Text Classification: Spam or Ham
NLP Project (UML602)

Submitted By:

Kashish Bansal – 101603155

Kashish Mittal – 101603156

BE Third Year, COE

Submitted to:

Ms. Diksha Hooda

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Computer Science and Engineering Department

TIET, Patiala

April 2019

Index

1. Introduction

Mobile spam in an increasing threat that may be addressed using filtering systems like those employed against email spam. We believe that email filtering techniques require some adaptation to reach good levels of performance on SMS spam, especially regarding message representation. In order to test this assumption, we have performed experiments on SMS filtering using email spam filters on mobile spam messages using a suitable feature representation, with results supporting our hypothesis.

A spam filter is an service feature designed to block spam from a user's inbox. Because a large amount of global messages are spam, effective spam filters are critical to maintaining clean and spam-free in-boxes. By keeping pesky spam messages away, spam filters increase user efficiency by sparing tedious manual sifting of legitimate messages and spam email deletion.

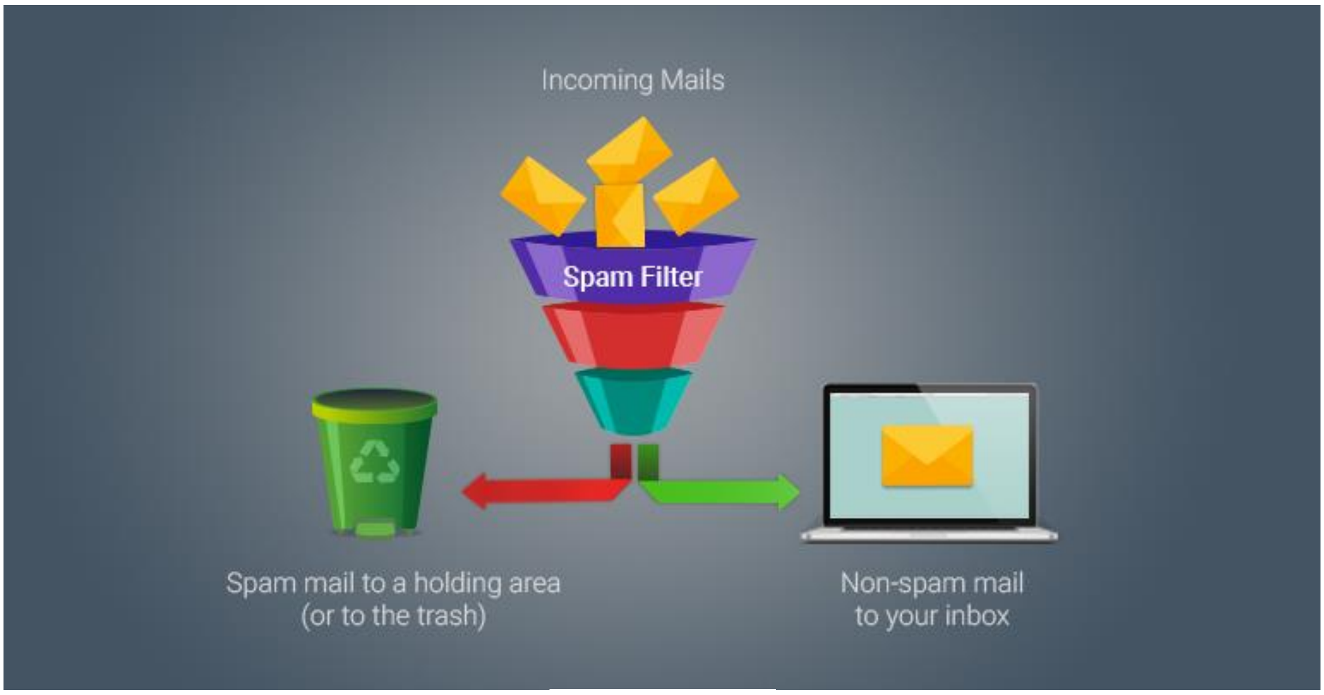

Incoming Mails
Spam Filter
Spam mail to a holding area (or to the trash)
Non-spam mail to your inbox

Fig:1 Spam Filter

In this, we will expand on this foundation and explore different ways to improve our text classification results. We will cover and use:

- Regular Expressions
- Feature Engineering
- Multiple scikit-learn Classifiers
- Ensemble Methods

1.1 Regular Expression:

Regular expressions are used to specify rules for matching strings of text. In other words, you can use regular expressions to match text in incoming messages. The settings in which you use the regular expressions will tell filter what to do with messages that have matching text.

1.2 Feature Engineering:

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process.

1.3 Scikit-learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.The library is built upon the SciPy (Scientific Python) that must be installed before you can use Scikit-learn. This stack that includes:

- **NumPy**: Base n-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D/3D plotting
- **IPython**: Enhanced interactive console
- **Sympy**: Symbolic mathematics
- **Pandas**: Data structures and analysis

Extensions or modules for SciPy care conventionally named SciKits. As such, the module provides learning algorithms and is named Scikit-learn.

1.4 Ensemble Method:

Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model. That is why ensemble methods placed first in many prestigious machine learning competitions, such as the Netflix Competition, KDD 2009, and Kaggle.

2. Steps of working

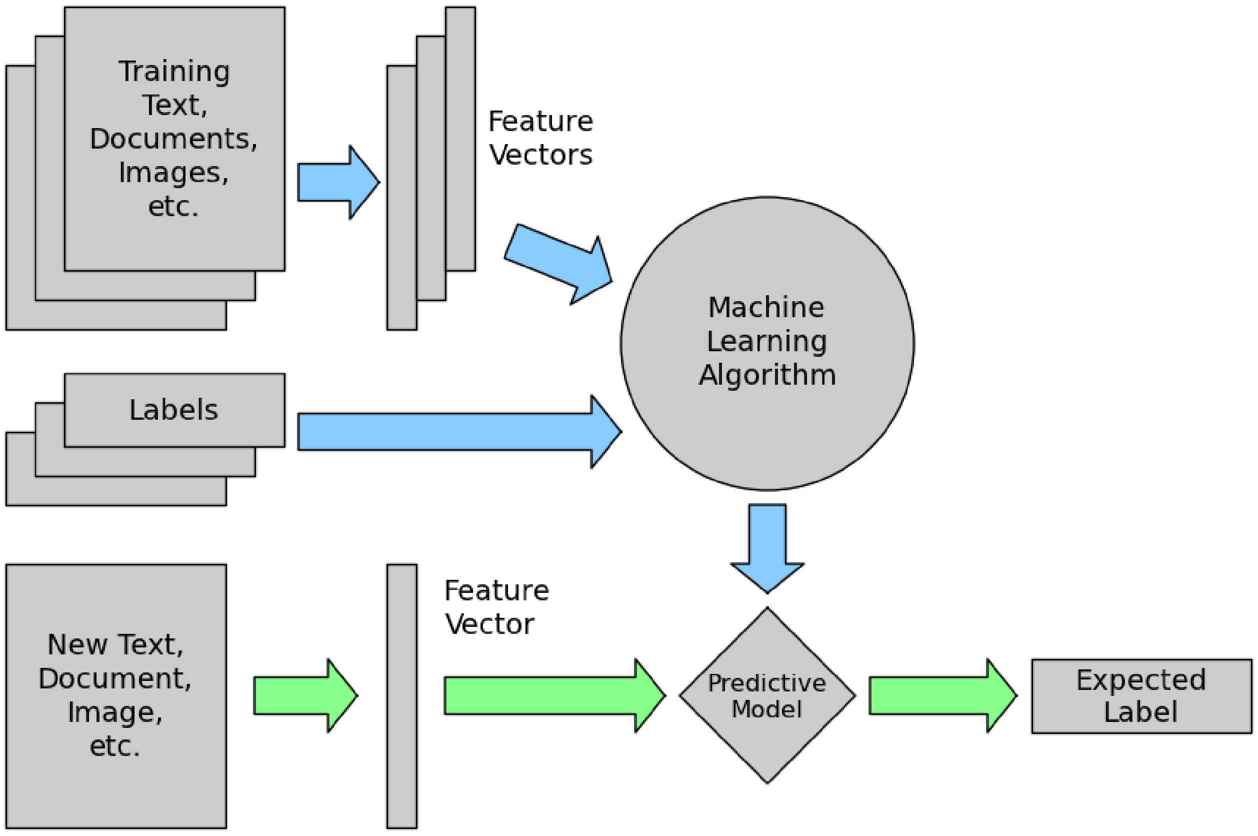

Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Fig 2: Basic Model

2.1 Import Necessary Libraries

To ensure the necessary libraries are installed correctly and up-to-date, print the version numbers for each library. This will also improve the reproducibility of our project.

Python: 2.7.13 |Continuum Analytics, Inc.| (default, May 11 2017, 13:17:26) [MSC v.1500 64 bit (AMD64)]
NLTK: 3.2.5
Scikit-learn: 0.19.1
Pandas: 0.21.0
Numpy: 1.14.1

2.2 Load the Dataset

Now that we have ensured that our libraries are installed correctly, let's load the data set as a Pandas DataFrame. Furthermore, let's extract some useful information such as the column information and class distributions.

