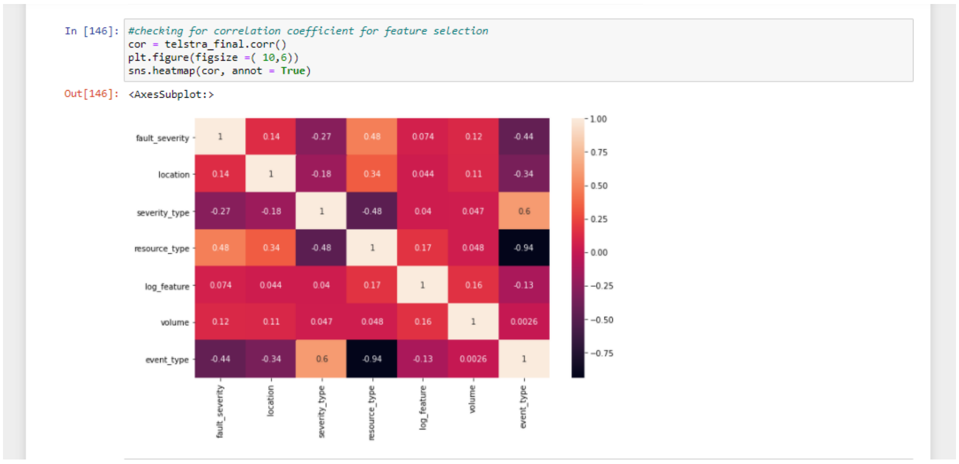


```
In [48]: from sklearn.ensemble import IsolationForest

In [49]: #Use the algorithm for outlier detection, then use it to predict each point
         #Any point labelled as -1 is an outlier
         clf = IsolationForest(max_samples=150, random_state = 1, contamination= 'auto')
         preds = clf.fit_predict(X_train)
         print(preds)
         totalOutliers=0
         for pred in preds:
             if pred == -1:
                 totalOutliers=totalOutliers+1
         print("Total number of outliers identified is: ",totalOutliers)

         #Calculate number of erroneos predictions where outlier predicction does not coindice with groundtruth
         newarray= ((preds == -1) & (ground_truth==0))

         n_errors= len([i for i in newarray if i==True])
         print("Number of incorrectly identified outliers: ",n_errors)

         [ 1  1  1 ... -1  1  1]
         Total number of outliers identified is:  1413
         Number of incorrectly identified outliers:  0
```

Fig ii.3 IsolationForest outlier detector

iii. FEATURE SELECTION FOR MACHINE LEARNING

The goal of feature selection in machine learning is to find the best set of features that allows one to build useful models of studied phenomena. The suitable technique used in the project is the supervised technique. A correlation method under the supervised technique was used to identify the relevant features for increasing the efficiency of the classification model.



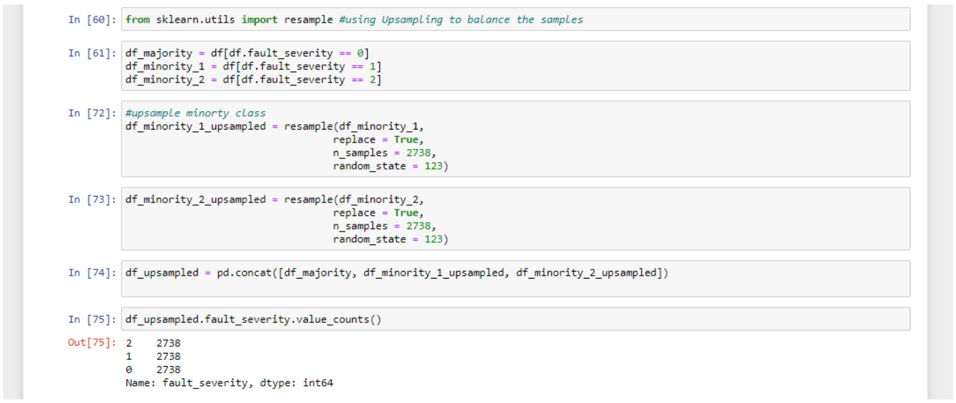
```
In [146]: #checking for correlation coefficient for feature selection
          cor = telstra_final.corr()
          plt.figure(figsize =( 10,6))
          sns.heatmap(cor, annot = True)
Out[146]: <AxesSubplot:>
```

