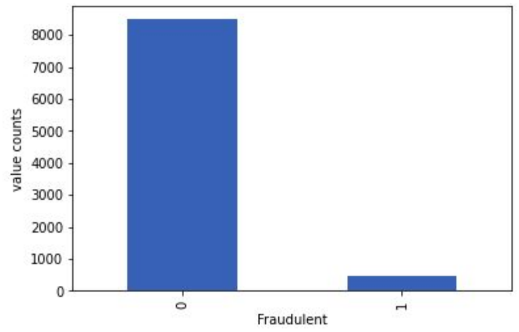
Foundation of Data Science: Project Report

Prajwal Krishna

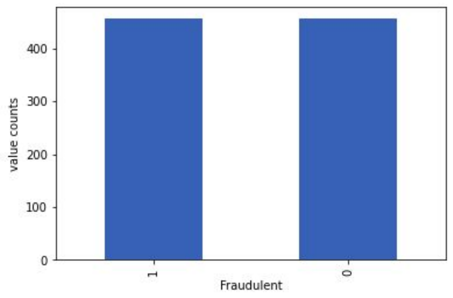
DATA ANALYSIS

- **Data imbalance problem**
  - After plotting the data, it was found that the fraudulent which is the label column is highly imbalanced as shown below
  - This imbalance in data was handled while fitting the model.

```
In [14]:
import matplotlib.pyplot as plt
plt.xlabel("Fraudulent")
plt.ylabel("value counts")
data_frame.fraudulent.value_counts().plot.bar()

Out[14]: <AxesSubplot:xlabel='Fraudulent', ylabel='value counts'>
```

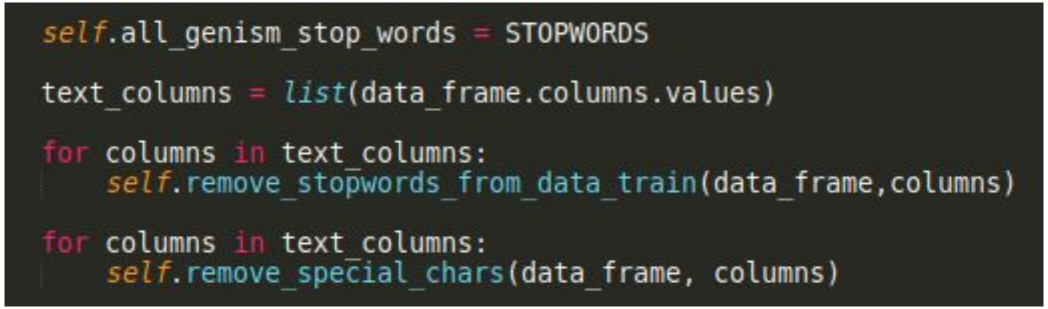


```
In [10]:
import matplotlib.pyplot as plt
plt.xlabel("Fraudulent")
plt.ylabel("value counts")
test_under.fraudulent.value_counts().plot.bar()

Out[10]: <AxesSubplot:xlabel='Fraudulent', ylabel='value counts'>
```