The data set we will be using comes from the UCI Machine Learning Repository. It contains over 5000 SMS labeled messages that have been collected for mobile phone spam research. It can be downloaded from the following URL:

https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

2.3 Preprocess the Data

Preprocessing the data is an essential step in natural language process. In the following cells, we will convert our class labels to binary values using the LabelEncoder from sklearn, replace email addresses, URLs, phone numbers, and other symbols by using regular expressions, remove stop words, and extract word stems.

2.3.1 Regular Expressions

Some common regular expression metacharacters - copied from wikipedia

^ Matches the starting position within the string. In line-based tools, it matches the starting position of any line.

. Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor-, character-encoding-, and platform-specific, but it is safe to assume that the line feed character is included). Within POSIX bracket expressions, the dot character matches a literal dot. For example, a.c matches "abc", etc., but [a.c] matches only "a", ".", or "c".

[ ] A bracket expression. Matches a single character that is contained within the brackets. For example, [abc] matches "a", "b", or "c". [a-z] specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: [abcx-z] matches "a", "b", "c", "x", "y", or "z", as does [a-cx-z]. The - character is treated as a literal character if it is the last or the first (after the ^, if present) character within the brackets: [abc-], [-abc]. Note that backslash escapes are not allowed. The] character can be included in a bracket expression if it is the first (after the ^) character: []abc].

[^] Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than "a", "b", or "c". [^a-z] matches any single character that is not a lowercase letter from "a" to "z". Likewise, literal characters and ranges can be mixed.

$ Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.

( ) Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, \n). A marked subexpression is also called a block or capturing group. BRE mode requires ( ).

\n Matches what the nth marked subexpression matched, where n is a digit from 1 to 9. This construct is vaguely defined in the POSIX.2 standard. Some tools allow referencing more than nine capturing groups.

* Matches the preceding element zero or more times. For example, ab*c matches "ac", "abc", "abbbc", etc.* *[xyz]* matches "", "x", "y", "z", "zx", "zyx", "xyzzy", and so on. (ab)* matches "", "ab", "abab", "ababab", and so on.

{m,n} Matches the preceding element at least m and not more than n times. For example, a{3,5} matches only "aaa", "aaaa", and "aaaaa". This is not found in a few older instances of regexes. BRE mode requires {m,n}.

2.4 Generating Features

Feature engineering is the process of using domain knowledge of the data to create features for machine learning algorithms. In this project, the words in each text message will be our features. For this purpose, it will be necessary to tokenize each word. We will use the 1500 most common words as features.

2.5 Scikit-Learn Classifiers with NLTK

Now that we have our dataset, we can start building algorithms! Let's start with a simple linear support vector classifier, then expand to other algorithms. We'll need to import each algorithm we plan on using from sklearn. We also need to import some performance metrics, such as accuracy_score and classification_report.

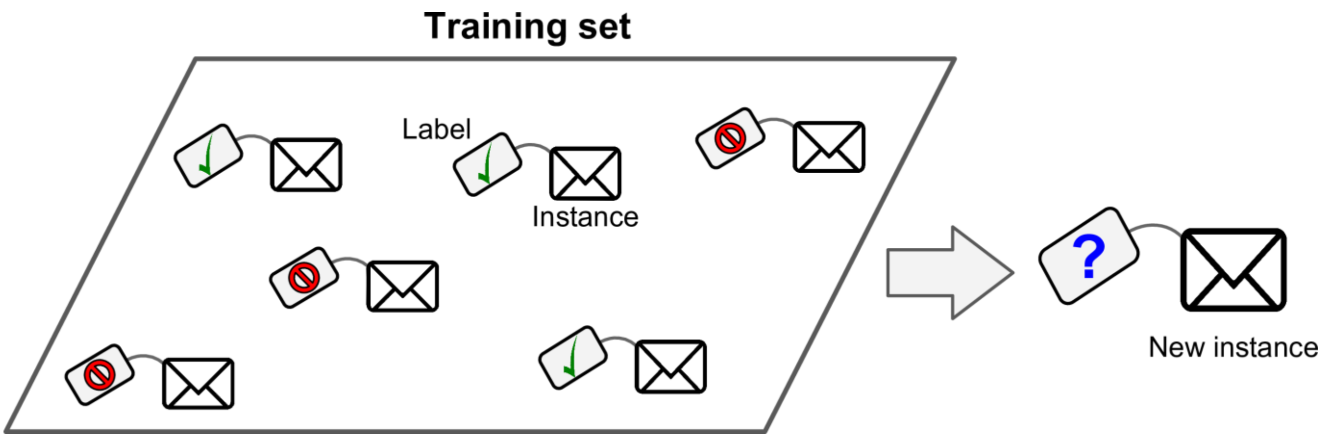


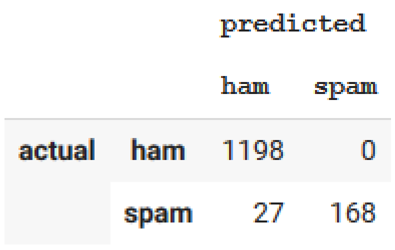
Fig 3: Training

3. Results

The dataset is trained with various models and the following results were obtained:

- K Nearest Neighbors Accuracy: 93.1083991385
- Decision Tree Accuracy: 97.4874371859
- SGD Classifier Accuracy: 97.9181622398
- Naive Bayes Accuracy: 97.5592246949
- SVM Linear Accuracy: 98.1335247667

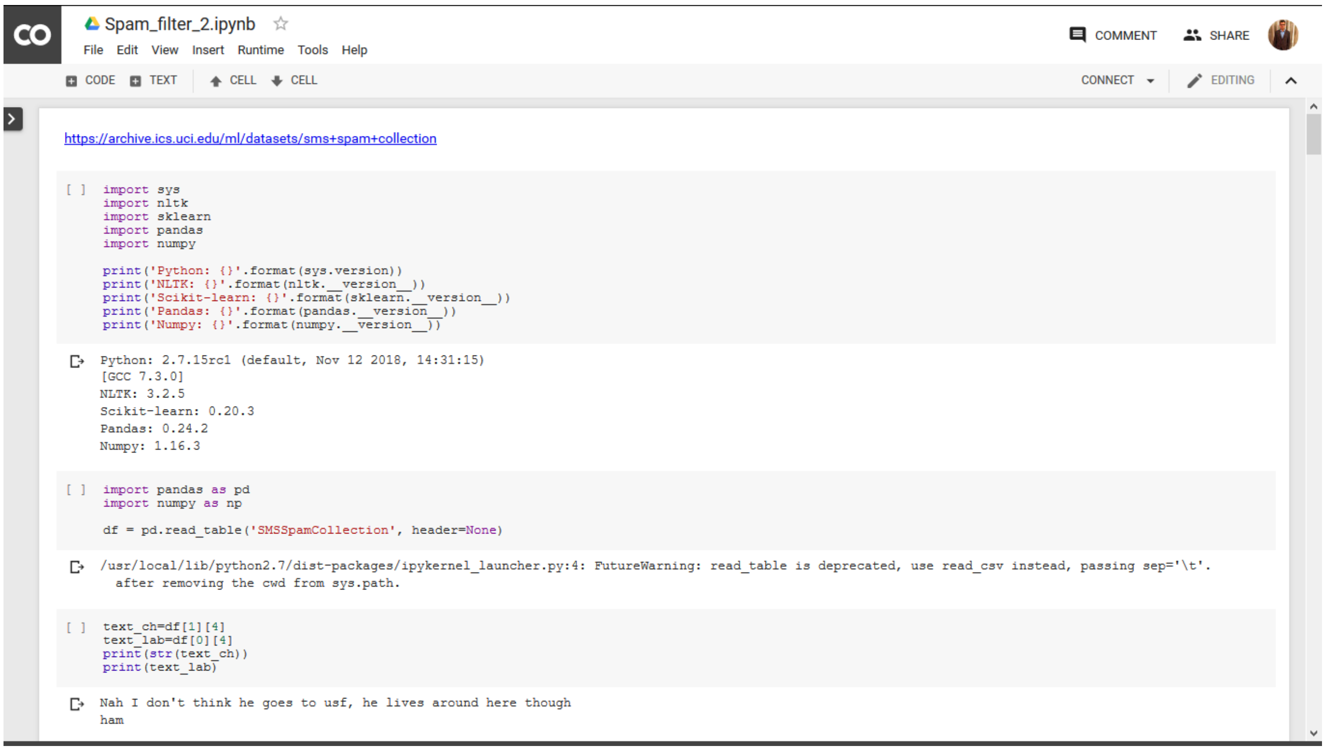
Then the model was trained with ensemble classifier in which voting of all the above models were done and following result was obtained:
Voting Classifier: Accuracy: 98.1335247667



		predicted	
		ham	spam
actual	ham	1198	0
	spam	27	168

Fig: Confusion Matrix

4. Code



https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

```
import sys
import nltk
import sklearn
import pandas
import numpy

print('Python: {}'.format(sys.version))
print('NLTK: {}'.format(nltk.__version__))
print('Scikit-learn: {}'.format(sklearn.__version__))
print('Pandas: {}'.format(pandas.__version__))
print('Numpy: {}'.format(numpy.__version__))
```

