SMS Spam Classification

This notebook illustrates classification of SMS as SPAM or NOT SPAM.

Business Problem

Spam emails are something we've all experienced. spam mail, often known as junk mail, is an email that is sent to a large number of people at once, sometimes with cryptic messages, scams, or, most dangerously, phishing information. The problem here is that the email needs to be classified as spam or ham based on the content of the mail.

About Dataset

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged acording being ham (legitimate) or spam.

Installing Dependicies

```
%matplotlib inline
import matplotlib.pyplot as plt
import sklearn
import pickle
from wordcloud import WordCloud
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import learning_curve
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

Preprocessing and Exploring the Dataset

Importing the Dataset spam.csv

```
data = pd.read_csv('spam.csv', encoding='latin-1')
data.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
		U dun say so early hor... U c already then			

Next steps: Generate code with data | View recommended plots | New interactive sheet

Removing unwanted columns

```
data = data.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
data = data.rename(columns={"v2" : "text", "v1":"label"})

data[1990:2000]
```

	label	text
1990	ham	HI DARLIN IVE JUST GOT BACK AND I HAD A REALLY...
1991	ham	No other Valentines huh? The proof is on your ...
1992	spam	Free tones Hope you enjoyed your new content. ...
1993	ham	Eh den sat u book e kb liao huh...
1994	ham	Have you been practising your curtsey?
1995	ham	Shall i come to get pickle
1996	ham	Lol boo I was hoping for a laugh
1997	ham	\YEH I AM DEF UP4 SOMETHING SAT
1998	ham	Well, I have to leave for my class babe ... Yo...
1999	ham	LMAO where's your fish memory when I need it?

```
data['label'].value_counts()
```

label	count
ham	4825
spam	747

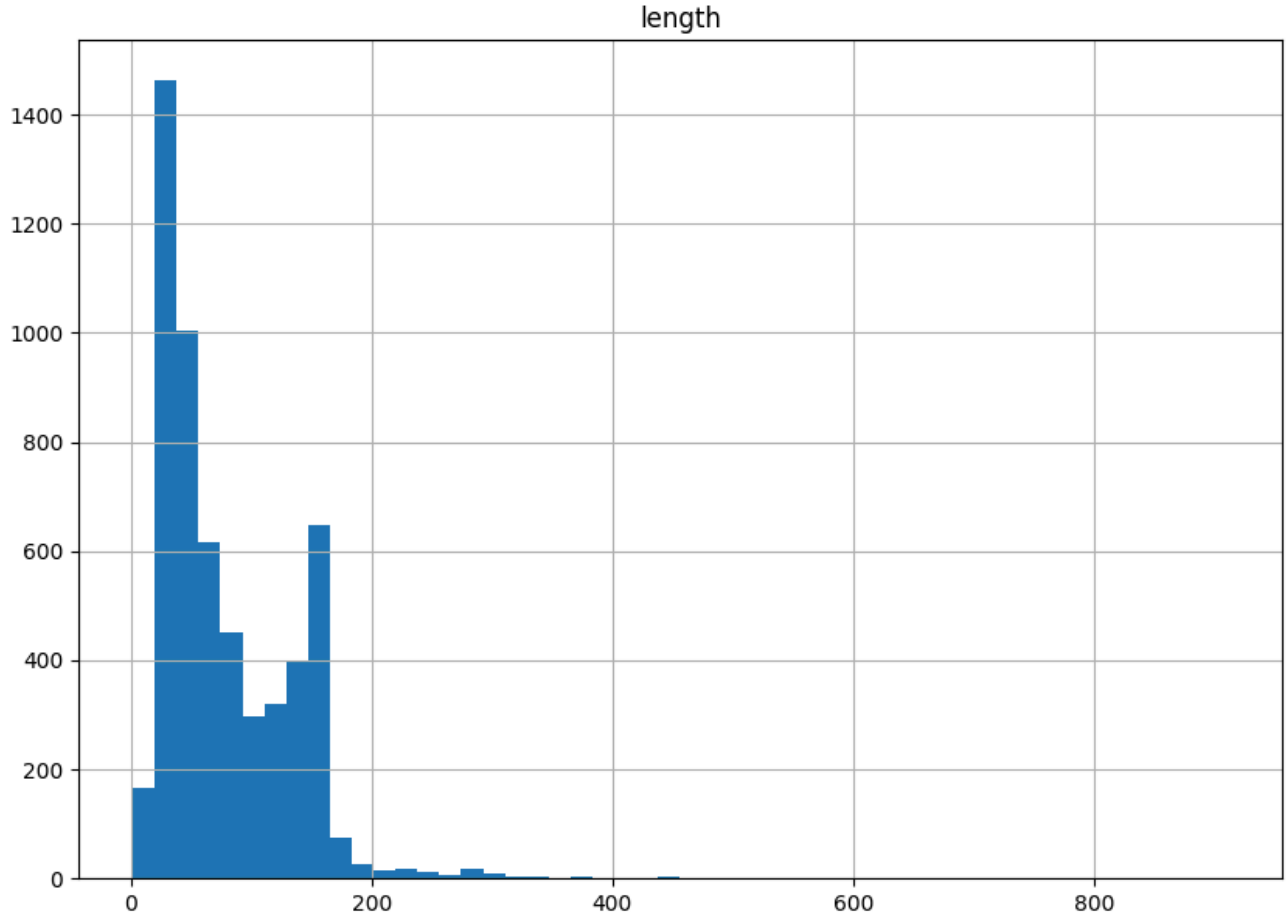
dtype: int64

```
Start coding or generate with AI.
```

