Spam Analysis on YouTube Comments

Presented by: Mustafa Furkan Kolanci

Date: 11/29/2021

Instructor: Angelica Spratley

github link: https://github.com/mfurkankolanci/CapstoneProject (https://github.com/mfurkankolanci/CapstoneProject)

Overview

In this project, I analyze user comments on popular music videos in YouTube. The dataset includes around 2,000 comments from Youtube music videos of 5 different artists. This dataset fits the business problem, as the goal is to provide YouTube with a model that can predict comments as spam or non-spam.

For data preparation, the first step is to clean the html tags present in the comments using regex. Then, regex is used again to detect the urls in the comments. After that, comments are tokenized to split each comment into words so that each word is a feature in the model. Stopwords are also removed as they provide little to no value to the spam analysis. Lemmatization is performed to group words with the same meaning together as one word. Then, TF-IDF is applied in order to assign each word in each comment a numeric value based on its importance across all comments.

I use pandas to perform data understanding and filtering, nltk to perform text preprocessing, and sklearn for TF-IDF.

For modeling, I use sklearn's MultinomialNB, LogisticRegression and RandomForestClassifier methods. I tune the models using GridSearchCV also provided by sklearn.

The best model has a test accuracy of 88%, which means that it correctly identifies the comments as spam and not spam 88% of the time. I use cross validation and the test set accuracy scores in order to validate the model's performance.

Business Understanding

As a data scientist, I was approached by a stakeholder, Youtube's maintance team. They have noticed that many popular music videos have several spam comments under them. Spam comments are undesired comments that are not related to the music video. They would like detect these comments in order to improve user experience. Therefore, they have asked me to generate a model that performs spam detection analysis in order to categorize comments as spam and not spam.

Data Understanding

The dataset is from the year 2013 and includes 1,956 Youtube comments. The comments are taken from music videos of PSY, Katy Perry, LMFAO, Eminem and Shakira. There are five columns, the first column includes the comment id, the second column is the author of comment, third column is the date of comment, fourth column is the comment text and fifth column is whether the comment is spam (1) or not (0).

Data Preparation

Let's start by importing modules necessary for analysis and reading in the data.

In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

Comments for each artist are in seperate csv files. Let's combine these files into a single dataframe.

In [2]:
```
df = pd.concat(map(pd.read_csv, ['Data/Youtube01-Psy.csv','Data/Youtube02-KatyPerry.csv',
                                 'Data/Youtube03-LMFAO.csv', 'Data/Youtube04-Eminem.csv',
                                 'Data/Youtube05-Shakira.csv']), ignore_index=True)
df.head()
```

Out[2]:

	COMMENT_ID	AUTHOR	DATE	CONTENT	CLASS
0	LZQPQhLyRh80UYxNuaDWhIGQYNQ96IuCg-AYWqNPjpU	Julius NM	2013-11-07T06:20:48	Huh, anyway check out this you[tube] channel: ...	1
1	LZQPQhLyRh_C2cTtd9MvFRJedxydaVW-2sNg5Diuo4A	adam riyati	2013-11-07T12:37:15	Hey guys check out my new channel and our firs...	1
2	LZQPQhLyRh9MSZYnf8djyk0gEF9BHDPYrrK-qCczIY8	Evgeny Murashkin	2013-11-08T17:34:21	just for test I have to say murdev.com	1
3	z13jhp0bxqncu512g22wvzkasxmvvzjaz04	ElNino Melendez	2013-11-09T08:28:43	me shaking my sexy ass on my channel enjoy ^_^	1
4	z13fwbwp1oujthgqj04chlngpvzmtt3r3dw	GsMega	2013-11-10T16:05:38	watch?v=vtaRGgvGtWQ Check this out .	1

In [3]:
```
# df['DATE'] = pd.to_datetime(df['DATE'])
# df["HOUR"] = df['DATE'].dt.hour
# df.head()
```

In [4]:
```
# temp = df[['CLASS','HOUR']]
# hourly_temp = temp.groupby(temp['HOUR']).sum()
# hourly_temp
```

In [5]:
```
# import matplotlib.pyplot as plt
# plt.bar(hourly_temp.index,hourly_temp.CLASS)
# plt.show()
```

The cell shown below indicates that the ratio of spam and non-spam comments are roughly equal. Therefore, there is no class imbalance

to resolve.

In [6]:
```
df['CLASS'].value_counts(normalize=True)
```

Out[6]:
```
1    0.513804
0    0.486196
Name: CLASS, dtype: float64
```

`COMMENT_ID` , `AUTHOR` and `DATE` columns provide no useful information in order to determine whether a comment is spam or not, so those columns are dropped.