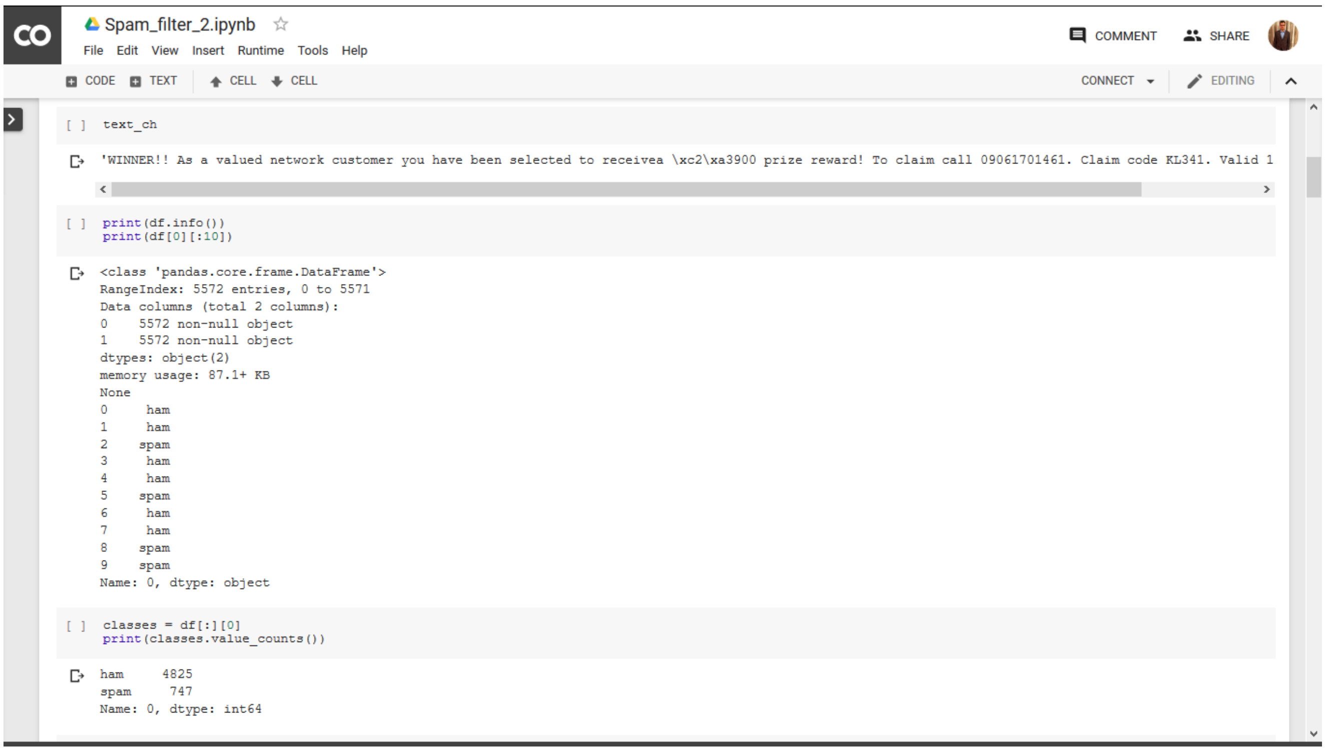
```
Python: 2.7.15rc1 (default, Nov 12 2018, 14:31:15)
[GCC 7.3.0]
NLTK: 3.2.5
Scikit-learn: 0.20.3
Pandas: 0.24.2
Numpy: 1.16.3
```

```
import pandas as pd
import numpy as np

df = pd.read_table('SMSSpamCollection', header=None)
```

```
/usr/local/lib/python2.7/dist-packages/ipykernel_launcher.py:4: FutureWarning: read_table is deprecated, use read_csv instead, passing sep='\t'.
  after removing the cwd from sys.path.
```

```
text_ch=df[1][4]
text_lab=df[0][4]
print(str(text_ch))
print(text_lab)
```

```
Nah I don't think he goes to usf, he lives around here though
ham
```



```
text_ch
```

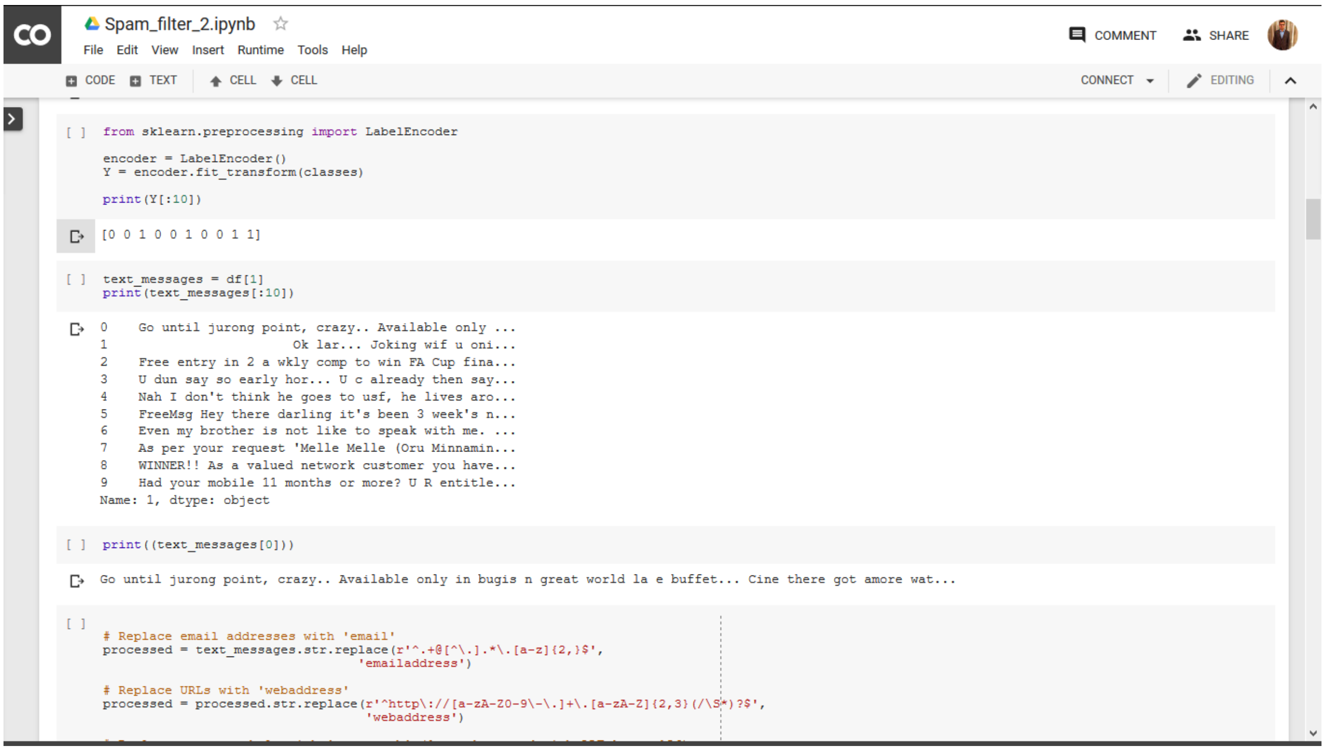
```
'WINNER!! As a valued network customer you have been selected to receivea \xc2\xa3900 prize reward! To claim call 09061701461. Claim code KL341. Valid 1
```

```
print(df.info())
print(df[0][:10])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
0    5572 non-null object
1    5572 non-null object
dtypes: object(2)
memory usage: 87.1+ KB
None
0     ham
1     ham
2    spam
3     ham
4     ham
5    spam
6     ham
7     ham
8    spam
9    spam
Name: 0, dtype: object
```

```
classes = df[:][0]
print(classes.value_counts())
```

```
ham     4825
spam     747
Name: 0, dtype: int64
```



```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
Y = encoder.fit_transform(classes)

print(Y[:10])
```

```
[0 0 1 0 0 1 0 0 1 1]
```

