PREDICTION OF NETWORK DISRUPTION

(A CASE STUDY OF TELSTRA NETWORK)

BY

CHRISTIANA ADERONKE OWOSENI

A Capstone Project on Data Science, using DecisionTree Classifier Model for Fault Severity Prediction

SUBMITTED TO

DATALAB NIGERIA

(www.datalab.com.ng)

APRIL 2021

Table of Content

Overview

Project Cycle

OVERVIEW

Problem Statement: Disruption in communication and connectivity services due to dropped VPN connection, poor connectivity in remote areas and others have been of concern hence the need to predict the severity of the disruption.

Aim: To predict network fault severity at a particular location over a period based on the data available using various machine learning models and deploy the best model.

Tools: Python, Jupyter Notebook, Numpy, Pandas, Matplotlib, Scikit-Learn, Machine Learning Models (LogisticRegression, KNeighborsClassifier, DecisionTreeClassifier, SVC, GaussianNB).

Framework: The framework used for the web application is Django. Django is an open-source web framework that follows the model-template-views architectural pattern.

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)

PROJECT CYCLE

Data Preparation

This stage involves importing of the given the dataset into Jupyter Notebook using the necessary data science libraries as shown in the figure 1.0.

Jupyter Notebook is a data science Integrated Development Environment (IDE) used for data cleaning, data transformation, data visualization, machine learning and so on.

Fig. 1.0

```
# Load libaries
import pandas as pd
import numpy as np
pd.plotting.register_matplotlib_converters()
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Pandas is the library used to read in file(data) into the Jupyter Notebook. Fig. 1.1 Illustrate how the dataset were read into the Notebook.

Fig. 1.1 depicts how the dataset were read into Jupyter Notebook.

```
event_type = pd.read_csv("event_type.csv")
log_feature = pd.read_csv("log_feature.csv")
resource_type = pd.read_csv("resource_type.csv")
severity_type = pd.read_csv("severity_type.csv")
train = pd.read_csv("train.csv")
print("Event Type shape: ",event_type.shape )
print("Log Feature shape: ",log_feature.shape )
print("Resource Type shape: ",resource_type.shape )
print("Severity Type shape: ",severity_type.shape )
print("Train shape: ",train.shape )
```

```
Event Type shape:  (31170, 2)
Log Feature shape:  (58671, 3)
Resource Type shape:  (21076, 2)
Severity Type shape:  (18552, 2)
Train shape:  (7381, 3)
```

Data Preparation also involves merging data sets. Data merging is the process of combining two or more data sets into a single data set, this is necessary because it will enhance data analysis. **Merge** is the keyword used to merge data sets in data science. Fig. 1.2 shows how the five data sets were merged in Jupyter Notebook.

Fig 1.2 Merging the five datasets into a file.

```
# merging data files
print("Merged data file info: ")
print()

train_1 = train.merge(severity_type, how = 'left', left_on='id', right_on='id')
# train_1
train_2 = train_1.merge(resource_type, how = 'left', left_on='id', right_on='id')
# train_2
train_3 = train_2.merge(log_feature, how = 'left', left_on='id', right_on='id')
# train_3
train_4 = train_3.merge(event_type, how = 'left', left_on='id', right_on='id')
train_4
```

```
Merged data file info:
```

	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

DATA CLEANING

This stage involves removing duplicate records in the merged file and checking for null and nan values. **Drop_duplicates()** is the pandas method used to drop duplicates. This method will return a dataframe that has no duplicate. Fig, 2.0 Illustrates the duplicates from the merged file were removed.

Info() is a pandas method used to retrieve information about the merged file. This information includes the count of rows and columns.

Fig 2.0 Duplicate Removal