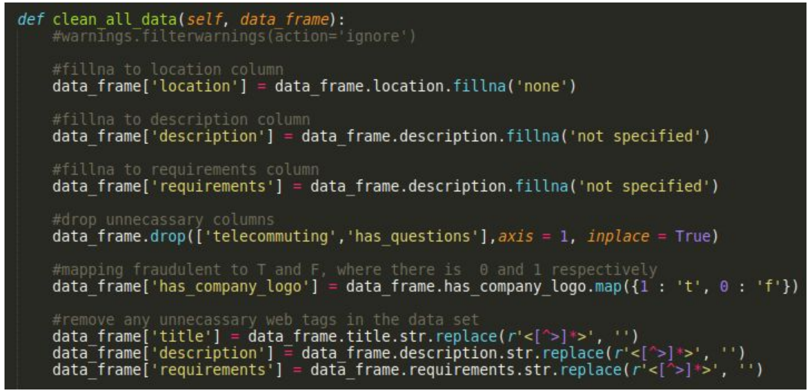


Preprocessing

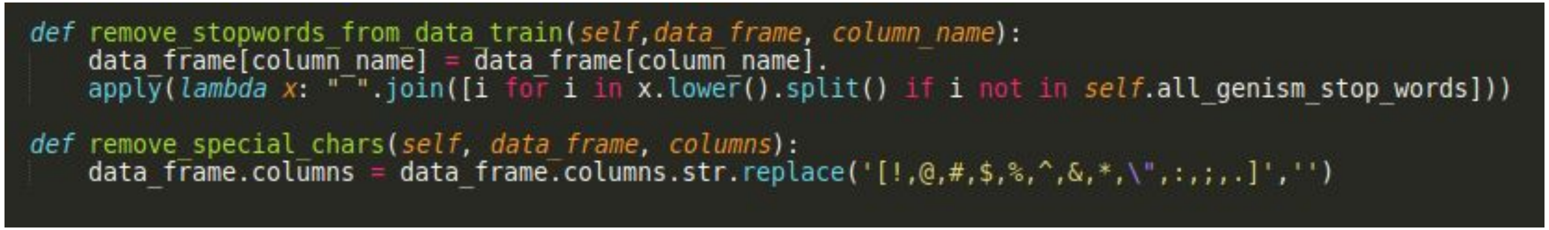
- FillNa on all text columns of the training data set
- Removing special characters, unnecessary web tags, characters that are not words and unnecessary white spaces.
- Removing STOPWORDS from the training dataset



```python
self.all_genism_stop_words = STOPWORDS

text_columns = list(data_frame.columns.values)

for columns in text_columns:
    self.remove_stopwords_from_data_train(data_frame,columns)

for columns in text_columns:
    self.remove_special_chars(data_frame, columns)
```



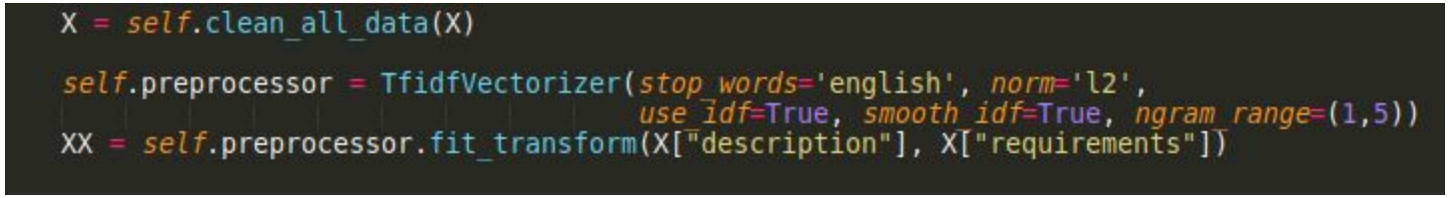
```python
def clean_all_data(self, data_frame):
    #warnings.filterwarnings(action='ignore')

    #fillna to location column
    data_frame['location'] = data_frame.location.fillna('none')

    #fillna to description column
    data_frame['description'] = data_frame.description.fillna('not specified')

    #fillna to requirements column
    data_frame['requirements'] = data_frame.description.fillna('not specified')

    #drop unnecassary columns
    data_frame.drop(['telecommuting','has_questions'],axis = 1, inplace = True)

    #mapping fraudulent to T and F, where there is  0 and 1 respectively
    data_frame['has_company_logo'] = data_frame.has_company_logo.map({1 : 't', 0 : 'f'})

    #remove any unnecassary web tags in the data set
    data_frame['title'] = data_frame.title.str.replace(r'<[^>]*>', '')
    data_frame['description'] = data_frame.description.str.replace(r'<[^>]*>', '')
    data_frame['requirements'] = data_frame.requirements.str.replace(r'<[^>]*>', '')
```



```python
def remove_stopwords_from_data_train(self,data_frame, column_name):
    data_frame[column_name] = data_frame[column_name].
    apply(lambda x: " ".join([i for i in x.lower().split() if i not in self.all_genism_stop_words]))

def remove_special_chars(self, data_frame, columns):
    data_frame.columns = data_frame.columns.str.replace('[!,@,#,$,%,^,&,*,\",:,;,.]','')
```