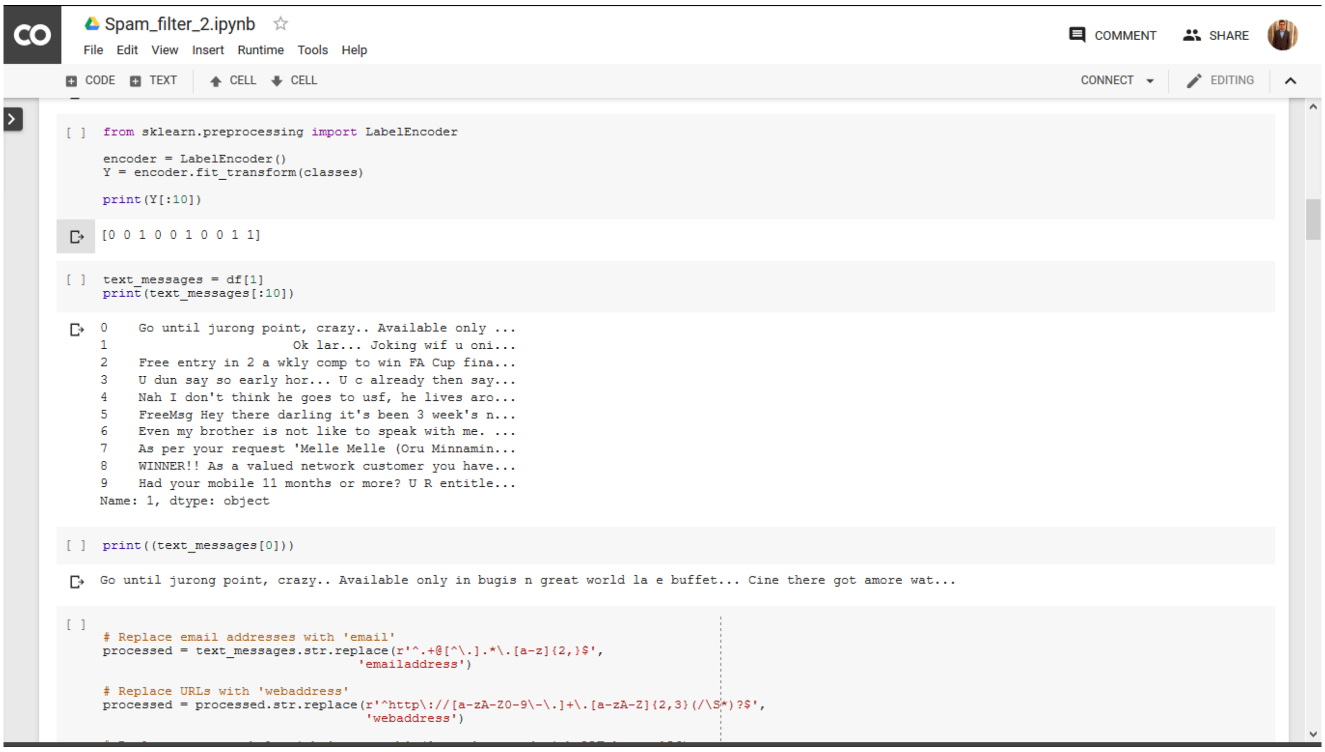
```
text_messages = df[1]
print(text_messages[:10])
```

```
0    Go until jurong point, crazy.. Available only ...
1                        Ok lar... Joking wif u oni...
2    Free entry in 2 a wkly comp to win FA Cup fina...
3    U dun say so early hor... U c already then say...
4    Nah I don't think he goes to usf, he lives aro...
5    FreeMsg Hey there darling it's been 3 week's n...
6    Even my brother is not like to speak with me. ...
7    As per your request 'Melle Melle (Oru Minnamin...
8    WINNER!! As a valued network customer you have...
9    Had your mobile 11 months or more? U R entitle...
Name: 1, dtype: object
```

```
print((text_messages[0]))
```

```
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
```

```
# Replace email addresses with 'email'
processed = text_messages.str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                  'emailaddress')

# Replace URLs with 'webaddress'
processed = processed.str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                  'webaddress')
```



```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
Y = encoder.fit_transform(classes)

print(Y[:10])
```

```
[0 0 1 0 0 1 0 0 1 1]
```

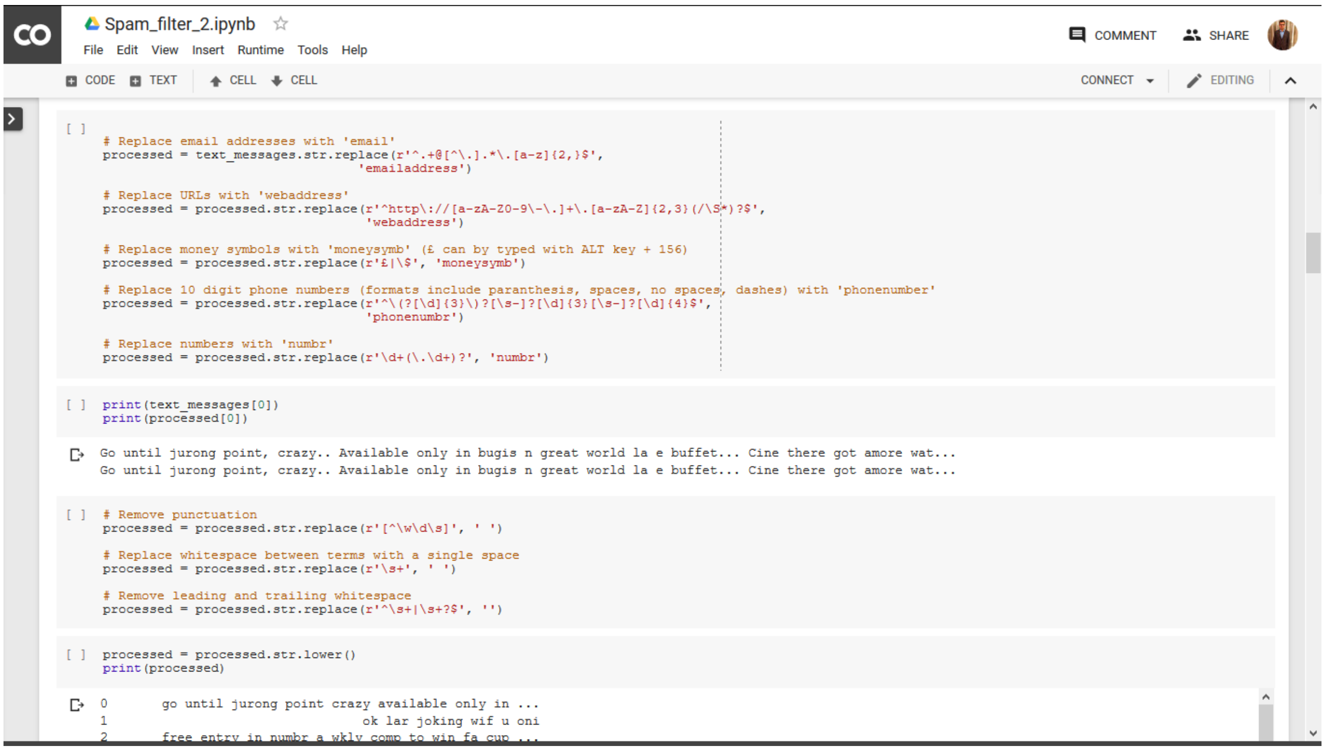
```
text_messages = df[1]
print(text_messages[:10])
```

```
0    Go until jurong point, crazy.. Available only ...
1                        Ok lar... Joking wif u oni...
2    Free entry in 2 a wkly comp to win FA Cup fina...
3    U dun say so early hor... U c already then say...
4    Nah I don't think he goes to usf, he lives aro...
5    FreeMsg Hey there darling it's been 3 week's n...
6    Even my brother is not like to speak with me. ...
7    As per your request 'Melle Melle (Oru Minnamin...
8    WINNER!! As a valued network customer you have...
9    Had your mobile 11 months or more? U R entitle...
Name: 1, dtype: object
```

```
print((text_messages[0]))
```

```
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
```

```
# Replace email addresses with 'email'
processed = text_messages.str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                  'emailaddress')

# Replace URLs with 'webaddress'
processed = processed.str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                  'webaddress')
```



```
[ ]
     # Replace email addresses with 'email'
     processed = text_messages.str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                          'emailaddress')

     # Replace URLs with 'webaddress'
     processed = processed.str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                       'webaddress')

     # Replace money symbols with 'moneysymb' (£ can by typed with ALT key + 156)
     processed = processed.str.replace(r'£|\$', 'moneysymb')

     # Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'phonenumber'
     processed = processed.str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$',
                                       'phonenumbr')

     # Replace numbers with 'numbr'
     processed = processed.str.replace(r'\d+(\.\d+)?', 'numbr')
```