Fig iii.1 Correlation for features selection

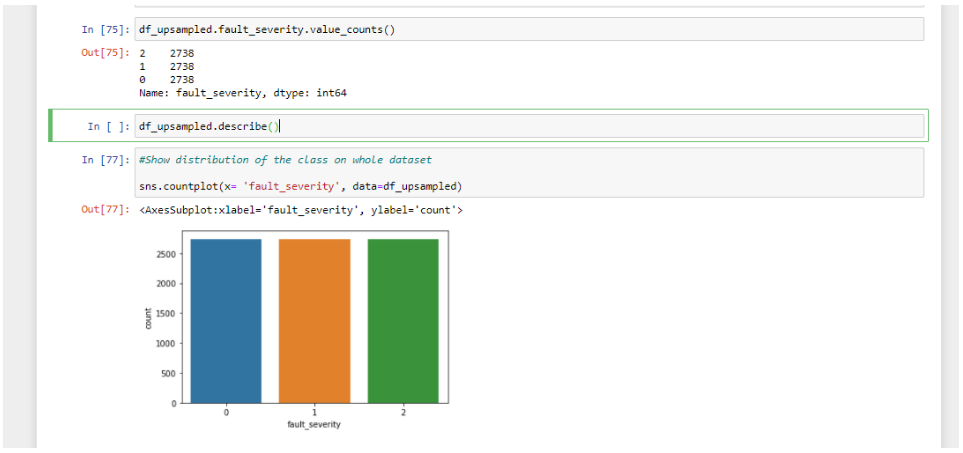
iv. SAMPLING METHOD – HANDLING IMBALANCED DATA

An imbalanced dataset is when training samples are not equally distributed across the target classes. The two main class that are used to tackle the class imbalance is upsampling/downsampling. The sampling process is applied only to the training set and no changes are made to the validation and testing data.

In the project the Upsampling procedure was applied where data points are synthetically generated (corresponding to minority class) and injected into the dataset. After this process, the counts of both labels are almost the same. This equalization procedure prevents the model from inclining towards the majority class.



```
In [60]: from sklearn.utils import resample #using Upsampling to balance the samples

In [61]: df_majority = df[df.fault_severity == 0]
         df_minority_1 = df[df.fault_severity == 1]
         df_minority_2 = df[df.fault_severity == 2]

In [72]: #upsample minorty class
         df_minority_1_upsampled = resample(df_minority_1,
                                            replace = True,
                                            n_samples = 2738,
                                            random_state = 123)

In [73]: df_minority_2_upsampled = resample(df_minority_2,
                                            replace = True,
                                            n_samples = 2738,
                                            random_state = 123)

In [74]: df_upsampled = pd.concat([df_majority, df_minority_1_upsampled, df_minority_2_upsampled])

In [75]: df_upsampled.fault_severity.value_counts()
Out[75]: 2    2738
         1    2738
         0    2738
         Name: fault_severity, dtype: int64
```



```
In [75]: df_upsampled.fault_severity.value_counts()
Out[75]: 2    2738
         1    2738
         0    2738
         Name: fault_severity, dtype: int64

In [ ]: df_upsampled.describe()

In [77]: #Show distribution of the class on whole dataset

         sns.countplot(x= 'fault_severity', data=df_upsampled)
Out[77]: <AxesSubplot:xlabel='fault_severity', ylabel='count'>
```

Fig iv.1 Upsampling method for balancing samples

v. MODEL EVALUATION

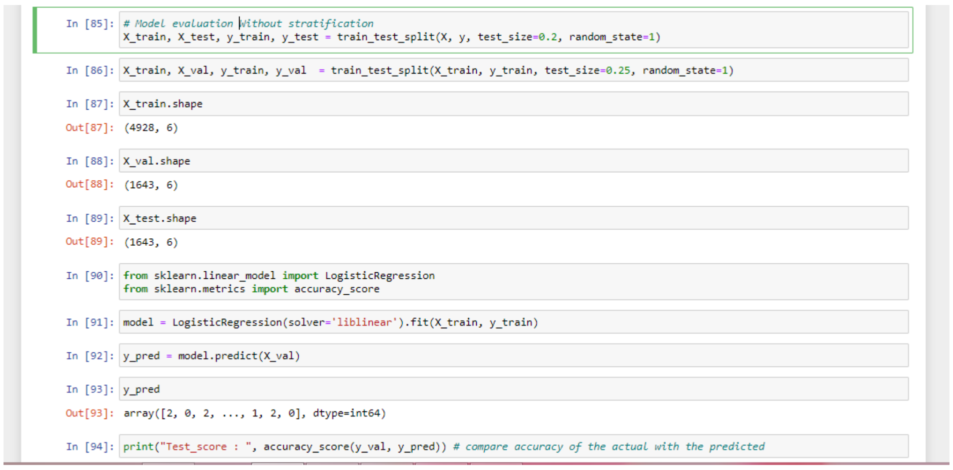
Machine learning models is about giving accurate predictions in order to create real value for a given organization. This step is one of the most important process that needs to be applied to this project in order to achieve the main objective. How a trained model generalizes on unseen data is an important aspect that should be considered in every machine learning process because we need to know whether it actually works and consequently, if we can trust its predictions. Model evaluation aims to estimate the generalization accuracy of a model on future (unseen) data.