Some Visualisations

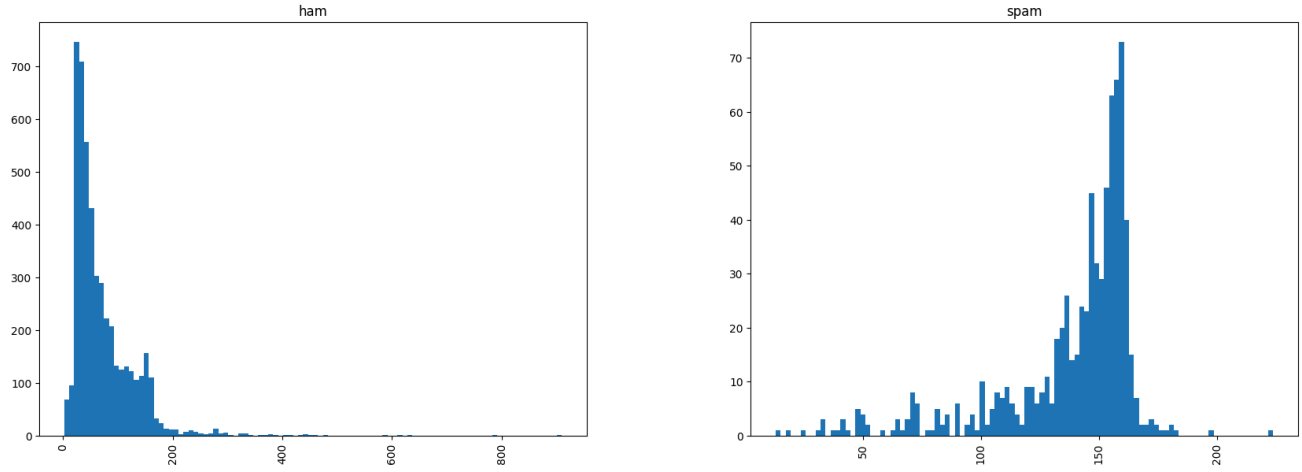
```
data['length'] = data['text'].map(lambda x: len(x))
data.hist(column='length',bins=50, figsize=(10,7))
```

```
array([[<Axes: title={'center': 'length'}>]], dtype=object)
```



```
data.hist(column='length', by='label', bins=100, figsize=(20,7))
```

```
array([<Axes: title={'center': 'ham'}>, <Axes: title={'center': 'spam'}>],
      dtype=object)
```




```
# import nltk
#nltk.download("punkt")
import warnings
warnings.filterwarnings('ignore')
```

WordClouds- to see which words are common in SPAM and NOT SPAM mesaages

```
ham_words = ''
spam_words = ''
```