```
#dropping the duplicate records

train_4.drop_duplicates(subset= 'id', keep= 'first', inplace = True)
train_4
```

```
print("Train_4 info: ")
print(train_4.info())
print()

print("Check for na values:")
print(train_4.isna().sum())
print()

print("Check for null values:")
print(train_4.isnull().sum())
```

```
Train_4 info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7381 entries, 0 to 61836
Data columns (total 8 columns):
```

DATA TRANSFORMATION

This is the process of converting data from one format to another. Here, it is seen that the merged file has both object and integer (int) datatypes. In other to have a useful dataset that will be used in training a machine learning model, it is necessary that we convert all object datatype to integer (int) datatype using Label Encoding as seen in Fig. 2.1.

Label Encoding is a process of converting labels into numeric form. Label Encoding encodes labels with a value between 0 and n_classes – 1, where n is the number of distinct labels. Fig.2.0, depicts what the Labels in the merged data file.

Fig. 2.0 Merged Data Label Illustration.

]:

	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

Fig. 2.1 Data Transformation Using Label Encoding

```
# Label encoding
for col in train_4.select_dtypes(include=['object']).columns:
    encoder = LabelEncoder()
    train_4[col] = encoder.fit_transform(train_4[col])

train_4
```

	id	location	fault_severity	severity_type	resource_type	log_feature	volume	event_type
0	14121	131	1	1	2	143	19	22
4	9320	850	0	1	2	146	200	22
8	14394	163	1	1	2	88	1	23
12	8218	870	1	0	8	211	9	5

FEATURE SELECTION

This is a process of selecting a subset of relevant features in a data set that contributes most too the target variable. This selection can be done automatically or manually. This selection is done to improve accuracy scores. Here, the features are selected using **SelectKBest** which is a class in the **sklearn.feature_selection** as seen in the figure 3.1.

The target variable in this data set (train_4) is **fault_severity**, while others are the features excluding the id. Fig 3.0 illustrates this.

Fig. 3.0 Target and Features

	id	location	fault_severity	severity_type	resource_type	log_feature	volume	event_type
0	14121	131	1	1	2	143	19	22
4	9320	850	0	1	2	146	200	22
8	14394	163	1	1	2	88	1	23
12	8218	870	1	0	8	211	9	5

Fig. 3.1 Feature Selection

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# features(X) = ['location', 'severity_type', 'resource_type', 'log_feature', 'event_type', 'volume']
X = train_4.iloc[:,[1,3,4,5,6,7]]
y = train_4.iloc[:,2]
print()
chi2_features = SelectKBest(score_func=chi2, k=4)
X_Kbest_features = chi2_features.fit_transform(X, y)
print("Original feature number: ", X.shape[1])
print('Reduced feature number: ', X_Kbest_features.shape[1])
print("Best features are: ")
print(X.iloc[:,[0,3,4,5]])
```

```
Original feature number:  6
Reduced feature number:  4
Best features are:
      location  log_feature  volume  event_type
0          131          143      19          22
4          850          146     200          22
```

Fig. 3.1 shows that the best features in the data set **train_4** are **location, log_feature, volume and event_type.** In view of this, the target (fault_severity) and the best features will be sliced from train_4 data set. Fig. 3.2 depicts how this was done.

Fig. 3.2 Data Set Slicing into target and Best Features

```
# Spliting train_4 into target(y) and features(X)
y = train_4.iloc[:,1]
X = train_4.iloc[:,[0,2,3,4]]
print("Target is: ")
print(y)
print("Features are: ")
print(X)
```

```
Target is:
0        1
4        0
8        1
12       1
18       0
        ..
61824    0
61828    0
61830    2
61832    0
61836    0
Name: fault_severity, Length: 7381, dtype: int64
Features are:
       location  log_feature  volume  event_type
0           131          143      19          22
4           850          146     200          22
8           163           88       1          23
```

The result of the sliced data was further divided into train, test and val (validation) data set. This was done to make sampling and model training easy. Fig. 3.3 shows how this was done.

