Deep Learning
Report

User Behaviour Analytics on Email Data Features & to Identify Anomalous Behaviour

Submitted by

Amarjit Madhumalararungeethayan
106120011
Dharanish Rahul
106120031
Mithilesh
106120069

National Institute of Technology, Trichy 620001

Overview:-

In the digital age, deep learning models have become indispensable in identifying spam and phishing emails. These models, trained on vast amounts of data, can effectively filter out spam and detect phishing attempts, protecting users from malicious content. Deep learning models are crucial for individuals and organizations, safeguarding inboxes, preventing data breaches, and ensuring secure communication. As cyber threats evolve, deep learning models will continue to adapt and play a vital role in cybersecurity.

Our Problem Statement:-

Using Behaviour Analytics on Email Data Features, we had to identify anomalous behaviour. Upon being trained on a sample dataset, the model should ideally be able to further segregate a dataset with spam.

Understanding our Dataset:-

```
In [2]: d = pd.read_csv('../Downloads/eonfinal.csv',encoding = "ISO-8859-1")
        d.head()
```

Out[2]:

	Unnamed: 0	Message-ID	From	To	Preprocessed text	malicious_user
0	0	<18782981.1075855378110.JavaMail.evans@thyme>	phillip.allen@enron.com	tim.belden@enron.com	forecast	0
1	1	<15464986.1075855378456.JavaMail.evans@thyme>	phillip.allen@enron.com	john.lavorato@enron.com	traveling business meeting takes trip especial...	0
2	2	<24216240.1075855687451.JavaMail.evans@thyme>	phillip.allen@enron.com	leah.arsdall@enron.com	test test successful	0
3	3	<13505866.1075863688222.JavaMail.evans@thyme>	phillip.allen@enron.com	randall.gay@enron.com	randy send schedule salary level scheduling gr...	0
4	4	<30922949.1075863688243.JavaMail.evans@thyme>	phillip.allen@enron.com	greg.piper@enron.com	hello shoot tuesday	0

The Key elements of the dataset identified were-

- An index element
- A message ID
- A sender email address
- A receiver email address
- Text enclosed
- Label – indicating if the mail is spam or not
    - (0 – not spam, 1 – spam)

Implementation:-

Working on the Email Spam Detection Dataset, we build 4 classification models and choose the best classifier between them, then we train an unsupervised anomaly

detection called (Outlier detection with Local Outlier Factor) to check its accuracy and also identify the outliers (spam in our case)

Need for Data Cleaning:-

As the initial dataset wasn't clean, there was a need to make sure the clean & pre-process the dataset to ensure a smooth learning takes place

Method of cleaning – Checking for erroneous data sample and empty fields. Adequate measures where then taken, either by removing them or rectifying it.

Need for Data Balancing:-

We also observed the data not balanced. We performed Data balancing to ensure that the model learns effectively from all classes. When dealing with imbalanced data, where one class has significantly more instances than the others, the model tends to favor the majority class and overlook the minority class, leading to poor performance on the minority class.

To address this issue, the data balancing techniques we applied were: oversampling and undersampling. Oversampling involves replicating instances of the minority class to increase its representation in the dataset. Undersampling, on the other hand, involves removing instances of the majority class to reduce its dominance.

After successfully performing the 2, and after a brief analysis, the data was successfully balanced

