SMS Organizer

Live Project

Shesheer Rao Kokkirala

Objective

The main aim of this project is to build an efficient classifier model. We will build a model to detect whether a text message is a promotional message, otp or transactional.

Dataset

The dataset contains around 200 messages and three classes OTP and PROMOTIONAL and TRANSACTIONAL. We use ML algorithms to classify the classes present in the dataset.

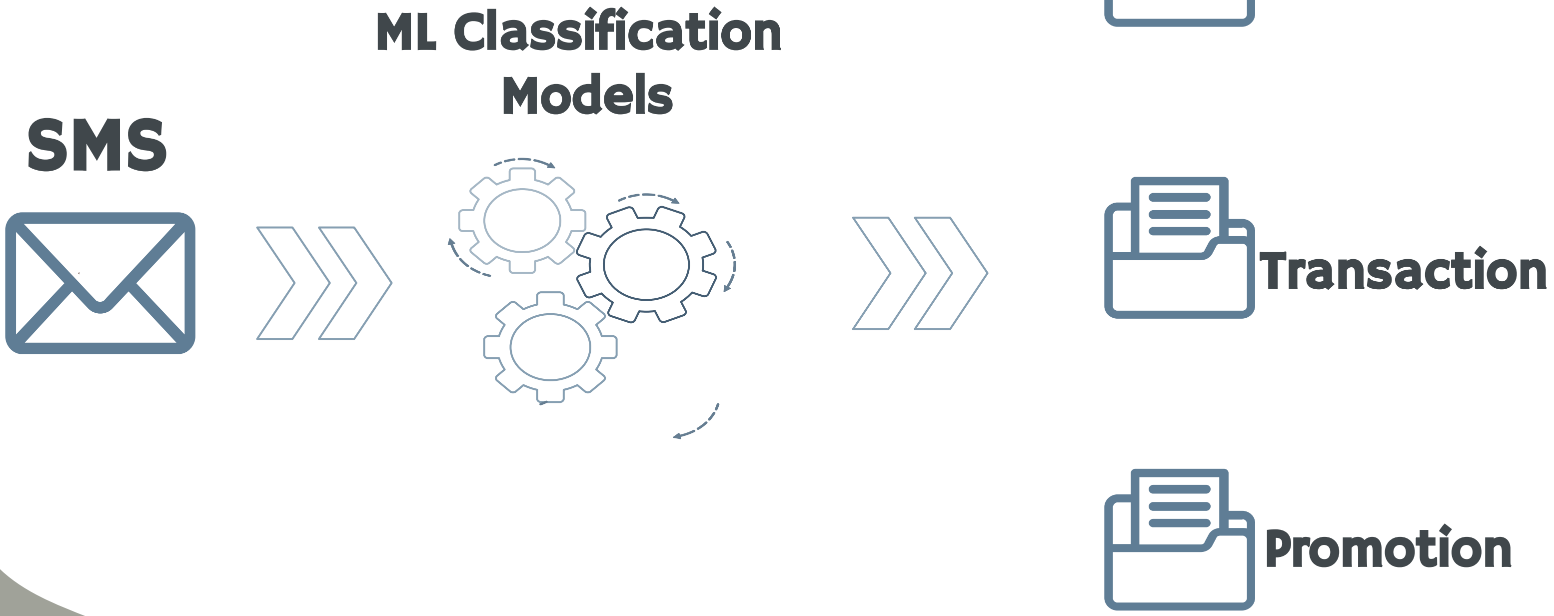
User verifica ion code 729416 for xyz company.
Otp for onlin purchase thru State Bank Debit Card 4523*****52 is 255325. Do not share this with anyone.
Greetings fr m Bajaj Finserv ! Please help us to complete your profile in order to bring some exciting exclusive offers. Click below: https://convl ot.hellotars.com/conv/BJD66I?campaign=9311444460&amount=1100000
Hello! Your a c no 49591359 has been debited by Rs 5449 on 2017-10-10. The a/c balance is Rs 600604.56. Info: NACH-DR- TP ACH ICICI BANK