Fig. 3.3 Splitting Target and Features into Train, Test and Validation Data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

X_train, X_val, y_train, y_val  = train_test_split(X_train, y_train, test_size=0.25, random_state=1)
```

```
#Training data
X_train.shape
```

```
(4428, 4)
```

SAMPLING

Data Sampling is a technique used to select, manipulate, and analyse subset of data to identify patterns and trends.

In other to have an unbiased sample, extreme values that deviate from other observations on the data set (known as **Outliers**) must be identified and removed. Fig. 4.1 shows how Outliers on the data set were identified and removed.

Fig. 4.1 Outlier Detection and Removal

```
#import the implementation of this algorihm from sklearn
from sklearn.cluster import DBSCAN

#Use the algorithm for outlier detection, the retun in clusters will show the membership of each point
#Any point labelled as -1 is an outlier

outlier_detection = DBSCAN(min_samples = 3, eps = 3)
clusters = outlier_detection.fit_predict(X_train)

#Count total number of outliers as count of those labelled as -1
TotalOutliers=list(clusters).count(-1)
print("Total number of outliers identified is: ",TotalOutliers)
```

```
Total number of outliers identified is:  2040
```

```
# select all rows that are not outliers and update the X_train and y_train.
mask = clusters != -1
X1_train, y1_train = X_train[mask], y_train[mask]

# summarize the shape of the updated training dataset
print(X1_train.shape, y1_train.shape)
X1_train.head(5)
```

```
(2388, 4) (2388,)
```

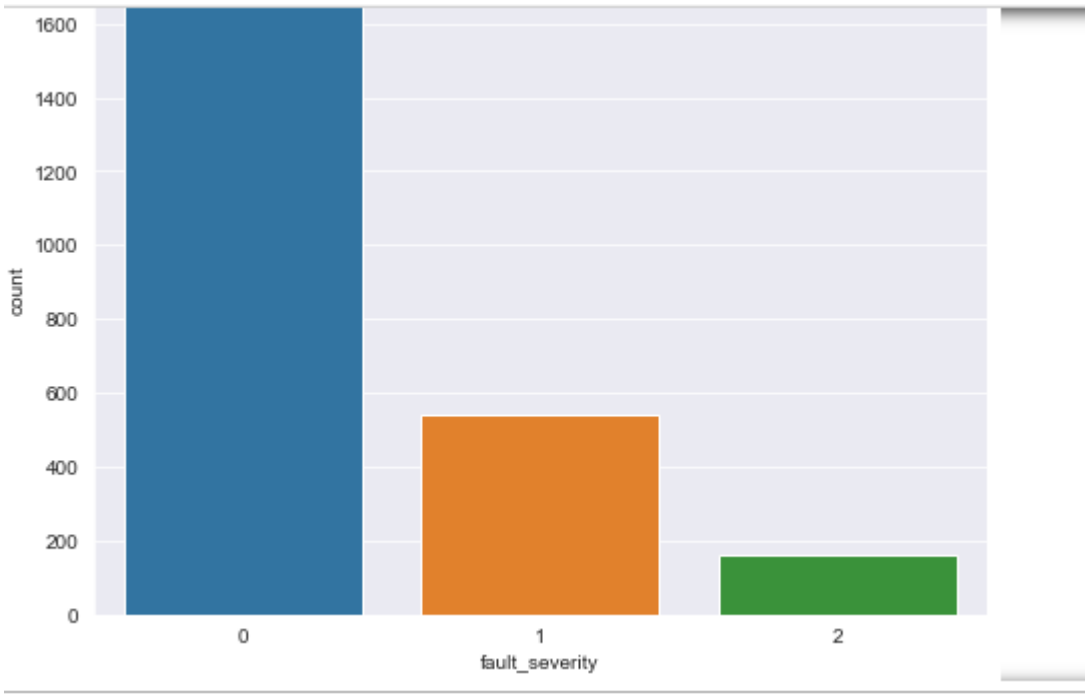
In this project, the statistical feature of the target class (**fault_severity**) which ranges from 0 - 2 shows that the target is not represented equally in the data set, fig.4.2 illustrate this.

Fig. 4.2 Visual Representation of the Target Class (fault_Severity)

```
y1_train.value_counts()
```

```
0    1684
1     542
2     162
Name: fault_severity, dtype: int64
```

```
#Show distribution of the  fault_severity
plt.figure(figsize=(8,6))
sns.countplot(y1_train)
plt.show()

print()