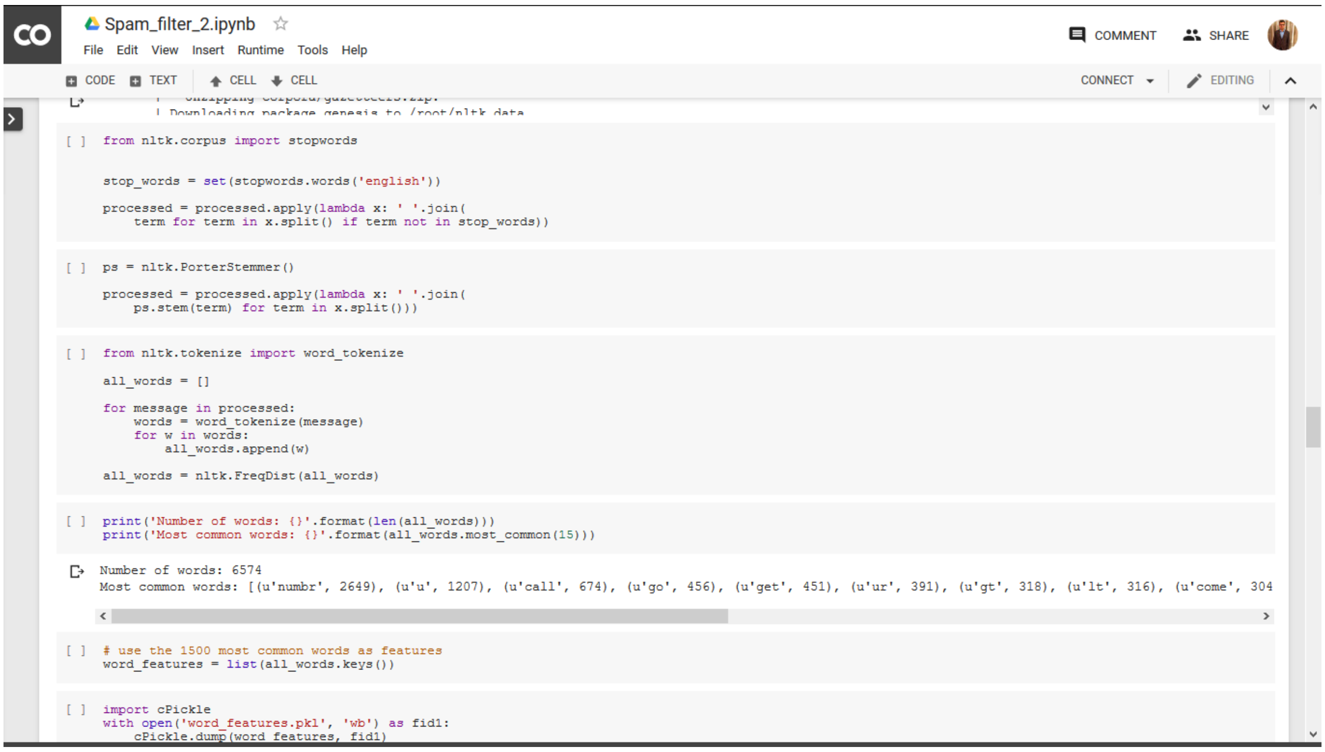
```
[ ]  print(text_messages[0])
     print(processed[0])
```

```
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
```

```
[ ]  # Remove punctuation
     processed = processed.str.replace(r'[^\w\d\s]', ' ')

     # Replace whitespace between terms with a single space
     processed = processed.str.replace(r'\s+', ' ')

     # Remove leading and trailing whitespace
     processed = processed.str.replace(r'^\s+|\s+?$', '')
```

```
[ ]  processed = processed.str.lower()
     print(processed)
```

```
0        go until jurong point crazy available only in ...
1                                 ok lar joking wif u oni
2        free entry in numbr a wkly comp to win fa cup ...
```



```
       |  Unzipping corpora/gazetteers.zip.
       | Downloading package genesis to /root/nltk_data
```

```
[ ]  from nltk.corpus import stopwords


     stop_words = set(stopwords.words('english'))

     processed = processed.apply(lambda x: ' '.join(
         term for term in x.split() if term not in stop_words))
```

```
[ ]  ps = nltk.PorterStemmer()

     processed = processed.apply(lambda x: ' '.join(
         ps.stem(term) for term in x.split()))
```

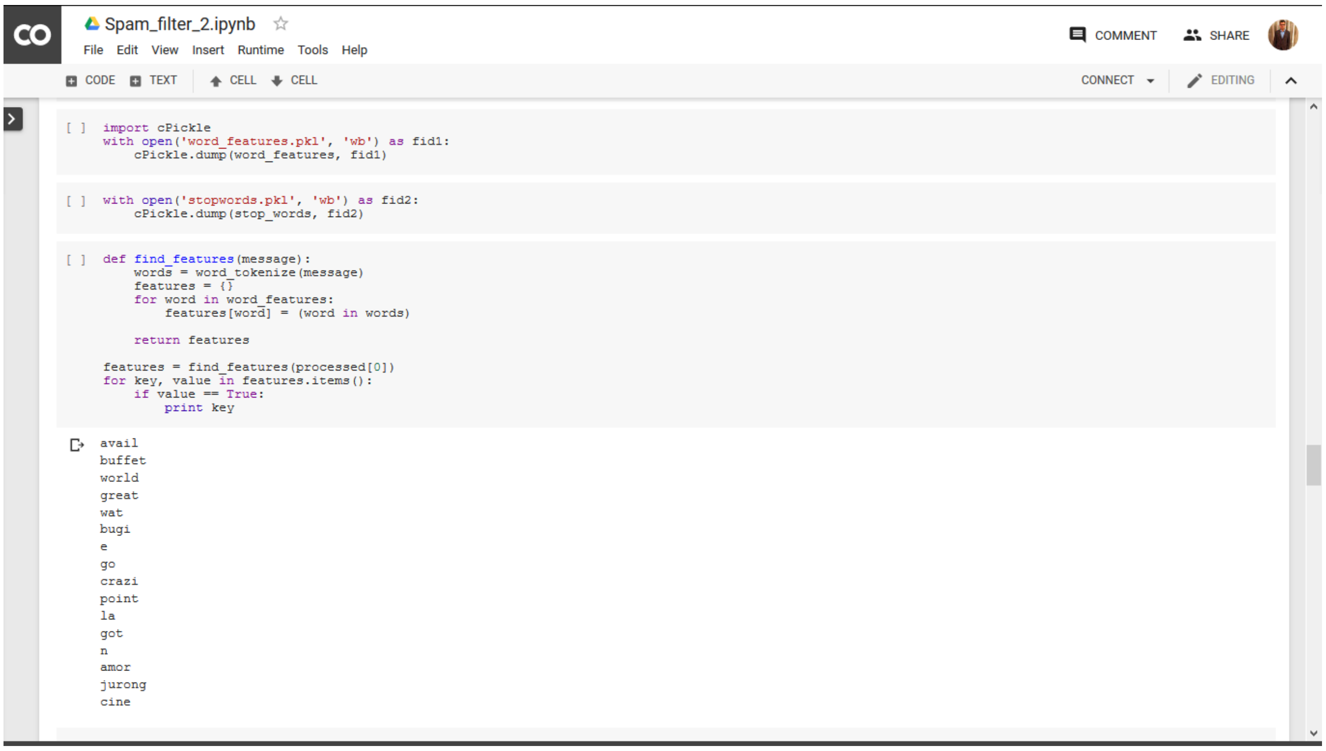
```
[ ]  from nltk.tokenize import word_tokenize

     all_words = []

     for message in processed:
         words = word_tokenize(message)
         for w in words:
             all_words.append(w)

     all_words = nltk.FreqDist(all_words)
```

```
[ ]  print('Number of words: {}'.format(len(all_words)))
     print('Most common words: {}'.format(all_words.most_common(15)))
```

```
Number of words: 6574
Most common words: [(u'numbr', 2649), (u'u', 1207), (u'call', 674), (u'go', 456), (u'get', 451), (u'ur', 391), (u'gt', 318), (u'lt', 316), (u'come', 304
```

```
[ ]  # use the 1500 most common words as features
     word_features = list(all_words.keys())
```

```
[ ]  import cPickle
     with open('word_features.pkl', 'wb') as fid1:
         cPickle.dump(word_features, fid1)
```