In this project the Holdout evaluation method was applied. The purpose of the method is to test a model on different data than it was trained on, and this provides an unbiased estimate of learning performance.

The dataset is randomly divided into three subsets:

1. Training set: is a subset of the dataset used to build predictive models
2. Validation set: is a subset of the dataset used to access the performance of the model built in the training phase. It provides a test platform for fine-tuning a model's parameters and selecting the best performing model.
3. Test set: also known as unseen data, is a subset of the dataset used to assess the likely future performance of the model.

The holdout approach is useful because of its speed, simplicity and flexibility.



```
In [85]: # Model evaluation Without stratification
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

In [86]: X_train, X_val, y_train, y_val  = train_test_split(X_train, y_train, test_size=0.25, random_state=1)

In [87]: X_train.shape
Out[87]: (4928, 6)

In [88]: X_val.shape
Out[88]: (1643, 6)

In [89]: X_test.shape
Out[89]: (1643, 6)

In [90]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score

In [91]: model = LogisticRegression(solver='liblinear').fit(X_train, y_train)

In [92]: y_pred = model.predict(X_val)

In [93]: y_pred
Out[93]: array([2, 0, 2, ..., 1, 2, 0], dtype=int64)

In [94]: print("Test_score : ", accuracy_score(y_val, y_pred)) # compare accuracy of the actual with the predicted
```

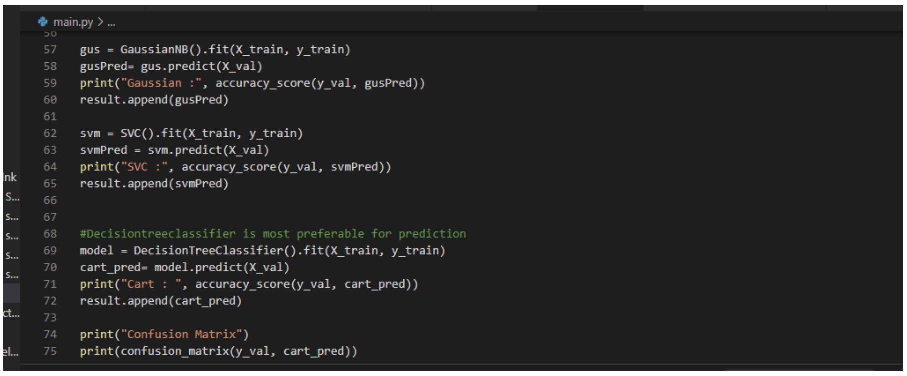
Fig v.1 Model evaluation splitted with the Holdout method

vi. MODEL SELECTION

Model selection is the process of selecting one final machine learning model that addresses a predictive modelling problem from among a collection of candidate machine learning models for a training dataset. Fitting models is relatively straightforward, although selecting among them is the true challenge of applied machine learning. In this project, different models such as Logistic regression, SVM, Gaussian, KNN and the DecisionTreeClassifier were trained on the Telstra dataset. A good enough model to look out for should:

- Meet the requirement and constraints of the project stakeholder Telstra.
- Be sufficiently skillful given the time and resources available.
- Be skillful relative to other tested models.
- Be simpler and easy to understand.