In [7]:
```
df.drop(['COMMENT_ID','AUTHOR','DATE'],axis=1, inplace=True)
```

Text Preprocessing

The comments are webscraped and therefore some comments include html syntax and tags. The html needs to be cleaned so that only the comment itself remains.

In [8]:
```
import html
df['CONTENT'] = df['CONTENT'].apply(html.unescape)

import re
htmlregex = re.compile('<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});')

def cleanhtml(comment):
    html_filtered_comment = re.sub(htmlregex, '', comment)
    return html_filtered_comment

df['CONTENT'] = df['CONTENT'].apply(cleanhtml)

df['CONTENT'] = df['CONTENT'].str.replace('\ufeff','')
```

Next, some comments include url's and these url's should be detected. The cell below detects url's and replaces them with the keyword `url` .

In [9]:
```
urlregex = re.compile(r"""(?i)\b((?:https?:(?:/{1,3}|[a-z0-9%])|[a-z0-9.\-]+[.](?:com|net|org|edu|gov|mil|aero|a

def detecturl(comment):
    url_filtered_comment = re.sub(urlregex, 'url', comment)
    return url_filtered_comment
```

In [10]:
```
df['CONTENT'] = df['CONTENT'].apply(detecturl)
```

Finally, many comments include non-English characters, let's filter each comment such that only characters of the English language and numbers remain.

In [11]:
```
alphanumericregex = re.compile(r'[^A-Za-z0-9 ]+')
def onlyalphanumeric(comment):
    alphanumeric_filtered_comment = re.sub(alphanumericregex, '', comment)
    return alphanumeric_filtered_comment
```

In [12]:
```
df['CONTENT'] = df['CONTENT'].apply(onlyalphanumeric)
```

Next, let's perform some feature engineering on the tweets. In the following cell, let's initiate the tokenization, stopword removal and lemmatization of the tweets.

Tokenization will split each tweets into its respective words so that each word be analyzed by itself in the model.

Stopword removal is necessary as stopwords are words that provide little to no sentiment value for spam classification.

Lemmatization is needed as it allows to group words that are from the same root so that words that have the same meaning are considered the same. This way, the model is not hurt by needlessly increasing dimensionality.

In [13]:
```
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer

stopwords_list = stopwords.words('english')

lemmatizer = WordNetLemmatizer()

def lemmatize_and_tokenize(text):
    tokens = word_tokenize(text)
    return [lemmatizer.lemmatize(token) for token in tokens]
```

Next, let's perform a train-test split.

In [14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['CONTENT'], df['CLASS'], random_state=42) ## TRAIN TEST S
```

Now that the dataset is cleaned, lemmatized and tokenized, it can now be converted into a vector format. In order to do so, TF-IDF vectorization will be used. This method is based on the idea that rare words are more valuable for prediction. The method utilizes two metrics:

- TF (term frequency) refers to the ratio of number of times a word appear in the document to the total number of words in the document.
- IDF (Inverse Document Frequency) refers to the logged ratio of number of documents to the number of documents including the word.

In [21]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=10, ngram_range=(1,2), stop_words=stopwords_list, tokenizer=lemmatize_and_t

X_train_vectorized = tfidf.fit_transform(X_train)

X_train_df = pd.DataFrame.sparse.from_spmatrix(X_train_vectorized, columns=tfidf.get_feature_names())
```

In [22]:
```
# transforming the test data
X_test_vectorized = tfidf.transform(X_test)
X_test_df = pd.DataFrame.sparse.from_spmatrix(X_test_vectorized, columns=tfidf.get_feature_names())
```

MODELING

Baseline Model: Naive Bayes

Naive Bayes is selected as the baseline model as it is a fast, easy and simple classification algorithm.