OTP

Promotion

Transaction

SMS Classifier (OTP; Promotion; Transaction)



Process of Classification

01 Raw Text

02 Tokenization

03 Stop Words

04 Lemmatizatio
n

05 Cleaned Data

06 ML Model

07 Classification (70:30)

Definition(s)

Tokenization: *Tokenization* is breaking the raw text into small chunks. Tokenization breaks the raw text into words, sentences called tokens. These tokens help in understanding the context or developing the model for Classification.

Stopwords: *Stopwords* are the most common words in any natural language. For the purpose of analyzing text data and building NLP models, these stopwords might not add much value to the meaning of the document hence they are removed.

Lemmatization: *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*.

By doing the above processes we have a cleaned dataset that can be used for classification.

Word Cloud

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance.

Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

Word Cloud visualization can assist evaluators with exploratory textual analysis by identifying words that frequently appear in a set of interviews, documents, or other text.

It can also be used for communicating the most salient points or themes in the reporting stage.

In the next slides you will be able to see the word clouds from our data set with respect to each bucket.

Code For Word Clouds

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```
def visualize(label):
    words = ''
    for msg in df1[df1['Class'] == label]['Message']:
     msg = msg.lower()
     words += msg + ''

    wordcloud=WordCloud(width=1000, height=800).generate(words)
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()
```

Word Cloud - Promotion

```
In [25]: visualize('promotion')
```

Word Cloud - OTP

```
]: visualize('otp')
```

Word Cloud - Transaction

```
[26]: visualize('transaction')
```

Confusion Matrix

A confusion matrix is a technique for summarizing the performance of a classification algorithm.

It creates a *N X N* matrix, where N is the number of classes or categories that are to be predicted.

In our case since we have 3 classes it create a 3X3 matrix.

Confusion Matrix - Parameters

There are 4 terms you should keep in mind:

1. True Positives: It is the case where we predicted Yes and the real output was also yes.
2. True Negatives: It is the case where we predicted No and the real output was also No.
3. False Positives: It is the case where we predicted Yes but it was actually No.
4. False Negatives: It is the case where we predicted No but it was actually Yes.

Accuracy (all correct / all) = TP + TN / TP + TN + FP + FN

Step 1 : Importing the Libraries

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

In [1]:
```
import nltk
from nltk.corpus import stopwords

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

import pandas as pd
import string
import seaborn as sns
```

In [2]:
```
df = pd.read_csv("data3.csv")
df.head()
```

Displaying the dataset

Out[2]:

	Class	Message
0	transaction	Your SIP installment under HDFCMF folio 949181...
1	promotion	Priye Grahak! Aapke Rs.70 ke rchg ka benefit R...
2	otp	OTP for transaction of INR 23300.00 on your ca...
3	promotion	Big Bazaar NCR - Potato/Radish@10/kg Spinach@1...
4	promotion	Buy Shops PRELEASED with Most Trusted Brand of...

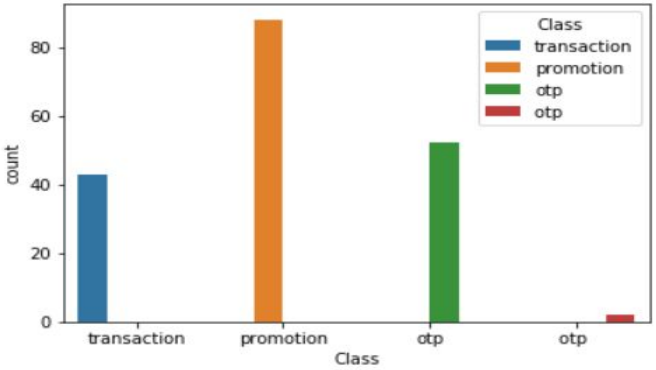
Step 2 : Dropping the Null Values

```
In [5]: df1 = df.dropna()

In [6]: df1.isnull().sum()

Out[6]: Class      0
        Message    0
```

We are checking, if there are any null values in the dataset since the presence of null values gives an error while training the dataset for classification.

Step 3 : Count Plot

```
In [10]: sns.countplot(x='Class',hue='Class', data=df1)
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x258adb93128>
```



Simple count plot, showing the different classes of messages in the dataset.

Step 4: Splitting the data into Training and Testing sets.