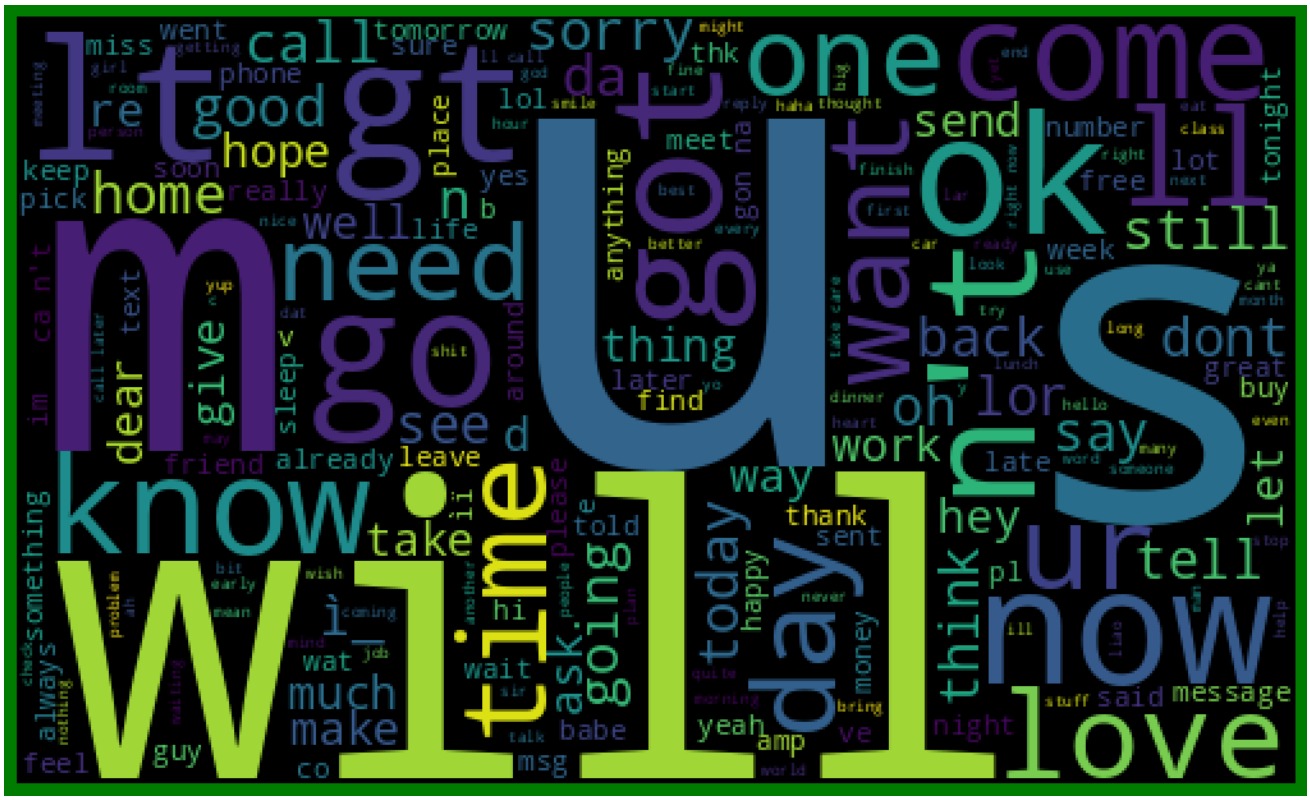
```
    for val in data[data['label'] == 'spam'].text:
        text = val.lower()
        tokens = nltk.word_tokenize(text)
        for words in tokens:
            spam_words = spam_words + words + ' '

    for val in data[data['label'] == 'ham'].text:
        text = val.lower()
        tokens = nltk.word_tokenize(text)
        for words in tokens:
            ham_words = ham_words + words + ' '


    spam_wordcloud = WordCloud(width=500, height=300).generate(spam_words)
    ham_wordcloud = WordCloud(width=500, height=300).generate(ham_words)


    #Spam Word cloud
    plt.figure( figsize=(10,8), facecolor='w')
    plt.imshow(spam_wordcloud)
    plt.axis("off")
    plt.tight_layout(pad=0)
    plt.show()
```


```
plt.figure( figsize=(10,8), facecolor='g')
plt.imshow(ham_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```