TFIDF vectorization

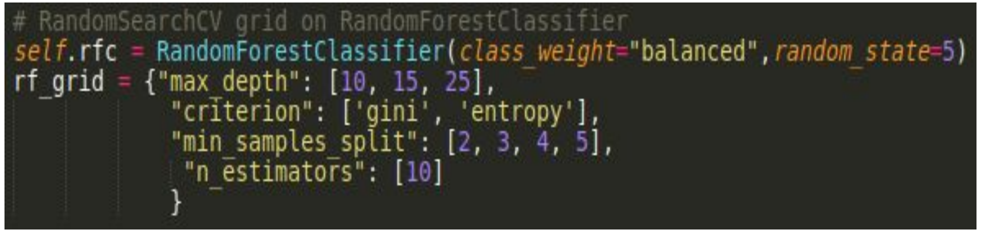
- TFIDF vectorization has been done on cleaned data
- Vectorization has been done on description and requirements columns as I decided it was the 2 columns that were giving most of the credible information



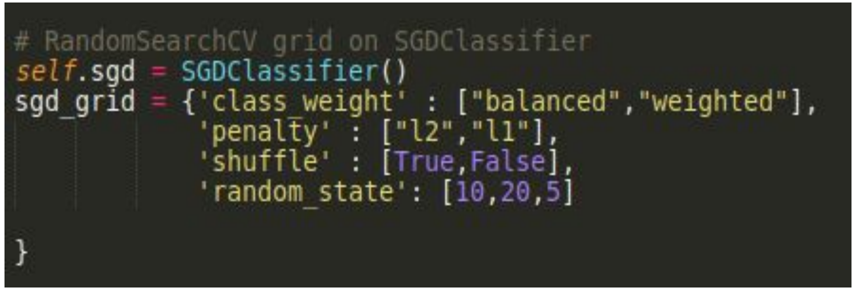
```
X = self.clean_all_data(X)

self.preprocessor = TfidfVectorizer(stop_words='english', norm='l2',
                                    use_idf=True, smooth_idf=True, ngram_range=(1,5))
XX = self.preprocessor.fit_transform(X["description"], X["requirements"])
```

Model Selection and Tuning

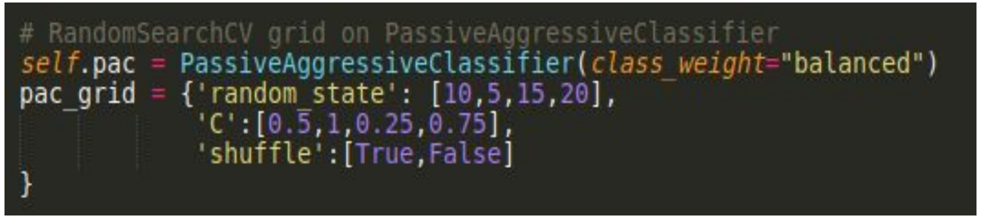
- Model Tuning is done on 3 models
  - SGDClassifier
  - RandomForestClassifier
  - PassiveAgressiveClassifier



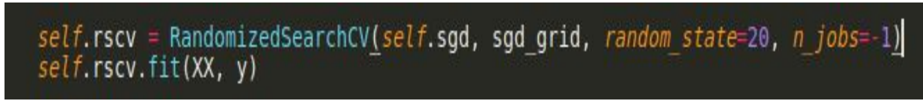
```python
# RandomSearchCV grid on RandomForestClassifier
self.rfc = RandomForestClassifier(class_weight="balanced",random_state=5)
rf_grid = {"max_depth": [10, 15, 25],
           "criterion": ['gini', 'entropy'],
           "min_samples_split": [2, 3, 4, 5],
            "n_estimators": [10]
           }
```



```python
# RandomSearchCV grid on SGDClassifier
self.sgd = SGDClassifier()
sgd_grid = {'class_weight' : ["balanced","weighted"],
            'penalty' : ["l2","l1"],
            'shuffle' : [True,False],
            'random_state': [10,20,5]

}
```