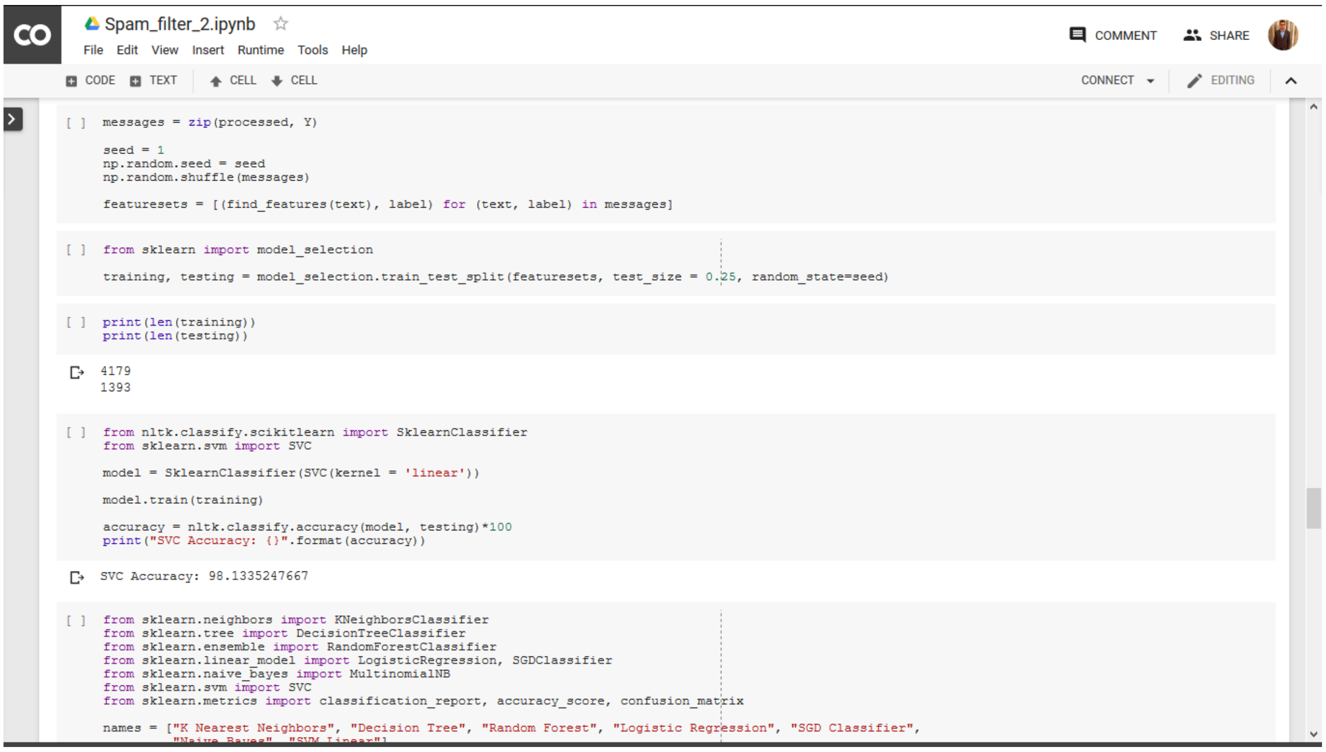


```
[ ]  import cPickle
     with open('word_features.pkl', 'wb') as fid1:
         cPickle.dump(word_features, fid1)
```

```
[ ]  with open('stopwords.pkl', 'wb') as fid2:
         cPickle.dump(stop_words, fid2)
```

```
[ ]  def find_features(message):
         words = word_tokenize(message)
         features = {}
         for word in word_features:
             features[word] = (word in words)

         return features

     features = find_features(processed[0])
     for key, value in features.items():
         if value == True:
             print key
```

```
avail
buffet
world
great
wat
bugi
e
go
crazi
point
la
got
n
amor
jurong
cine
```



```
[ ]  messages = zip(processed, Y)

     seed = 1
     np.random.seed = seed
     np.random.shuffle(messages)

     featuresets = [(find_features(text), label) for (text, label) in messages]
```

```
[ ]  from sklearn import model_selection

     training, testing = model_selection.train_test_split(featuresets, test_size = 0.25, random_state=seed)
```

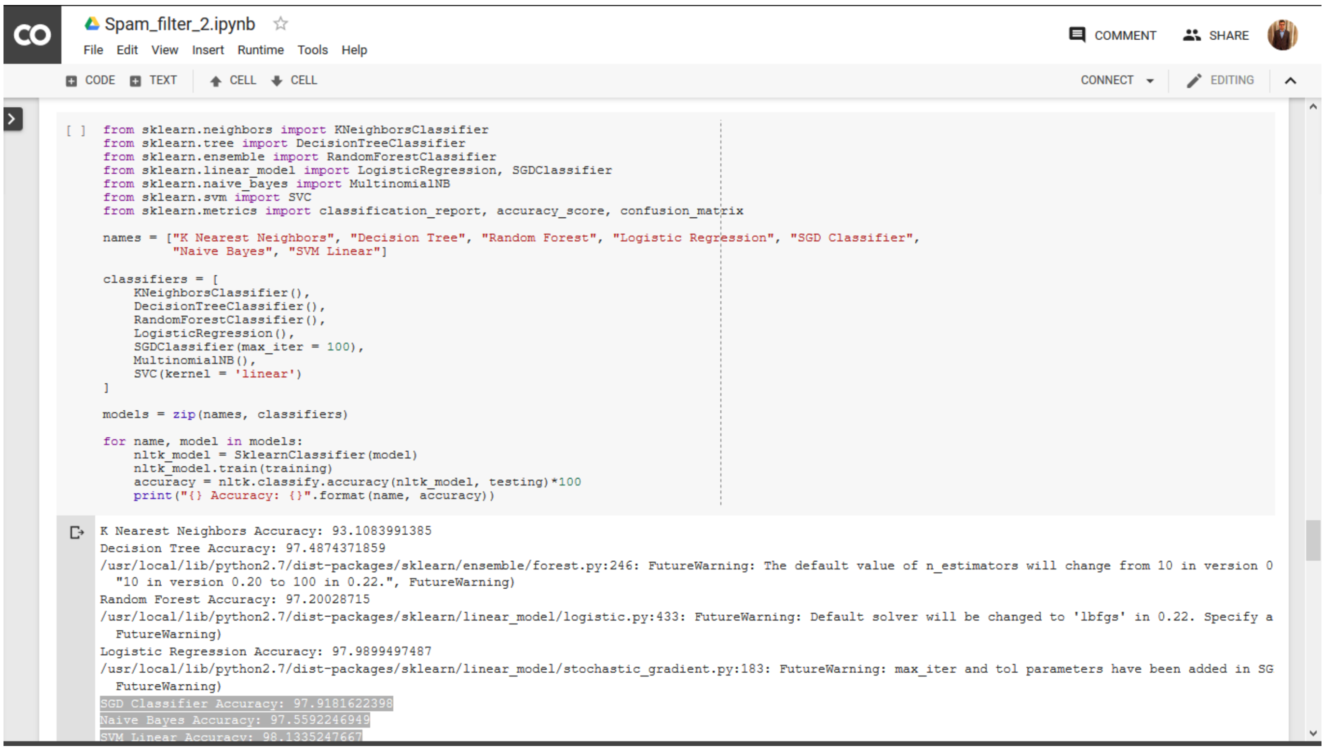
```
[ ]  print(len(training))
     print(len(testing))
```

```
4179
1393
```

```
[ ]  from nltk.classify.scikitlearn import SklearnClassifier
     from sklearn.svm import SVC

     model = SklearnClassifier(SVC(kernel = 'linear'))

     model.train(training)

     accuracy = nltk.classify.accuracy(model, testing)*100
     print("SVC Accuracy: {}".format(accuracy))
```

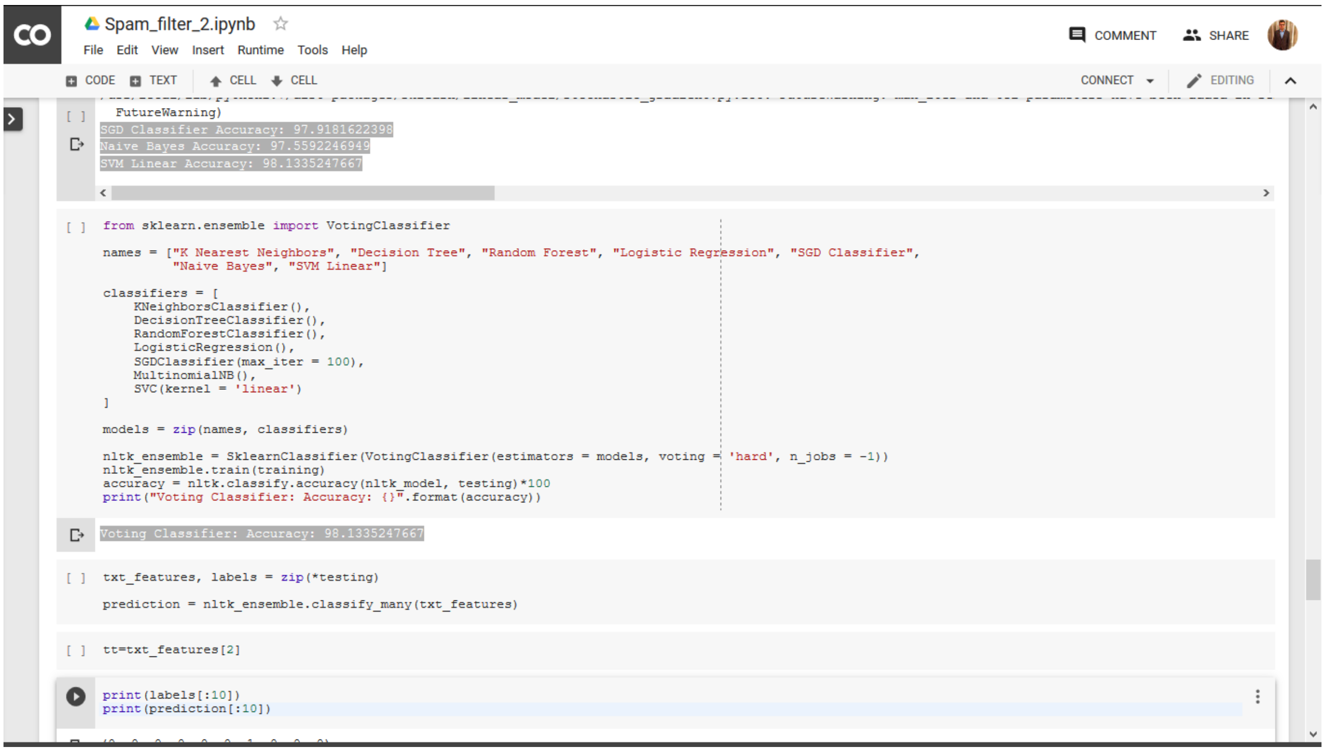
```
SVC Accuracy: 98.1335247667
```

```
[ ]  from sklearn.neighbors import KNeighborsClassifier
     from sklearn.tree import DecisionTreeClassifier
     from sklearn.ensemble import RandomForestClassifier
     from sklearn.linear_model import LogisticRegression, SGDClassifier
     from sklearn.naive_bayes import MultinomialNB
     from sklearn.svm import SVC
     from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

     names = ["K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic Regression", "SGD Classifier",
              "Naive Bayes", "SVM Linear"]
```