#Show distribution of the  fault_severity
sns.distplot(y1_train)
plt.show()
```



In view of this, **UpSampling** was done on the minority classes within the target classes. The minority classes are class 1 and class 2. Fig. 4.3 illustrates how the minority classes were upsampled to class 0.

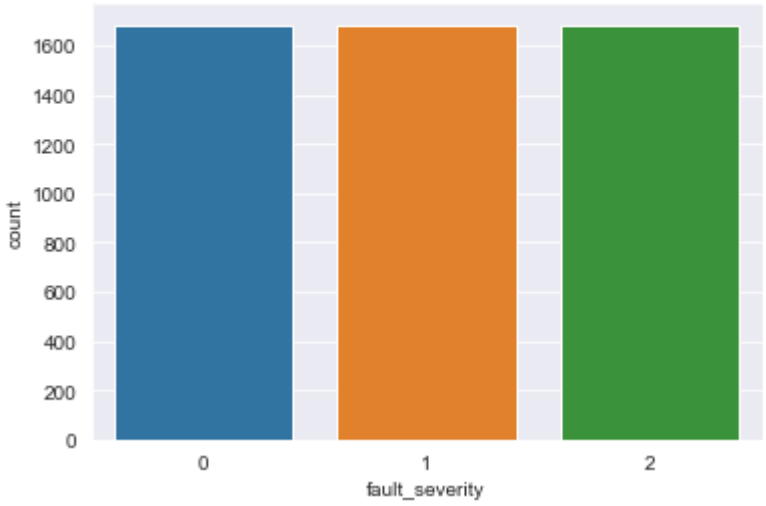
Fig. 4.3 UpSampling Minority Classes

```
from sklearn.utils import resample

df_majority = df[df.fault_severity == 0]
df_minority = df[df.fault_severity == 1]
df_minority_2 =  df[df.fault_severity == 2]

# upsample minority class fault = 1
df_minority_1_upsampled = resample(df_minority,
                                   replace=True,
                                   n_samples=1684,
                                   random_state=123)

# upsample minority class fault = 2
df_minority_2_upsampled = resample(df_minority_2,
                                   replace=True,
                                   n_samples=1684,
                                   random_state=123)

# combine minority class fault_1 anf fault_2 with majority class
df_upsampled = pd.concat([df_majority, df_minority_1_upsampled, df_minority_2_upsampled])

# display new class counts
print("upsampled target is: ")
print(df_upsampled.fault_severity.value_counts())
```

```
upsampled target is:
2    1684
1    1684
0    1684
Name: fault_severity, dtype: int64
```

```
#Show distribution of the class on whole dataset
sns.countplot(x= 'fault_severity', data=df_upsampled)
```

```
<AxesSubplot:xlabel='fault_severity', ylabel='count'>
```



MODEL EVALUATION

In this project stage, various Machine Learning Models such as DecisionTreeClassifier, KNeigbhorsClassifier, LogisticRegression, SVC, GaussianNB were evaluated to determine the model that has learnt well and has the highest accuracy with respect to the data set. Fig. 5.0 illustrates this.

Fig. 5.0 Model Evaluation

```
# compare standalone models for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
```

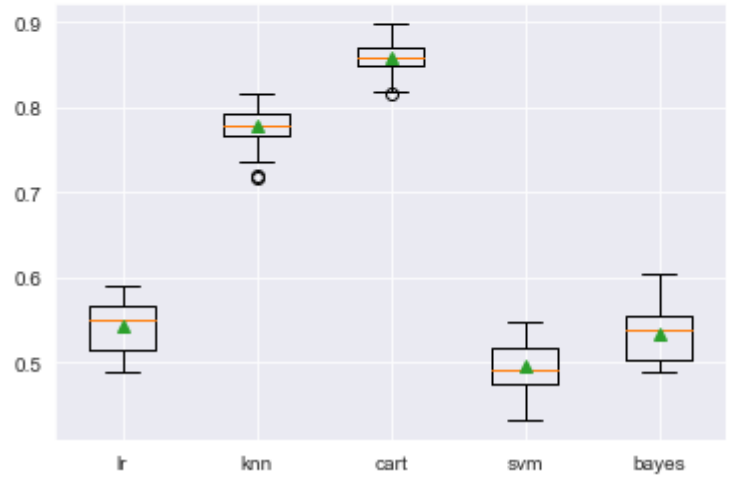
```
# get a list of models to evaluate
def get_models():
    models = dict()
    models['lr'] = LogisticRegression()
    models['knn'] = KNeighborsClassifier()
    models['cart'] = DecisionTreeClassifier()
    models['svm'] = SVC()
    models['bayes'] = GaussianNB()
    return models
```