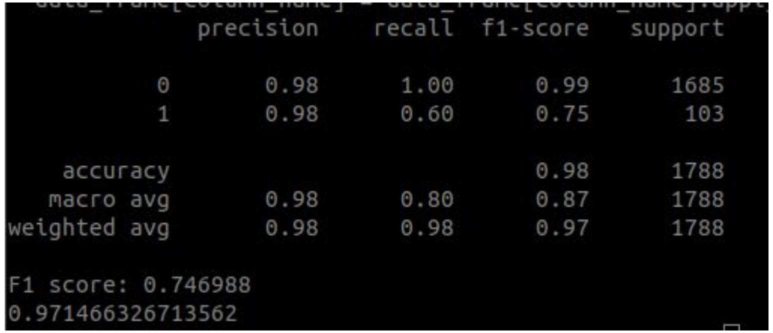


```python
# RandomSearchCV grid on PassiveAggressiveClassifier
self.pac = PassiveAggressiveClassifier(class_weight="balanced")
pac_grid = {'random_state': [10,5,15,20],
            'C':[0.5,1,0.25,0.75],
            'shuffle':[True,False]
}
```



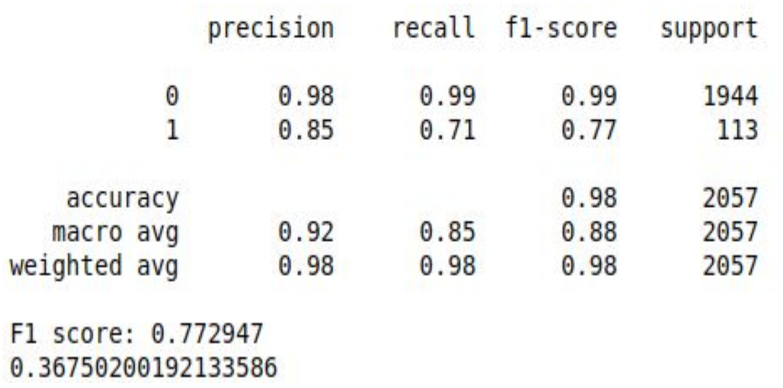
```python
self.rscv = RandomizedSearchCV(self.sgd, sgd_grid, random_state=20, n_jobs=-1)
self.rscv.fit(XX, y)
```

Performances of the models



```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1685
           1       0.98      0.60      0.75       103

    accuracy                           0.98      1788
   macro avg       0.98      0.80      0.87      1788
weighted avg       0.98      0.98      0.97      1788

F1 score: 0.746988
0.971466326713562
```

SGD classifier
classification report

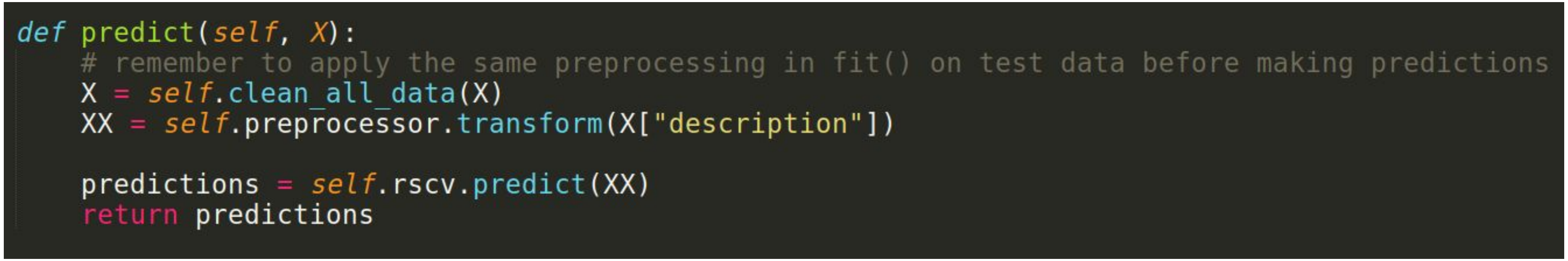


```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1944
           1       0.85      0.71      0.77       113

    accuracy                           0.98      2057
   macro avg       0.92      0.85      0.88      2057
weighted avg       0.98      0.98      0.98      2057

F1 score: 0.772947
0.36750200192133586
```