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

names = ["K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic Regression", "SGD Classifier",
         "Naive Bayes", "SVM Linear"]

classifiers = [
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    LogisticRegression(),
    SGDClassifier(max_iter = 100),
    MultinomialNB(),
    SVC(kernel = 'linear')
]

models = zip(names, classifiers)

for name, model in models:
    nltk_model = SklearnClassifier(model)
    nltk_model.train(training)
    accuracy = nltk.classify.accuracy(nltk_model, testing)*100
    print("{} Accuracy: {}".format(name, accuracy))
```

```
K Nearest Neighbors Accuracy: 93.1083991385
Decision Tree Accuracy: 97.4874371859
/usr/local/lib/python2.7/dist-packages/sklearn/ensemble/forest.py:246: FutureWarning: The default value of n_estimators will change from 10 in version 0
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
Random Forest Accuracy: 97.20028715
/usr/local/lib/python2.7/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
  FutureWarning)
Logistic Regression Accuracy: 97.9899497487
/usr/local/lib/python2.7/dist-packages/sklearn/linear_model/stochastic_gradient.py:183: FutureWarning: max_iter and tol parameters have been added in SG
  FutureWarning)
SGD Classifier Accuracy: 97.9181622398
Naive Bayes Accuracy: 97.5592246949
SVM Linear Accuracy: 98.1335247667
```



```
  FutureWarning)
SGD Classifier Accuracy: 97.9181622398
Naive Bayes Accuracy: 97.5592246949
SVM Linear Accuracy: 98.1335247667
```

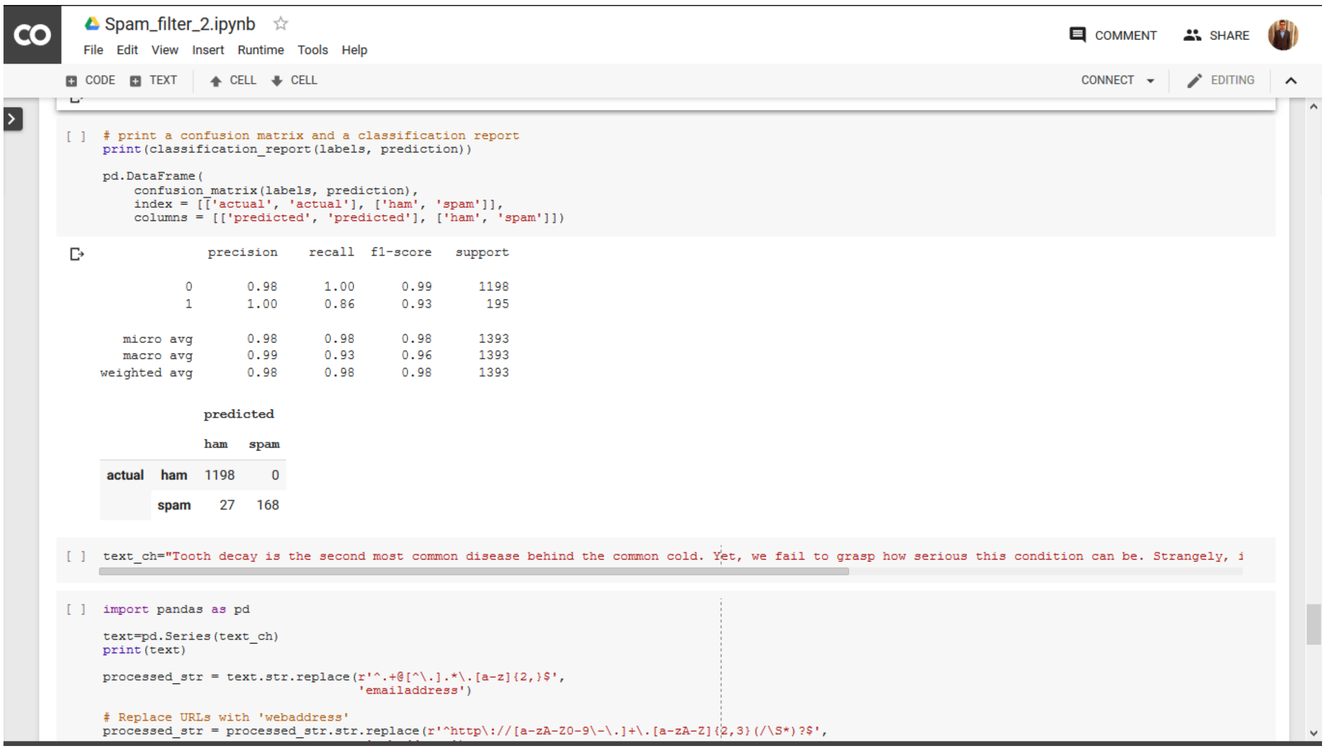
```
from sklearn.ensemble import VotingClassifier

names = ["K Nearest Neighbors", "Decision Tree", "Random Forest", "Logistic Regression", "SGD Classifier",
         "Naive Bayes", "SVM Linear"]

classifiers = [
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    LogisticRegression(),
    SGDClassifier(max_iter = 100),
    MultinomialNB(),
    SVC(kernel = 'linear')
]

models = zip(names, classifiers)

nltk_ensemble = SklearnClassifier(VotingClassifier(estimators = models, voting = 'hard', n_jobs = -1))
nltk_ensemble.train(training)
accuracy = nltk.classify.accuracy(nltk_model, testing)*100
print("Voting Classifier: Accuracy: {}".format(accuracy))
```

```
Voting Classifier: Accuracy: 98.1335247667
```

```
txt_features, labels = zip(*testing)

prediction = nltk_ensemble.classify_many(txt_features)
```

```
tt=txt_features[2]
```

```
print(labels[:10])
print(prediction[:10])
```



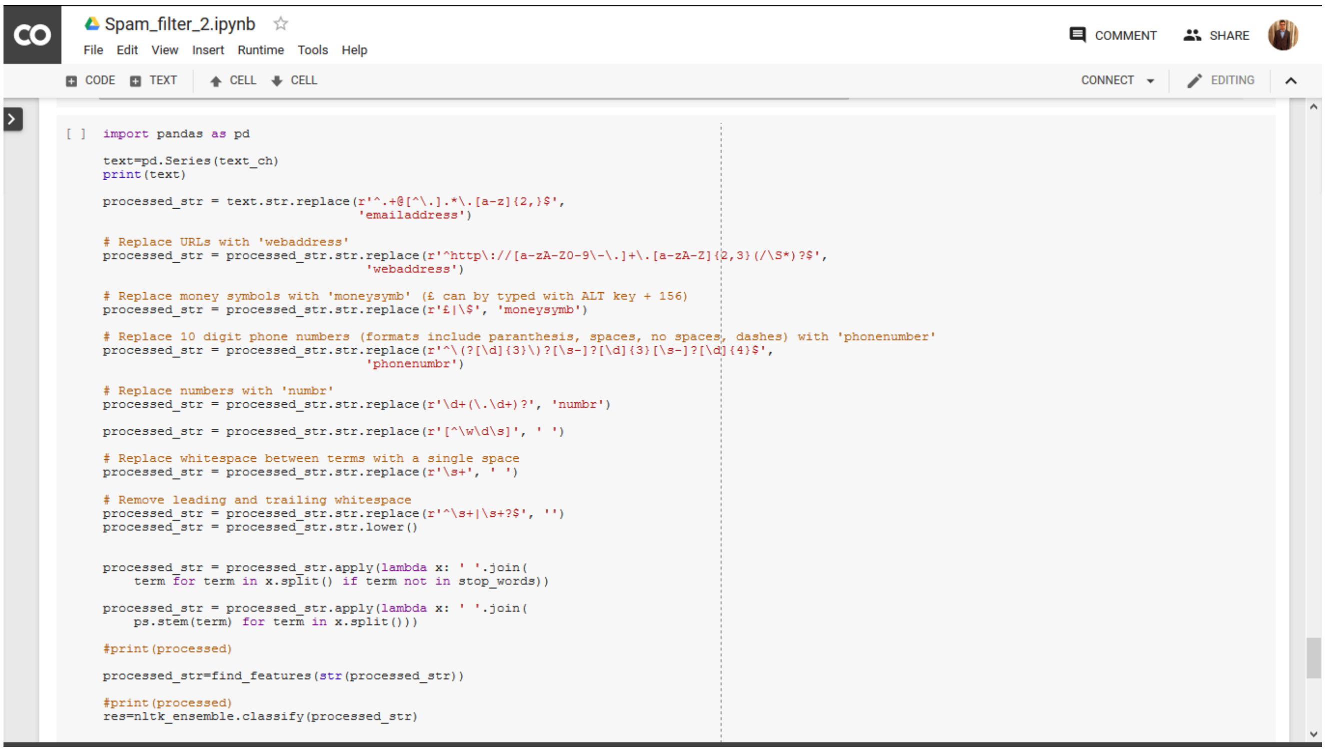
```
[ ] # print a confusion matrix and a classification report
    print(classification_report(labels, prediction))

