Python Code for Fire Fatality Profiling Model

Alfi Gözaçan*

December 16, 2021

*Humberside Fire and Rescue Service

Dataset Name	Dimensions (Excluding Headers)	Description	Hyperlink
`casualties.csv`	(85137, 18)	All casualties in FRS incidents	Click here
`fatalities.csv`	(3126, 15)	All fatalities in FRS incidents	Click here

Table 1: Table of datasets used.

1 Overview

The code used in this project can be split into four parts: data cleaning, model training, and model diagnostics. It is recommended to first read through the report that explains the chosen methods in more detail before reading the code (which can be found here). Now, the following snippet lists the modules to be imported.

```
import numpy as np
import pandas as pd
import scikitplot as skplt
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

2 Data Cleaning

Table 1 summarises the data collected and used in the fire fatality profiling. Each dataset is available in `.csv` format (possibly after converting from `.ods`) and will be referred to in any Python scripts by its name as listed in Table 1.

Firstly, the data is imported into a Python environment / Jupyter Notebook using `pandas`.

```
casualties_df = pd.read_csv("C:\\path_to_data\\casualties.csv")
fatalities_df = pd.read_csv("C:\\path_to_data\\fatalities.csv")
```

Then, any entries with "other" recorded in a column are removed.

```
drop_cas_ids = list(
    set.union(
        set(casualties_df[casualties_df["CAUSE_OF_FIRE"] == "Unspecified cause"].index),
        set(casualties_df[casualties_df["SOURCE_OF_IGNITION"] == "Other/ Unspecified"].index),
        set(casualties_df[casualties_df["FIRE_START_LOCATION"] == "Other"].index),
        set(casualties_df[casualties_df["VICTIM_AGE"] == "Unspecified"].index),
        set(casualties_df[casualties_df["VICTIM_GENDER"] == "Not known"].index)
    )
)

drop_fat_ids = list(
    set.union(
        set(fatalities_df[fatalities_df["CAUSE_OF_FIRE"] == "Unspecified cause"].index),
```

```
        set(fatalities_df[fatalities_df["SOURCE_OF_IGNITION"] == "Other/ Unspecified"].index),
        set(fatalities_df[fatalities_df["FIRE_START_LOCATION"] == "Other"].index),
        set(fatalities_df[fatalities_df["VICTIM_AGE"] == "Unspecified"].index),
        set(fatalities_df[fatalities_df["VICTIM_GENDER"] == "Not known"].index)
    )
)

casualties_df.drop(drop_cas_ids, axis=0, inplace=True)
fatalities_df.drop(drop_fat_ids, axis=0, inplace=True)
```

Next, the response variable `FATALITY` is initiated by inserting a 0 into all `casualties.csv` entries and a 1 into all `fatalities.csv` entries.

```
casualties_df["FATALITY"] = [0 for i in range(len(casualties_df))]
fatalities_df["FATALITY"] = [1 for i in range(len(fatalities_df))]
```

Then, the columns which are not to be used in the model are removed.

```
casualties_df.drop(["FINANCIAL_YEAR",
                    "OCCUPANCY_STATUS",
                    "VICTIM_LOCATION_START",
                    "VICTIM_LOCATION_FOUND",
                    "VICTIM RESCUED",
                    "INJURY_TYPE",
                    "INJURY_SEVERITY",
                    "FIRE_RELATED",
                    "CASUALTY_TOTAL"], axis=1, inplace=True)

fatalities_df.drop(["FINANCIAL_YEAR",
                    "VICTIM_LOCATION_START",
                    "VICTIM_LOCATION_FOUND",
                    "FATALITY_CIRCUMSTANCES_DESCRIPTION",
                    "FATALITY_CAUSE",
                    "FATALITY_TOTAL"], axis=1, inplace=True)
```

After that, the casualties dataset is randomly undersampled so that the two datasets are balanced and can be concatenated vertically.

```
np.random.seed(0)
drop_indices = np.random.choice(a=casualties_df.index,
                                size=len(casualties_df)-len(fatalities_df),
                                replace=False)
casualties_df.drop(drop_indices, axis=0, inplace=True)
combined_df = pd.concat([casualties_df, fatalities_df])
```

Finally, the dataset is dummy encoded so that the models can deal with the categorical data.

```
encoder = OneHotEncoder(drop="first", sparse=False)
dummy_view = encoder.fit_transform(combined_df)
df = pd.DataFrame(dummy_view)
df.columns = encoder.get_feature_names(combined_df.columns)
```

3 Model Training

In this section of the code, the model is being trained by the machine learning algorithms. Firstly, the code is split into training and testing data.