In [ ]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.metrics import plot_confusion_matrix, accuracy_score
mnb = MultinomialNB()
mnb.fit(X_train_df, y_train)
y_test_pred = mnb.predict(X_test_df)
```

Let's use cross-validation to observe how the baseline model does with unseen training data.

In [ ]:
```
np.mean(cross_val_score(mnb,X_train_df, y_train,cv=5, scoring = 'accuracy'))
```

Let's look at the confusion matrix to see how the baseline model does on the test set.

In [ ]:
```
plot_confusion_matrix(mnb,X_test_df,y_test);
```

Finally, let's calculate the accuracy score of the baseline model on the test set.

In [ ]:
```
accuracy_score(y_test,y_test_pred)
```

With a mean cross validation accuracy score of 67% and a test set accuracy score of 65%, Naive Bayes does better than random chance, however, there is still room for improvement.

Accuracy will be used as the performance metric for the models. This is the case as identifying a spam comment as non-spam and a non-spam comment spam are equally faulty within the business problem. If a spam comment is recognized as non-spam, this disrupts the user experience of the video owner. On the other hand, if a non-spam comment is recognized as spam, this disrupts the user experience

of the comment owner. Since video owners and comment owners are equally valuable to the Youtube community, the model cannot prioritize minimizing one error over the other.Therefore, accuracy is used as it gives equal importance to each error while measuring the ability to correctly label each comment.

Model #2: Logistic Regression

Logistic Regression is selected as the second model as it is a simple classification algorithm which is computationally efficient. `GridSearchCV` will be utilized in order to try different hyperparameters and optimize the model.

In [ ]:
```
from sklearn.model_selection import GridSearchCV
logreg = LogisticRegression()
param_grid =  {
    'penalty': ['l2', 'l1', 'elasticnet'],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
    'C' : [1,100,10000]
}
gs_logreg = GridSearchCV(logreg, param_grid, cv=3, scoring='accuracy')
gs_logreg.fit(X_train_df, y_train)
gs_logreg.best_params_
```

In [ ]:
```
logreg1 = LogisticRegression(penalty = 'l2', solver = 'newton-cg', C = 100)
logreg1.fit(X_train_df, y_train)
```

Let's use cross-validation to observe how the model does with unseen training data.

In [ ]:
```
np.mean(cross_val_score(logreg1,X_train_df, y_train,cv=5, scoring = 'accuracy'))
```

Let's look at the confusion matrix to see how the model does on the test set.

In [ ]:
```
plot_confusion_matrix(logreg1,X_test_df,y_test);
```

Finally, let's calculate the accuracy score of the model on the test set.

In [ ]:
```
y_test_pred = logreg1.predict(X_test_df)
accuracy_score(y_test,y_test_pred)
```

With a mean cross validation accuracy score of 91% and a test set accuracy score of 88%, Logistic Regression does much better than the baseline model.

Model #3: Random Forest

Random Forest is selected as the third model as it is an ensemble model which means it is more immune to overfitting and usually has stronger performance compared to simpler models such as Logistic Regression and Naive Bayes. Once again `GridSearchCV` will be utilized in order to try different hyperparameters and optimize the model.

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier()
param_grid = {'n_estimators':[100,200,500],
              'criterion':['gini','entropy'],
              'max_depth':[50,100,200]
}
gs_forest = GridSearchCV(forest, param_grid, cv=3, scoring='accuracy')
gs_forest.fit(X_train_df, y_train)
gs_forest.best_params_
```

In [ ]:
```
forest1 = RandomForestClassifier(n_estimators=200 , criterion = 'gini',max_depth = 50)
forest1.fit(X_train_df, y_train)
```

Let's use cross-validation to observe how the tuned random forest does with unseen training data.

In [ ]:
```
np.mean(cross_val_score(forest1,X_train_df, y_train,cv=5, scoring = 'accuracy'))
```

Let's look at the confusion matrix to see how the tuned random forest did on the test set.

In [ ]:
```
plot_confusion_matrix(forest1,X_test_df,y_test);
```

Finally, let's calculate the accuracy score of the tuned random forest on the test set.

In [ ]:
```
y_test_pred = forest1.predict(X_test_df)
accuracy_score(y_test,y_test_pred)
```

With a mean cross validation accuracy score of 91% and a test set accuracy score of 88%, performance of Random Forest is around the same as Logistic Regression.

Conclusion

The goal of this project is to generate a method to perform spam analysis on popular music video comments on YouTube. To do so, three classification models are created, with the best model having a cross validation accuracy score of 91% and test accuracy score of 88%. The project provides the stakeholder with the model so that it can be applied on other YouTube comments.

I recommend that the model should only used to detect spam comments under music videos. The dataset only includes music video comments, therefore one should be cautious of applying this model on comments from non-music videos. Another recommendation is that this model should be used to flag the detected spam comments for further analysis and not remove them. Although the model has high spam detection accuracy, it's not high enough to remove a comment right away as removing a comment disrupts a user's ability to freely express their opinion on YouTube.

Future Research

The dataset used has around 2,000 comments. Popular music videos tend to have hundreds of thousands of comments each, therefore the more comments should be added to the dataset for better generalization and model performance. Another future work that can be done is identifying accounts that tend to make spam comments. The model can be used to track the frequency and amount of spam comments made by each account and therefore can flag accounts as spam or non-spam.

Finally, the model can be tested on non-music video comments as well. If the model performance doesn't suffer, the model can be generalized to videos with other genres as well and if it does suffer, alternative models can be generated for comments of different genres.