```
[11]: from sklearn.model_selection import train_test_split
```

```
[12]: X = df1['Message']
      y = df1['Class']
```

```
[13]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.30, random_state=42)
```

In this code, we split the entire dataset into testing and training datasets, where the size of the test set is 30% and training set is 70%.

Step 5: Countvectorizer

```
In [14]: from sklearn.feature_extraction.text import CountVectorizer

In [15]: count_vect=CountVectorizer()

In [16]: count_vect.fit(X_train)
         X_train_counts=count_vect.transform(X_train)

In [17]: X_train_counts

Out[17]: <129x1182 sparse matrix of type '<class 'numpy.int64'>'
                 with 3142 stored elements in Compressed Sparse Row format>
```

Countvectorizer: It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.

Step 6: Tfidf Vectorizer

```
In [18]: from sklearn.feature_extraction.text import TfidfTransformer

In [19]: tfidf_transformer=TfidfTransformer()
         X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
         X_train_tfidf.shape

Out[19]: (129, 1182)

In [20]: from sklearn.feature_extraction.text import TfidfVectorizer

In [21]: vectorizer=TfidfVectorizer()

In [22]: X_train_tfidf=vectorizer.fit_transform(X_train)
```

If you need the term frequency (term count) vectors for different tasks, use Tfidftransformer.
The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.If you need to compute tf-idf scores on documents within your “training” dataset, use Tfidfvectorizer.
If you need to compute tf-idf scores on documents outside your “training” dataset, use either one, both will work.

Step 7 : Using KNN Classifier

```
In [28]: from sklearn.neighbors import KNeighborsClassifier
         clf=KNeighborsClassifier()
         clf.fit(X_train_tfidf,y_train)

Out[28]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                              weights='uniform')
```

Importing the KNN Model

```
In [29]: from sklearn.pipeline import Pipeline

In [30]: text_clf=Pipeline([('tfidf', TfidfVectorizer()), ('clf',KNeighborsClassifier())])

In [31]: text_clf.fit(X_train, y_train)

Out[31]: Pipeline(memory=None,
                steps=[('tfidf', TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                   dtype=<class 'numpy.float64'>, encoding='utf-8', input='content',
                   lowercase=True, max_df=1.0, max_features=None, min_df=1,
                   ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,...ki',
                      metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                      weights='uniform'))])
```

Training the data using KNN Model

```
In [32]: predictions=text_clf.predict(X_test)
```

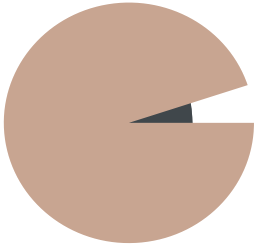
Testing the data using KNN Model

```
In [33]: print("CONFUSION MATRIX")
         from sklearn.metrics import confusion_matrix, classification_report
         print(confusion_matrix(y_test, predictions))