```
ncols = len(df.columns)
training_set, test_set = train_test_split(df, test_size = 0.3, random_state=1)
```

Then, in statistical terms, the design matrix X and response y are extracted.

```
X_train = training_set.iloc[:,:-1]
y_train = training_set.iloc[:,-1]
X_test = test_set.iloc[:,:-1]
y_test = test_set.iloc[:,-1]
```

Next, the algorithms are trained using the training data. Note the fixed random seed, to ensure reproducibility of the results.

```
adaboost = AdaBoostClassifier(random_state=1)
adaboost.fit(X_train, y_train)

rf = RandomForestClassifier(random_state=1)
rf.fit(X_train, y_train)

logreg = LogisticRegression(random_state=1)
logreg.fit(X_train, y_train)

xgboost = GradientBoostingClassifier(random_state=1)
xgboost.fit(X_train, y_train)
```

4 Model Diagnostics

The last section of code prints out some performance metrics / graphs of the models. Firstly, the prediction results are appended onto the test dataset.

```
y_ada_pred = adaboost.predict(X_test)
test_set.insert(ncols, "AdaBoost Predictions", y_ada_pred)

y_rf_pred = rf.predict(X_test)
test_set.insert(ncols+1, "RF Predictions", y_rf_pred)

y_lr_pred = logreg.predict(X_test)
test_set.insert(ncols+2, "LogReg Predictions", y_lr_pred)

y_xg_pred = xgboost.predict(X_test)
test_set.insert(ncols+3, "XGBoost Predictions", y_xg_pred)
```

Then, classification matrices are produced, recording the precision, recall, and f1-score of the algorithms.

```
print("AdaBoost:\n", classification_report(test_set.iloc[:,ncols-1],
                                           test_set.iloc[:,ncols]))
print("Random Forest:\n", classification_report(test_set.iloc[:,ncols-1],
                                                 test_set.iloc[:,ncols+1]))
```

```
print("Logistic Regression:\n", classification_report(test_set.iloc[:,ncols-1],
                                                      test_set.iloc[:,ncols+2]))
print("XGBoost\n:", classification_report(test_set.iloc[:,ncols-1],
                                          test_set.iloc[:,ncols+3]))
```

Next, the overall accuracy of the algorithms is computed and printed.

```
length = len(test_set.iloc[:,ncols-1])

ada_no_matched = sum([(test_set.iloc[i,ncols-1] * test_set.iloc[i,ncols]) +
                      ((1-test_set.iloc[i,ncols-1]) * (1-test_set.iloc[i,ncols]))
                      for i in range(length)])
rf_no_matched = sum([(test_set.iloc[i,ncols-1] * test_set.iloc[i,ncols+1]) +
                     ((1-test_set.iloc[i,ncols-1]) * (1-test_set.iloc[i,ncols+1]))
                     for i in range(length)])
lr_no_matched = sum([(test_set.iloc[i,ncols-1] * test_set.iloc[i,ncols+2]) +
                     ((1-test_set.iloc[i,ncols-1]) * (1-test_set.iloc[i,ncols+2]))
                     for i in range(length)])
xg_no_matched = sum([(test_set.iloc[i,ncols-1] * test_set.iloc[i,ncols+3]) +
                     ((1-test_set.iloc[i,ncols-1]) * (1-test_set.iloc[i,ncols+3]))
                     for i in range(length)])

ada_accuracy = ada_no_matched / length
rf_accuracy = rf_no_matched / length
lr_accuracy = lr_no_matched / length
xg_accuracy = xg_no_matched / length

print("AdaBoost Proportion Correctly Guessed:", round(ada_accuracy, 4))
print("Random Forest Proportion Correctly Guessed:", round(rf_accuracy, 4))
print("Logistic Regression Proportion Correctly Guessed:", round(lr_accuracy, 4))
print("XGBoost Proportion Correctly Guessed:", round(xg_accuracy, 4))
```

Finally, some ROC curves are drawn of each of the algorithms, also reporting the AUC scores.

```
adaprobs = adaboost.predict_proba(X_test)
rfprobs = rf.predict_proba(X_test)
lrprobs = logreg.predict_proba(X_test)
xgprobs = xgboost.predict_proba(X_test)

probas = [adaprobs, rfprobs, lrprobs, xgprobs]
titles = ["AdaBoost", "Random Forest", "Logistic Regression", "XGBoost"]

for i in range(len(probas)):

    skplt.metrics.plot_roc(y_test, probas[i], title=titles[i])

plt.show()
```