    pd.DataFrame(
        confusion_matrix(labels, prediction),
        index = [['actual', 'actual'], ['ham', 'spam']],
        columns = [['predicted', 'predicted'], ['ham', 'spam']])
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1198
           1       1.00      0.86      0.93       195

   micro avg       0.98      0.98      0.98      1393
   macro avg       0.99      0.93      0.96      1393
weighted avg       0.98      0.98      0.98      1393
```

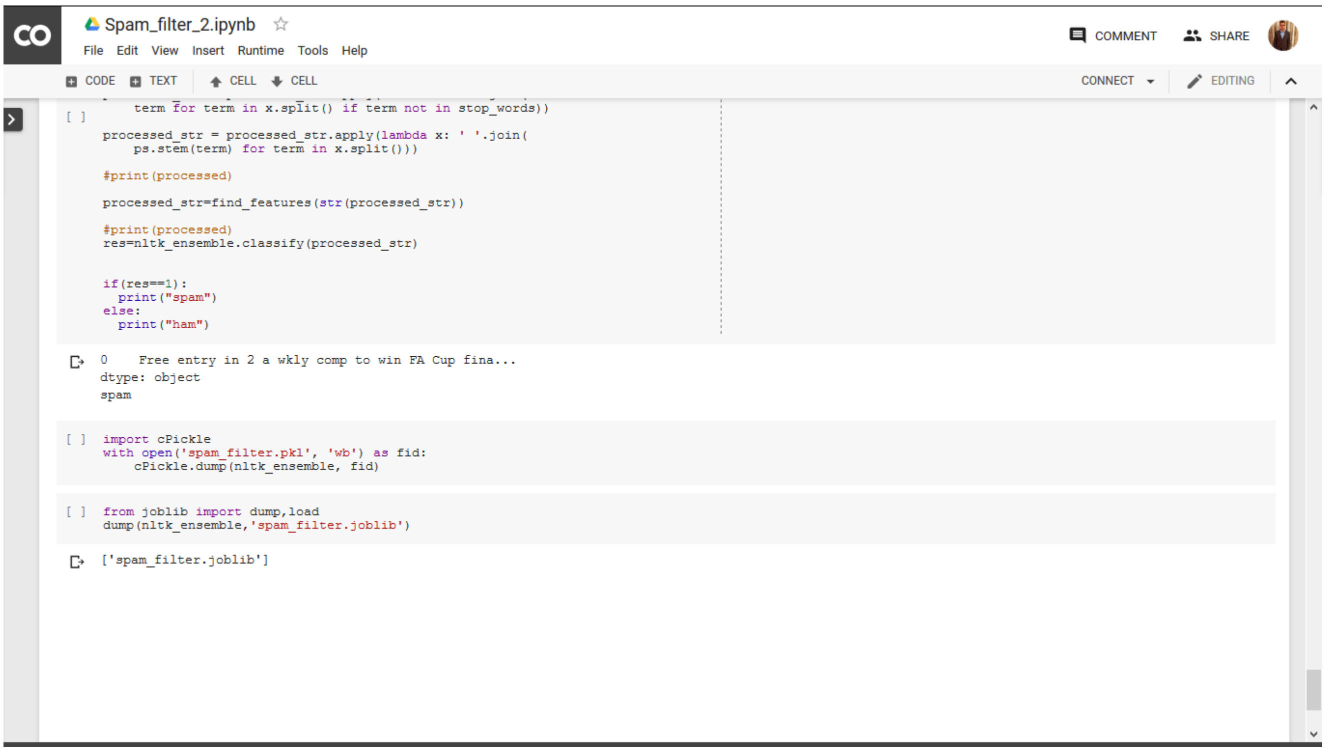
		predicted	
		ham	spam
actual	ham	1198	0
	spam	27	168

```
[ ] text_ch="Tooth decay is the second most common disease behind the common cold. Yet, we fail to grasp how serious this condition can be. Strangely, i
```

```
[ ] import pandas as pd

    text=pd.Series(text_ch)
    print(text)

    processed_str = text.str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                     'emailaddress')

    # Replace URLs with 'webaddress'
    processed_str = processed_str.str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
```



```
[ ] import pandas as pd

    text=pd.Series(text_ch)
    print(text)

    processed_str = text.str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',
                                     'emailaddress')

    # Replace URLs with 'webaddress'
    processed_str = processed_str.str.replace(r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$',
                                      'webaddress')

    # Replace money symbols with 'moneysymb' (£ can by typed with ALT key + 156)
    processed_str = processed_str.str.replace(r'£|\$', 'moneysymb')

    # Replace 10 digit phone numbers (formats include paranthesis, spaces, no spaces, dashes) with 'phonenumber'
    processed_str = processed_str.str.replace(r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$',
                                      'phonenumbr')

    # Replace numbers with 'numbr'
    processed_str = processed_str.str.replace(r'\d+(\.\d+)?', 'numbr')

    processed_str = processed_str.str.replace(r'[^\w\d\s]', ' ')

    # Replace whitespace between terms with a single space
    processed_str = processed_str.str.replace(r'\s+', ' ')

    # Remove leading and trailing whitespace
    processed_str = processed_str.str.replace(r'^\s+|\s+?$', '')
    processed_str = processed_str.str.lower()


    processed_str = processed_str.apply(lambda x: ' '.join(
        term for term in x.split() if term not in stop_words))

    processed_str = processed_str.apply(lambda x: ' '.join(
        ps.stem(term) for term in x.split()))

    #print(processed)

    processed_str=find_features(str(processed_str))

    #print(processed)
    res=nltk_ensemble.classify(processed_str)
```



```
[ ]     term for term in x.split() if term not in stop_words))

    processed_str = processed_str.apply(lambda x: ' '.join(
        ps.stem(term) for term in x.split()))

    #print(processed)

    processed_str=find_features(str(processed_str))

    #print(processed)
    res=nltk_ensemble.classify(processed_str)


    if(res==1):
      print("spam")
    else:
      print("ham")
```

```
0    Free entry in 2 a wkly comp to win FA Cup fina...
dtype: object
spam
```

```
[ ] import cPickle
    with open('spam_filter.pkl', 'wb') as fid:
        cPickle.dump(nltk_ensemble, fid)
```

```
[ ] from joblib import dump,load
    dump(nltk_ensemble,'spam_filter.joblib')
```

```
['spam_filter.joblib']
```

5. Applications

- This model will help the company to apply for model for predicting different SPAM/HAM SMS in the text analytics market.
- This tools help understand the working knowledge of the different technique which are used in analytics.
- It also involves the design of a Text Classification Model to perform predictive analysis.

6. Future Scope

- Fake news detection
- Sarcasm detection
- Classify large data in short time which take more time if read manually
- Tagging content as categories

7. Conclusion

Automatic Text Classification is a machine learning technique. Document can be set to predefined categories best on textual content and extraction features. It has important applications in spam filtering and text mining. In the recent years the automatic categorization of texts into predefined categories is booming interest. Due to the increased availability of data & documents in the day to day basics in digital form and ensuing the need to organize them. In the research community the commanding, approach to this problem is based on machine learning techniques. The advantages of this approach efficiency, considerable savings in terms of expert manpower and straightforward portability to different domains. This report emphasizes on the text categorization that fall within the machine learning technique. How Automatic Text Classification can be used to classify SMS SPAM filtering classification model. An analysis of SMS SPAM filtering classification model has also been done using Automatic Text Classification.

8. References

- https://github.com/eduonix/nlptextclassification/blob/master/NLP%20for%20Text%20Classification%20(Jupyter%20Notebook).ipynb
- https://scikit-learn.org/stable/documentation.html
- https://medium.com/swlh/classify-emails-into-ham-and-spam-using-naive-bayes-classifier-ffddd7faa1ef