CONFUSION MATRIX
[[20  0  0]
 [ 1 23  2]
 [ 1  0  9]]
```

Printing the confusion matrix for calculating the accuracy

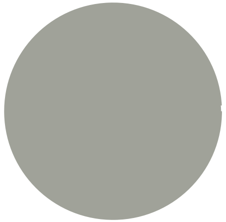
KNN

94.74%



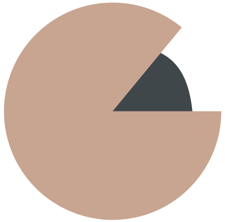
Promotion

100%



OTP

86.67%



Transaction

93.75%

Overall

Step 8 : Random Forest

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
        clf1=RandomForestClassifier()
        clf1.fit(X_train_tfidf,y_train)


        from sklearn.pipeline import Pipeline
        text_clf1=Pipeline([('tfidf', TfidfVectorizer()), ('clf', RandomForestClassifier())])
        text_clf1.fit(X_train, y_train)

        predictions1=text_clf1.predict(X_test)

        print("CONFUSION MATRIX")
        from sklearn.metrics import confusion_matrix, classification_report
        print(confusion_matrix(y_test, predictions1))
```

Importing the Random Forest Model

Training the data using Random Forest

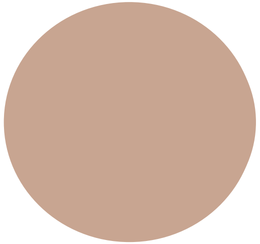
Testing the data using Random Forest

```
print(confusion_m

CONFUSION MATRIX
[[19  1  0]
 [ 0 26  0]
 [ 0  3  7]]
```

Printing the confusion matrix for calculating the accuracy

Random Forest

100%

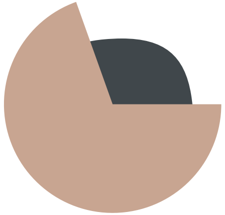


Promotion

92.31%

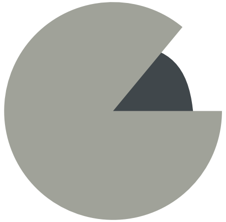
OTP

68.75%



Transaction

87.5%



Overall

Step 9: Decision Tree

```
In [ ]: from sklearn.tree import DecisionTreeClassifier
        clf2=DecisionTreeClassifier()
        clf2.fit(X_train_tfidf,y_train)

        from sklearn.pipeline import Pipeline
        text_clf2=Pipeline([('tfidf', TfidfVectorizer()), ('clf',DecisionTreeClassifier())])
        text_clf2.fit(X_train, y_train)

        print("CONFUSION MATRIX")
        predictions2=text_clf2.predict(X_test)
        print(confusion_matrix(y_test, predictions))
```

```
CONFUSION MATRIX
[[20  0  0]
 [ 1 23  2]
 [ 1  0  9]]
```

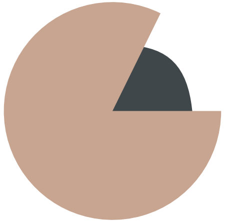
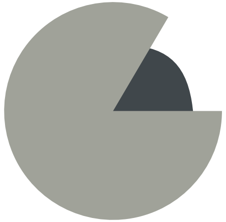
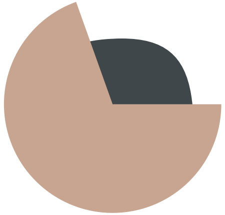
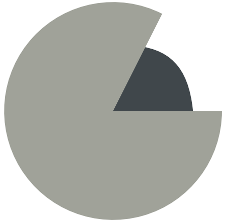
Importing the Decision Tree Model

Training the data using Decision Tree

Testing the data using Decision Tree

Printing the confusion matrix for calculating the accuracy

Decision Tree

84.21%	84.62%	68.75%	79.17%
			
Promotion	OTP	Transaction	Overall

Step 10: Linear SVC

```
In [ ]: from sklearn.svm import LinearSVC
        clf3=LinearSVC()
        clf3.fit(X_train_tfidf,y_train)

        text_clf3=Pipeline([('tfidf', TfidfVectorizer()), ('clf',LinearSVC())])
        text_clf3.fit(X_train, y_train)

        predictions3=text_clf3.predict(X_test)

        print("CONFUSION MATRIX")
        from sklearn.metrics import confusion_matrix, classification_report
        print(confusion_matrix(y_test, predictions3))
```