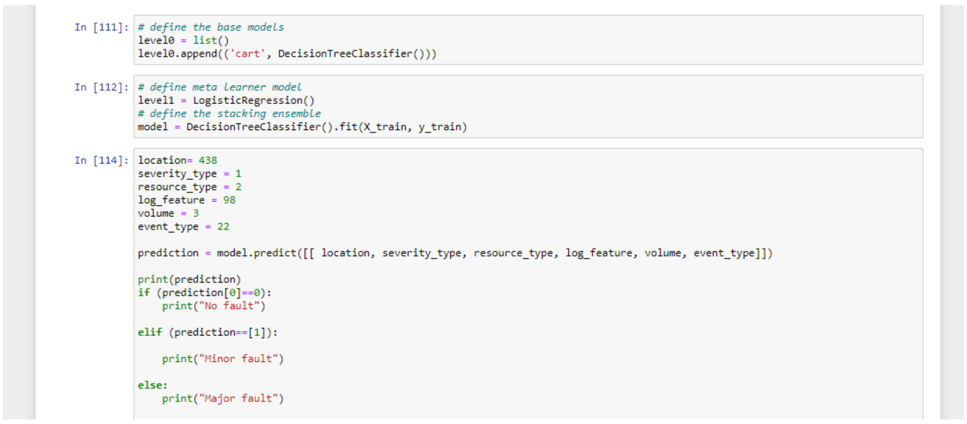
The model selected for this project was the Decisiontree Classifier.



main.py > ...

```
gus = GaussianNB().fit(X_train, y_train)
gusPred= gus.predict(X_val)
print("Gaussian :", accuracy_score(y_val, gusPred))
result.append(gusPred)

svm = SVC().fit(X_train, y_train)
svmPred = svm.predict(X_val)
print("SVC :", accuracy_score(y_val, svmPred))
result.append(svmPred)


#Decisiontreeclassifier is most preferable for prediction
model = DecisionTreeClassifier().fit(X_train, y_train)
cart_pred= model.predict(X_val)
print("Cart : ", accuracy_score(y_val, cart_pred))
result.append(cart_pred)

print("Confusion Matrix")
print(confusion_matrix(y_val, cart_pred))
```

Fig vi.1 Fitting different models for predictions

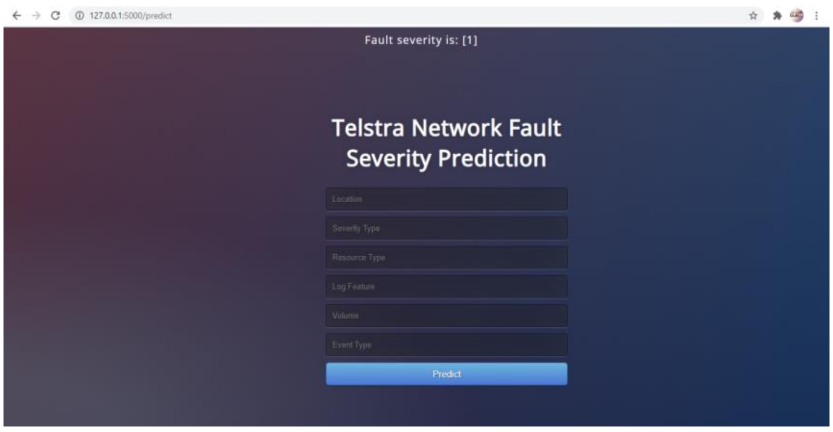


```
In [111]: # define the base models
          level0 = list()
          level0.append(('cart', DecisionTreeClassifier()))

In [112]: # define meta learner model
          level1 = LogisticRegression()
          # define the stacking ensemble
          model = DecisionTreeClassifier().fit(X_train, y_train)

In [114]: location= 438
          severity_type = 1
          resource_type = 2
          log_feature = 98
          volume = 3
          event_type = 22

          prediction = model.predict([[ location, severity_type, resource_type, log_feature, volume, event_type]])

          print(prediction)
          if (prediction[0]==0):
              print("No fault")

          elif (prediction==[1]):

              print("Minor fault")

          else:
              print("Major fault")
```

Fig vi.2 Fitting DecisionTreeClassifier as the preferred model for prediction

vii. MODEL DEPLOYMENT

After fitting the preferred model to the split dataset, some predictions were made to test the model using the features and target set aside for training and it was predicted accurately. Next was to deploy it on the Google Chrome web interface using CSS and HTML templates.



127.0.0.1:5000/predict

Fault severity is: [1]

Telstra Network Fault Severity Prediction

Fig vi.1 Front end of the model deployed on a web application

viii. RECOMMENDATIONS FOR IMPROVEMENT

The project finally came to a conclusion and was executed as expected. Howbeit, there are always room for improvement in terms of using other techniques in finding out the correlating features for accurate ML predictions. Applying a different model other than the preferred one for this project could also address the prediction in a better way.