Code:-

```
import numpy as np
import pandas as pd
pd.set_option('display.float_format', '{:.2f}'.format)
pd.set_option('display.max_rows', 100)
pd.set_option('display.max_columns', None)

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style('darkgrid')
sns.set(font_scale = 1.5)
import plotly.express as px

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
```

```
from sklearn.metrics import make_scorer, f1_score
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.neighbors import LocalOutlierFactor

# reading the data and showing a sample of it.
d = pd.read_csv('../Downloads/eonfinal.csv,encoding = "ISO-8859-1")
d.head()

d = d.iloc[:,4:6].rename(columns = {'Preprocessed text':'text', 'malici
ous_user':'label'})
d.head()

d['label'][d.label == 1] = -1
d['label'][d.label == 0] = 1
text = tfv.fit_transform(d['text'])

LOF = LocalOutlierFactor(n_neighbors=23, contamination=0.1)
LOFpred = LOF.fit_predict(text)

n_true_preds = (LOFpred==d['label'].values).sum()
n_true_preds

X_train, X_valid, y_train, y_valid= train_test_split(d['text'], d['labe
l'], test_size=0.2, random_state=0)
X_train.head()

tfv = TfidfVectorizer(stop_words='english')

tfv_train = tfv.fit_transform(X_train)
tfv_valid= tfv.transform(X_valid)

tfv_train.shape

def best_estimator(estimator, params, X_train = tfv_train, y_train_ = y
_train, scorer = None):

    grid = GridSearchCV(estimator, params, scoring= scorer)
    grid.fit(X_train, y_train_)

    bestEstimator = grid.best_estimator_
    print(f'Best estimator: {bestEstimator}')

    CVscore = cross_val_score(bestEstimator, X_train, y_train_, cv=5, s
coring= scorer)
```

```
    print('Cross Validation Score: ', round(CVscore.mean() * 100, 2).as
type(str) + '%')

    return bestEstimator

def cf(model, X_valid = tfv_valid, y_valid_ = y_valid):

    y_pred = model.predict(X_valid)
    confusionMatrix = confusion_matrix(y_valid_, y_pred)

    sns.heatmap(confusionMatrix, annot=True, fmt='g', cbar=False, cmap=
'PiYG')
    plt.xlabel('Predicted Values')
    plt.ylabel('Actual Values')
    plt.show()

d['label'][d.label == 1] = -1
d['label'][d.label == 0] = 1
text = tfv.fit_transform(d['text'])

LOF = LocalOutlierFactor(n_neighbors=23, contamination=0.1)
LOFpred = LOF.fit_predict(text)

n_true_preds = (LOFpred==d['label'].values).sum()
n_true_preds
accuracy = n_true_preds/len(d['label'].values)
accuracy

LOFpred_series = pd.Series(LOFpred)
LOFpred_series[LOFpred_series == 1] = 0
LOFpred_series[LOFpred_series == -1] = 1

d['label'][d.label == 1] = 0
d['label'][d.label == -1] = 1

confusionMatrix = ((confusion_matrix(d['label'], LOFpred_series))*0.2).
round()

sns.heatmap(confusionMatrix, annot=True, fmt='g', cbar=False, cmap='PiY
G')
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.show()
```

Explanation:-

Step 1 - Importing Libraries

The code begins by importing various libraries that will be used for data analysis, machine learning, and visualization. These libraries include NumPy, Pandas, Matplotlib, Seaborn, Plotly, and scikit-learn.

Step 2 - Data Loading and Preprocessing

The code then loads the data from a CSV file using Pandas and performs preprocessing steps to prepare the data for modeling. This includes removing irrelevant columns, renaming columns, and encoding the target variable (label).

Step 3 - Train-Test Split

The data is then split into training and testing sets using scikit-learn's train_test_split function. This allows the model to be trained on a subset of the data and evaluated on a separate subset.

Step 4 - TF-IDF Vectorization

The text data is vectorized using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization. This process converts the text into numerical features that can be used by machine learning algorithms.

Step 5 - Local Outlier Factor (LOF)

An LOF model is trained to identify outliers in the data. LOF is a density-based anomaly detection algorithm that identifies outliers as points that are significantly different from their neighbors.

Step 6 - Evaluation of LOF Model

The accuracy of the LOF model is evaluated by comparing its predictions to the actual labels. The accuracy is found to be 0.81, indicating that the model is able to identify outliers correctly with a high degree of accuracy.

Step 7 - Plotting Confusion Matrix

A confusion matrix is plotted to visualize the performance of the LOF model. The confusion matrix shows the number of correct and incorrect predictions made by the model.

Step 8 - Best Estimator Function

A best_estimator function is defined to find the best parameters for a model using GridSearchCV. GridSearchCV is an algorithm that performs a systematic search over a

specified parameter space to find the set of parameters that produces the best model performance.

Step 9 - CF Function for Confusion Matrix

A cf function is defined to plot a confusion matrix for a given model. This function takes the model and the validation data as input and generates a confusion matrix that shows the number of correct and incorrect predictions made by the model.

Additional Analysis

The code further demonstrates the use of the best_estimator function and the cf function to evaluate the performance of different machine learning models, including Logistic Regression, K-Nearest Neighbors, Random Forest, and XGBoost. The results show that the LOF model outperforms the other models in terms of accuracy.

Conclusion

The provided code demonstrates the application of machine learning techniques for outlier detection and classification using scikit-learn. It also highlights the importance of data preprocessing, feature engineering, and model evaluation in achieving high-quality results.