Importing the Linear SVC Model

Training the data using SVC

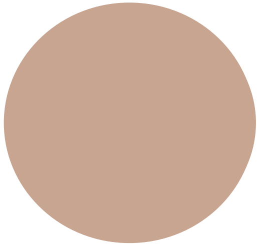
Testing the data using SVC

Printing the confusion matrix for calculating the accuracy

```
CONFUSION MATRIX
[[20  0  0]
 [ 0 26  0]
 [ 0  0 10]]
```

Linear SVC

100%

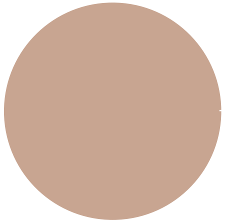


Promotion

92.31%

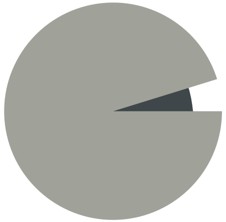
OTP

100%



Transaction

95.83%



Overall

Step II: Naive Bayes

```
In [ ]: from sklearn.naive_bayes import MultinomialNB
        clf4=MultinomialNB()
        clf4.fit(X_train_tfidf,y_train)

        text_clf4=Pipeline([('tfidf', TfidfVectorizer()), ('clf',MultinomialNB())])
        text_clf4.fit(X_train, y_train)


        predictions4=text_clf4.predict(X_test)

        print("CONFUSION MATRIX")
        from sklearn.metrics import confusion_matrix, classification_report
        print(confusion_matrix(y_test, predictions4))
```

Importing the Naive Bayes Model

Training the data using Naive Bayes

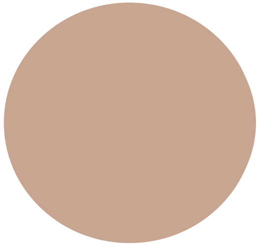
Testing the data using Naive Bayes

Printing the confusion matrix for calculating the accuracy

```
CONFUSION MATRIX
[[19  1  0]
 [ 0 26  0]
 [ 0  1  9]]
```

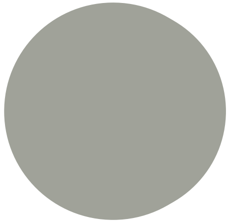
Naive Bayes

100%



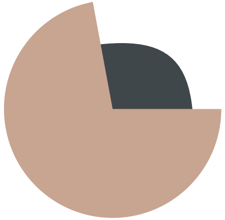
Promotion

100%



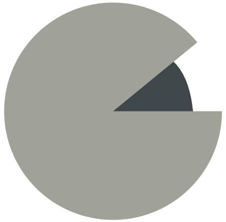
OTP

70.59%



Transaction

89.58%



Overall

Step 12: Prediction

```
In [37]: text_clf.predict(["You have initiated a txn of INR 34.00 at PayTm Mobi on ICICI Bank Card no.5504. OTP is 752565. DONT SHARE WITH

Out[37]: array(['otp'], dtype=object)

In [38]: text_clf.predict(["Dear Customer, Purchase of INR 20.00 has been made on Debit Card linked to Acct XX1041 on 14-Sep-17. Info: VPS

Out[38]: array(['transaction'], dtype=object)
```

Here, we take random message and test it for classification, which predicts the correct class.

CONCLUSION

To conclude, we would like to use the SVM model, because :

SVM Classifier in comparison to other classifiers have better computational complexity and even if the number of positive and negative examples are not same, SVM can be used as it has the ability to normalize the data or to project into the space of the decision boundary separating the two classes.The Execution time comes out to be very little in comparison to other algorithms.

In contrast to other models, in decision tree, for more data, the number of decision nodes increases, and for the random forest number of trees increases.

For Naive Bayes, as it is a probabilistic model, it won't be efficient for large amount of data, and for KNN – as it searches for its neighbours everytime, its time complexity is more.