```
data = data.replace(['ham','spam'],[0, 1])
```

```
data.head(10)
```

	label	text	length
0	0	Go until jurong point, crazy.. Available only ...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61
5	1	FreeMsg Hey there darling it's been 3 week's n...	148
6	0	Even my brother is not like to speak with me. ...	77
7	0	As per your request 'Melle Melle (Oru Minnamin...	160
8	1	WINNER!! As a valued network customer you have...	158
9	1	Had your mobile 11 months or more? U R entitle...	154

Next steps: Generate code with `data` View recommended plots New interactive sheet

Text Preproceaaing

- **Convert every text to lower case**
- **Remove text username**
- **Remove punctuations, numbers and special characters**
- **Convert more than 2 letter repetitions to 2 letter (example (wooooooow --> woow))**
- **Remove extra spaces**
- **Remove URLs**
- **Emoji analysis**
- **Handle contractions words**
  - **" can't " >> " can not "**
  - **" won't " >> " will not "**
  - **" should't " >> " should not "**
- **Tokenization**
- **(Optional) Remove Stop words**
- **(Optional) Text Normalization (Stemming / Lemmatization)**

```
def emoji(text):
    # Smile -- :), : ), :-), (:, ( :, (-:, :') , :O
    text = re.sub(r'(:\s?\)|:-\)|\(\s?:|\(-:|:\'\)|:O)', ' positiveemoji ', text)
    # Laugh -- :D, : D, :-D, xD, x-D, XD, X-D
    text = re.sub(r'(:\s?D|:-D|x-?D|X-?D)', ' positiveemoji ', text)
    # Love -- <3, :*
    text = re.sub(r'(<3|:\*)', ' positiveemoji ', text)
    # Wink -- ;-), ;), ;-D, ;D, (;,  (-; , @-)
    text = re.sub(r'(;-?\)|;-?D|\(-?;|@-\))', ' positiveemoji ', text)
    # Sad -- :-(, : (, :(, ):, )-:, :-/ , :-|
    text = re.sub(r'(:\s?\(|:-\(|\)\s?:|\)-:|:-/|:-\|)', ' negetiveemoji ', text)
    # Cry -- :,(, :'(, :"(
    text = re.sub(r'(:,\(|:\'\(|:"\()', ' negetiveemoji ', text)
    return text


import re

def process_text_data(text):
    text = text.lower()                                          # Lowercases the stri
    text = re.sub('@[^\s]+', '', text)                           # Removes usernames
    text = re.sub('((www\.[^\s]+)|(https?://[^\s]+))', ' ', text)   # Remove URLs
    text = re.sub(r"\d+", " ", str(text))                        # Removes all digits
    text = re.sub('&quot;'," ", text)                            # Remove (&quot;)
    text = emoji(text)                                           # Replaces Emojis
    text = re.sub(r"\b[a-zA-Z]\b", "", str(text))                # Removes all single

    text = re.sub(r"[^\w\s]", " ", str(text))                    # Removes all punctua
    text = re.sub(r'(.)\1+', r'\1\1', text)                      # Convert more than 2
    text = re.sub(r"\s+", " ", str(text))                        # Replaces double spa
    return text


data['processed_text'] = data['text'].apply(process_text_data)


data
```

	label	text	length	processed_text
0	0	Go until jurong point, crazy.. Available only ...	111	go until jurong point crazy available only in ...
1	0	Ok lar... Joking wif u oni...	29	ok lar joking wif oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	free entry in wkly comp to win fa cup final tk...
3	0	U dun say so early hor... U c already then say...	49	dun say so early hor already then say
4	0	Nah I don't think he goes to usf, he lives aro...	61	nah don think he goes to usf he lives around h...
...	...	...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...	161	this is the nd time we have tried contact have...
5568	0	Will Ì_ b going to esplanade fr home?	37	will ì_ going to esplanade fr home
5569	0	Pity, * was in mood for that.	57	pity was in mood for that so any

Next steps: Generate code with data | View recommended plots | New interactive sheet

Removing Stopwords from the messages

["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will", "just", "don", "should", "now"]

We can't use every word from here. Because some words like "no", "nor" etc. playes significant roles in sentiment.

So we will be making our custom list of stopwords.