```
# evaluate a given model using cross-validation
#The evaluate_model() function below takes a model instance and returns a list of scores from three repeats of stratified

def evaluate_model(model,X_train, y_train):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X_train, y_train, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
    return scores
```

```
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results.append(scores)
    names.append(name)
    print('>%s %.2f (%.2f)' % (name, mean(scores), std(scores)))
```

```
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

```
>lr 0.54 (0.03)
>knn 0.78 (0.02)
>cart 0.86 (0.02)
>svm 0.49 (0.03)
>bayes 0.53 (0.03)
```



The above model evaluation shows that Decision Tree Classifier model performed better with a mean accuracy of 86%.

MODEL SELECTION

The Model Evaluation phase showed that Decision Tree Classifier Model has performed better with a mean accuracy of 86 %. In view of this, fault severity prediction was made using Decision Tree Classifier. Pickle, a data science library was be used to save this choice model. Fig. 6.0 shows how the chosen model was saved.

Fig. 6.0 Testing and Saving Decision Tree Classifier

```
cart = DecisionTreeClassifier()
cart.fit(X, y)

# make a prediction for one example
data = [[86,213,21,5]]
predicted_fault = cart.predict(data)
print('Predicted Class: %d' % (predicted_fault))
if predicted_fault == 0:
    print("This implies that the fault severity is low")
elif predicted_fault == 1:
    print("This implies that the fault severity is medium")
elif predicted_fault == 2:
    print("This implies that the fault severity is high")
else:
    print("Fault can not be categorised")
```

```
Predicted Class: 2
This implies that the fault severity is high
```

```
# save the model to disk
import pickle

filename = 'faultSeverity_classifierModel.sav'
pickle.dump(cart, open(filename, 'wb'))
```

DEPLOYMENT

A web application was developed using Django Framework in other to allow users make predictions on network fault severity by leveraging on the saved model **faultSeverity_classifierModel.sav**.

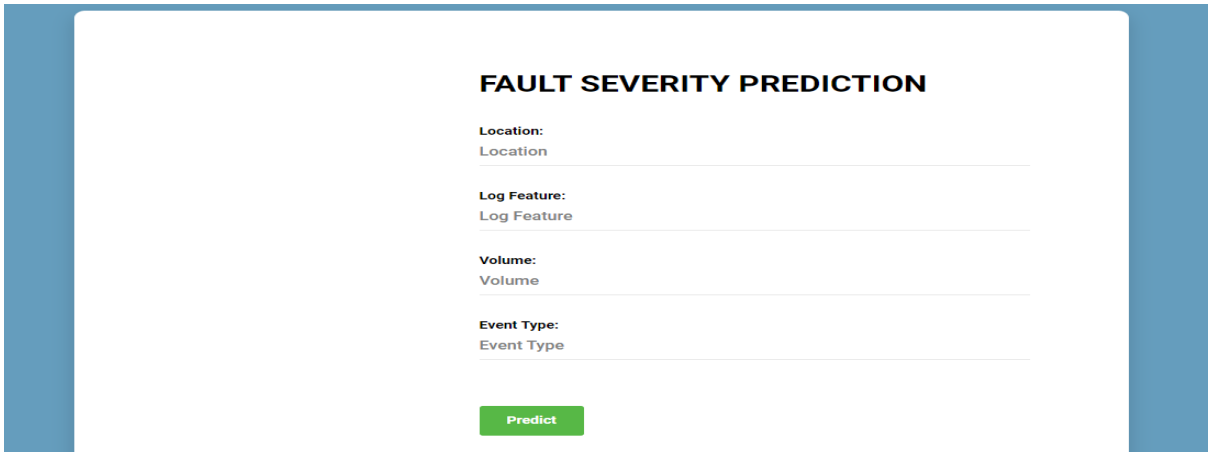
This model was loaded into the Django project using **pickle library.** Below is a snapshot of how the saved model is loaded into and project.