PAC classifier
Classification report

PREDICTIONS

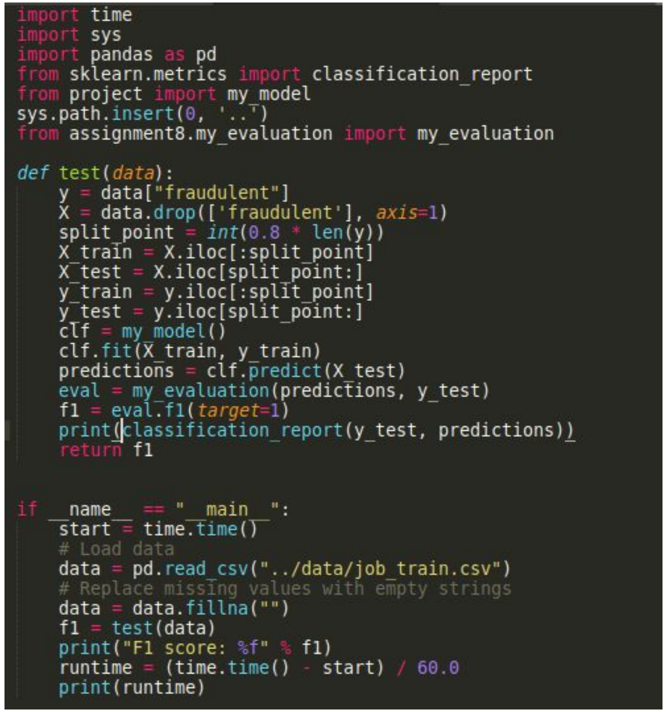
- Cleaning the testing data set with the same pre_processor
- TF IDF Transformation on the testing data set
- Calling predict method on the Random Search CV



```
def predict(self, X):
    # remember to apply the same preprocessing in fit() on test data before making predictions
    X = self.clean_all_data(X)
    XX = self.preprocessor.transform(X["description"])

    predictions = self.rscv.predict(XX)
    return predictions
```

Model Evaluation

- Using the Assignment 8 my_evaluation class to get the f1 score
- F1 score : 0.75 - 0.77



```python
import time
import sys
import pandas as pd
from sklearn.metrics import classification_report
from project import my_model
sys.path.insert(0, '..')
from assignment8.my_evaluation import my_evaluation

def test(data):
    y = data["fraudulent"]
    X = data.drop(['fraudulent'], axis=1)
    split_point = int(0.8 * len(y))
    X_train = X.iloc[:split_point]
    X_test = X.iloc[split_point:]
    y_train = y.iloc[:split_point]
    y_test = y.iloc[split_point:]
    clf = my_model()
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    eval = my_evaluation(predictions, y_test)
    f1 = eval.f1(target=1)
    print(classification_report(y_test, predictions))
    return f1


if __name__ == "__main__":
    start = time.time()
    # Load data
    data = pd.read_csv("../data/job_train.csv")
    # Replace missing values with empty strings
    data = data.fillna("")
    f1 = test(data)
    print("F1 score: %f" % f1)
    runtime = (time.time() - start) / 60.0
    print(runtime)
```

Any Questions ?

THANK YOU