```
import string
def text_process(text):

    text = text.translate(str.maketrans('', '', string.punctuation))
    text = [word for word in text.split() if word.lower() not in stopwords.words('english

    return " ".join(text)


data['processed_text'] = data['processed_text'].apply(text_process)


data.head()
```

	label	text	length	processed_text
0	0	Go until jurong point, crazy.. Available only ...	111	go jurong point crazy available bugis great wo...
1	0	Ok lar... Joking wif u oni...	29	ok lar joking wif oni
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	free entry wkly comp win fa cup final tkts st ...
3	0	U dun say so early hor... U c already then say...	49	dun say early hor already say

Next steps: Generate code with `data` View recommended plots New interactive sheet

```
text = pd.DataFrame(data['processed_text'])
label = pd.DataFrame(data['label'])
```

Spelling correction

```
from textblob import TextBlob

# Join the list of words back into a single string for each row
data['processed_text'] = data['processed_text'].apply(lambda x: ' '.join(x))

# Now apply TextBlob for spell correction
data['processed_text'] = data['processed_text'].apply(lambda x: str(TextBlob(x).correct()

data.head(10)
```

	label	text	length	processed_text
0	0	Go until jurong point, crazy.. Available only ...	111	g o j u r o n g p o i n t c r a z y a ...
1	0	Ok lar... Joking wif u oni...	29	o k l a r j o k i n g w i f o n i
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	f r e e e n t r y w k l y c o m p w i ...
3	0	U dun say so early hor... U c already then say...	49	d u n s a y e a r l y h o r a l r e a ...
4	0	Nah I don't think he goes to usf, he lives aro...	61	n a h t h i n k g o e s u s f l i v e ...
5	1	FreeMsg Hey there darling it's been 3 week's n...	148	f r e e m s g h e y d a r l i n g w e e ...
6	0	Even my brother is not like to speak with me. ...	77	e v e n b r o t h e r l i k e s p e a k ...

Next steps: Generate code with `data` View recommended plots New interactive sheet

Stemming

The below code is commented out because *Lemmatization* gives slightly better accuracy in this sentiment analysis than *Stemming*. If you want to check, then uncomment the code below, comment the Lemmatization code below and run the whole code again.

```
from nltk.stem.porter import *
stemmer = PorterStemmer()

data['processed_text'] =data['processed_text'].apply(lambda x: [stemmer.stem(i) for i in
data['processed_text'].head()
```

	processed_text
0	[g, , o, , , , j, , u, , r, , o, , n, , g, , , , p, , o, , i, , n, , t, , , , c, , r, , a, , z, , y, , , , a, , v, , a, , i, , l, , a, , b, , l, , e, , , , b, , u, , g, , i, , s, , , , g, , r, , e, , a, , t, , , , w, , o, , r, , l, , d, , , , ...]
1	[o, , k, , , , l, , a, , r, , , , j, , o, , k, , i, , n, , g, , , , w, , i, , f, , , , o, , n, , i]
2	[f, , r, , e, , e, , , , e, , n, , t, , r, , y, , , , w, , k, , l, , y, , , , c, , o, , m, , p, , , , w, , i, , n, , , , f, , a, , , , c, , u, , p, , , , f, , i, , n, , a, , l, , , , t, , k, , t, , s, , , , s, , t, , , , m, , a, , y, , , , ...]
3	[d, , u, , n, , , , s, , a, , y, , , , e, , a, , r, , l, , y, , , , h, , o, , r, , , , a, , l, , r, , e, , a, , d, , y, , , , s, , a, , y]
4	[n, , a, , h, , , , t, , h, , i, , n, , k, , , , g, , o, , e, , s, , , , u, , s, , f, , , , l, , i, , v, , e, , s, , , , a, , r, , o, , u, , n, , d, , , , t, , h, , o, , u, , g, , h]

Lemmatization

```
import nltk
nltk.download('wordnet')

from nltk.stem.wordnet import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

data['processed_text'] =data['processed_text'].apply(lambda x: [lemmatizer.lemmatize(i) f
data.head()
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

	label	text	length	processed_text
0	0	Go until jurong point, crazy.. Available only ...	111	[g, , o, , , , j, , u, , r, , o, , n, ...
1	0	Ok lar... Joking wif u oni...	29	[o, , k, , , , l, , a, , r, , , , j, ...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155	[f, , r, , e, , e, , , , e, , n, , t, ...
3	0	U dun say so early hor... U c already then say...	49	[d, , u, , n, , , , s, , a, , y, , , ...