Fig. 7.1 Load faultSeverity_classifierModel.sav

```
# Load the choice model
filename = 'faultSeverity_classifierModel.sav'
loaded_model = pickle.load(open(filename, 'rb'))
# predict using the inputted values
predicted_fault = loaded_model.predict(data)
```

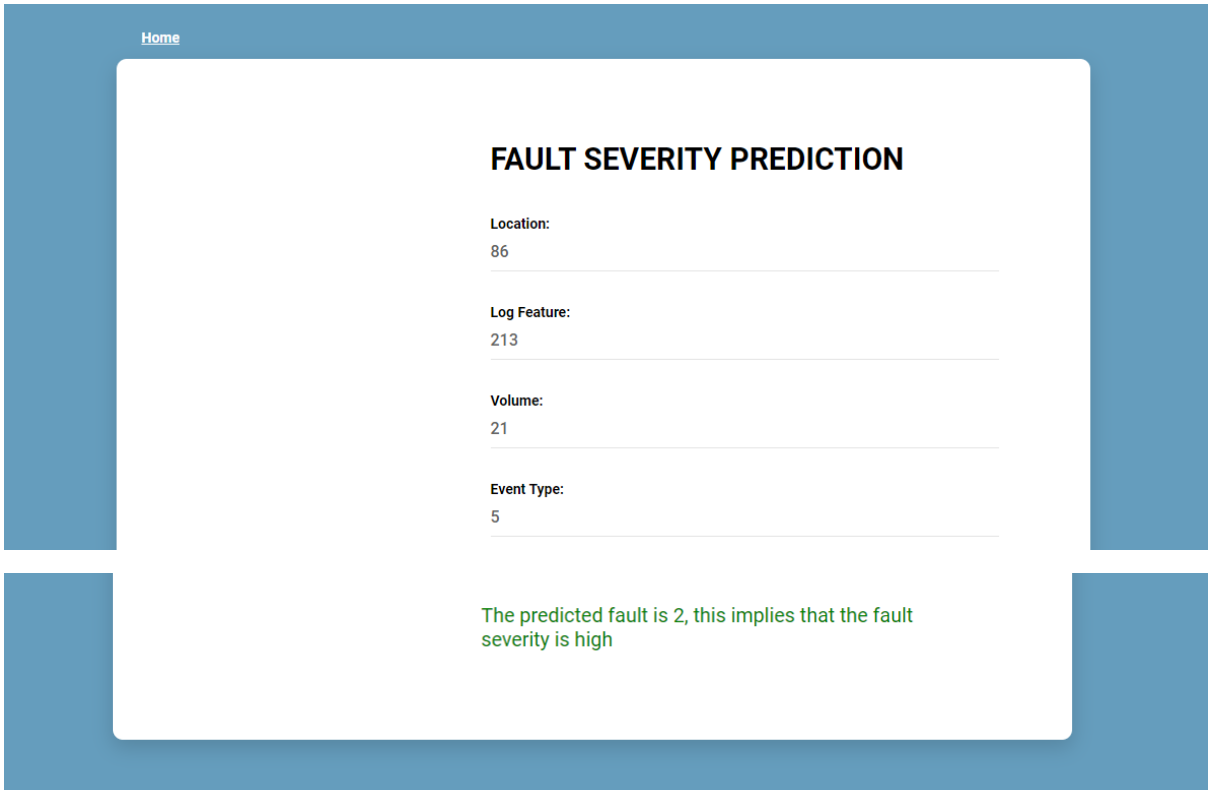
The home page of the webApp can be accessed using **http://127.0.0.1:8000/home**

Fig. 7.2 Fault Severity Prediction home page



The following test data was used to test the model on the web, these are {'location': 86, 'log_feature': 213, 'volume': 21, 'event_type': 5}. Fig. 7.3 illustrates this.

Fig. 7.3 WebApp Fault Severity Prediction



RECOMMENDATION

It is recommended that other Feature Selection methods for classification such as f_classif, mutual_info_classif be used to select the best feature so that the performance of the selected machine learning model (Decision Tree Classifier) can be further evaluated. Also, stratified sampling method can be used in sampling the data set.