Next steps: Generate code with `data` View recommended plots New interactive sheet

Converting words to vectors

- First create a vocabulary of all words in the dataset (text messages)
- Vector created as follows :
  - positions with respect to highest occuring word
  - Eg : 1 at first index means first word in vocab(most frequent occuring in vocab which is 'of') occurs twice in this sentence

```
text
```

	processed_text
0	go jurong point crazy available bugis great wo...
1	ok lar joking wif oni
2	free entry wkly comp win fa cup final tkts st ...
3	dun say early hor already say
4	nah think goes usf lives around though
...	...
5567	nd time tried contact å pound prize claim easy...
5568	ì going esplanade fr home
5569	pity mood suggestions
5570	guy bitching acted like interested buying some...
5571	rofl true name

5572 rows × 1 columns

Next steps: Generate code with text | View recommended plots | New interactive sheet

```
## Counting how many times a word appears in the dataset

from collections import Counter

total_counts = Counter()
for i in range(len(text)):
    for word in text.values[i][0].split(" "):
        total_counts[word] += 1

print("Total words in data set: ", len(total_counts))
```

```
Total words in data set:  7511
```

```
## Sorting in decreasing order (Word with highest frequency appears first)
vocab = sorted(total_counts, key=total_counts.get, reverse=True)
print(vocab[:60])
```

```
['call', 'get', 'ur', 'positiveemoji', 'gt', 'lt', 'å', 'ok', 'free', 'go', 'know', '
```

```
# Mapping from words to index

vocab_size = len(vocab)
word2idx = {}
#print vocab_size
for i, word in enumerate(vocab):
    word2idx[word] = i
```

```
### Text to Vector
def text_to_vector(text):
    word_vector = np.zeros(vocab_size)
    for word in text.split(" "):
        if word2idx.get(word) is None:
            continue
        else:
            word_vector[word2idx.get(word)] += 1
    return np.array(word_vector)


## Convert all titles to vectors
word_vectors = np.zeros((len(text), len(vocab)), dtype=np.int_)
for ii, (_, text_) in enumerate(text.iterrows()):
    word_vectors[ii] = text_to_vector(text_[0])


word_vectors.shape
```

```
(5572, 7511)
```

Converting words to vectors using TFIDF Vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(data['text'])
vectors.shape
```

```
(5572, 8672)
```

Choosing which algorithm we want to use a features : TFIDF or using custom vocabulary ?

```
#features = word_vectors
features = vectors


features
```

```
<5572x8672 sparse matrix of type '<class 'numpy.float64'>'
        with 73916 stored elements in Compressed Sparse Row format>
```

Splitting into training and test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, data['label'], test_size=0.
```

```
print (X_train.shape)
print (X_test.shape)
print (y_train.shape)
print (y_test.shape)
```

```
(4736, 8672)
(836, 8672)
(4736,)
(836,)
```

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy={1:4000}, random_state=42)

X_train, y_train = smote.fit_resample(X_train.astype('float'), y_train)
print("Before SMOTE:", Counter(y_train))
# print("After SMOTE:", Counter(y_train_smote))
```

```
Before SMOTE: Counter({0: 4100, 1: 4000})
```

Initial Classification using XGBoost

XGBoost using custom vocab

```
import xgboost as xgb

# Set our parameters for xgboost
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'error'
params['eta'] = 0.02
params['max_depth'] = 10

d_train = xgb.DMatrix(X_train, label=y_train)

watchlist = [(d_train, 'train')]

bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=50, verbose_eval=1
```

```
[0]     train-error:0.02235
[10]    train-error:0.01667
[20]    train-error:0.01642
[30]    train-error:0.01568
[40]    train-error:0.01444
[50]    train-error:0.01284
[60]    train-error:0.01160
[70]    train-error:0.01074
[80]    train-error:0.00926
```

```
[90]    train-error:0.00889
[100]   train-error:0.00815
[110]   train-error:0.00790
[120]   train-error:0.00753
[130]   train-error:0.00691
[140]   train-error:0.00642
[150]   train-error:0.00593
[160]   train-error:0.00593
[170]   train-error:0.00580
[180]   train-error:0.00543
[190]   train-error:0.00531
[200]   train-error:0.00519
[210]   train-error:0.00481
[220]   train-error:0.00457
[230]   train-error:0.00444
[240]   train-error:0.00444
[250]   train-error:0.00420
[260]   train-error:0.00420
[270]   train-error:0.00420
[280]   train-error:0.00420
[290]   train-error:0.00420
[300]   train-error:0.00407
[310]   train-error:0.00383
[320]   train-error:0.00358
[330]   train-error:0.00333
[340]   train-error:0.00333
[350]   train-error:0.00296
[360]   train-error:0.00296
[370]   train-error:0.00284
[380]   train-error:0.00272
[390]   train-error:0.00259
[399]   train-error:0.00259
```

XGBoost using TFIDF

```
import xgboost as xgb

# Set our parameters for xgboost
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'error'
params['eta'] = 0.02
params['max_depth'] = 10

d_train = xgb.DMatrix(X_train, label=y_train)

watchlist = [(d_train, 'train')]

bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=400, verbose_eval=
```

```
[0]     train-error:0.02235
[10]    train-error:0.01667
[20]    train-error:0.01642
[30]    train-error:0.01568
[40]    train-error:0.01444
[50]    train-error:0.01284
```

```
[60]    train-error:0.01160
[70]    train-error:0.01074
[80]    train-error:0.00926
[90]    train-error:0.00889
[100]   train-error:0.00815
[110]   train-error:0.00790
[120]   train-error:0.00753
[130]   train-error:0.00691
[140]   train-error:0.00642
[150]   train-error:0.00593
[160]   train-error:0.00593
[170]   train-error:0.00580
[180]   train-error:0.00543
[190]   train-error:0.00531
[200]   train-error:0.00519
[210]   train-error:0.00481
[220]   train-error:0.00457
[230]   train-error:0.00444
[240]   train-error:0.00444
[250]   train-error:0.00420
[260]   train-error:0.00420
[270]   train-error:0.00420
[280]   train-error:0.00420
[290]   train-error:0.00420
[300]   train-error:0.00407
[310]   train-error:0.00383
[320]   train-error:0.00358
[330]   train-error:0.00333
[340]   train-error:0.00333
[350]   train-error:0.00296
[360]   train-error:0.00296
[370]   train-error:0.00284
[380]   train-error:0.00272
[390]   train-error:0.00259
[399]   train-error:0.00259
```

```
from sklearn.metrics import accuracy_score

# Predict values for test set
d_test = xgb.DMatrix(X_test)
p_test = bst.predict(d_test)

# Apply function round() to each element in np array
# so predictions are all either 0 or 1.
npround = np.vectorize(round)
p_test_ints = npround(p_test)

# Error rate for test set
accuracy = accuracy_score(y_test, p_test_ints)
print("Test Accuracy: ", accuracy)
```

```
Test Accuracy:  0.9868421052631579
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, p_test_ints))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       725
           1       0.95      0.95      0.95       111

    accuracy                           0.99       836
   macro avg       0.97      0.97      0.97       836
weighted avg       0.99      0.99      0.99       836
```

Test Accuracy using TFIDF Algorithm: 0.971291866029

Test Accuracy using custom vocab: 0.972488038278

Classifying using sklearn pre built classifiers

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier


svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier(n_neighbors=49)
mnb = MultinomialNB(alpha=0.2)
dtc = DecisionTreeClassifier(min_samples_split=7, random_state=111)
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=31, random_state=111)


clfs = {'SVC' : svc,'KN' : knc, 'NB': mnb, 'DT': dtc, 'LR': lrc, 'RF': rfc}


def train(clf, features, targets):
    clf.fit(features, targets)

def predict(clf, features):
    return (clf.predict(features))


pred_scores_word_vectors = []
for k,v in clfs.items():
    train(v, X_train, y_train)
    pred = predict(v, X_test)
    pred_scores_word_vectors.append((k, [accuracy_score(y_test , pred)]))
```

Predictions using TFIDF Vectorizer algorithm

```
predictions = pd.DataFrame.from_dict(dict(pred_scores_word_vectors),orient='index', colum
predictions
# DataFrame.from_dict(dict(items))
```

	Score
SVC	0.991627
KN	0.226077
NB	0.992823
DT	0.966507
LR	0.974880
RF	0.989234

Next steps: Generate code with predictions | View recommended plots | New interactive sheet

Predictions using custom vocabulary

```
predictions_word_vectors = pd.DataFrame.from_dict(dict(pred_scores_word_vectors),orient='
predictions_word_vectors
```

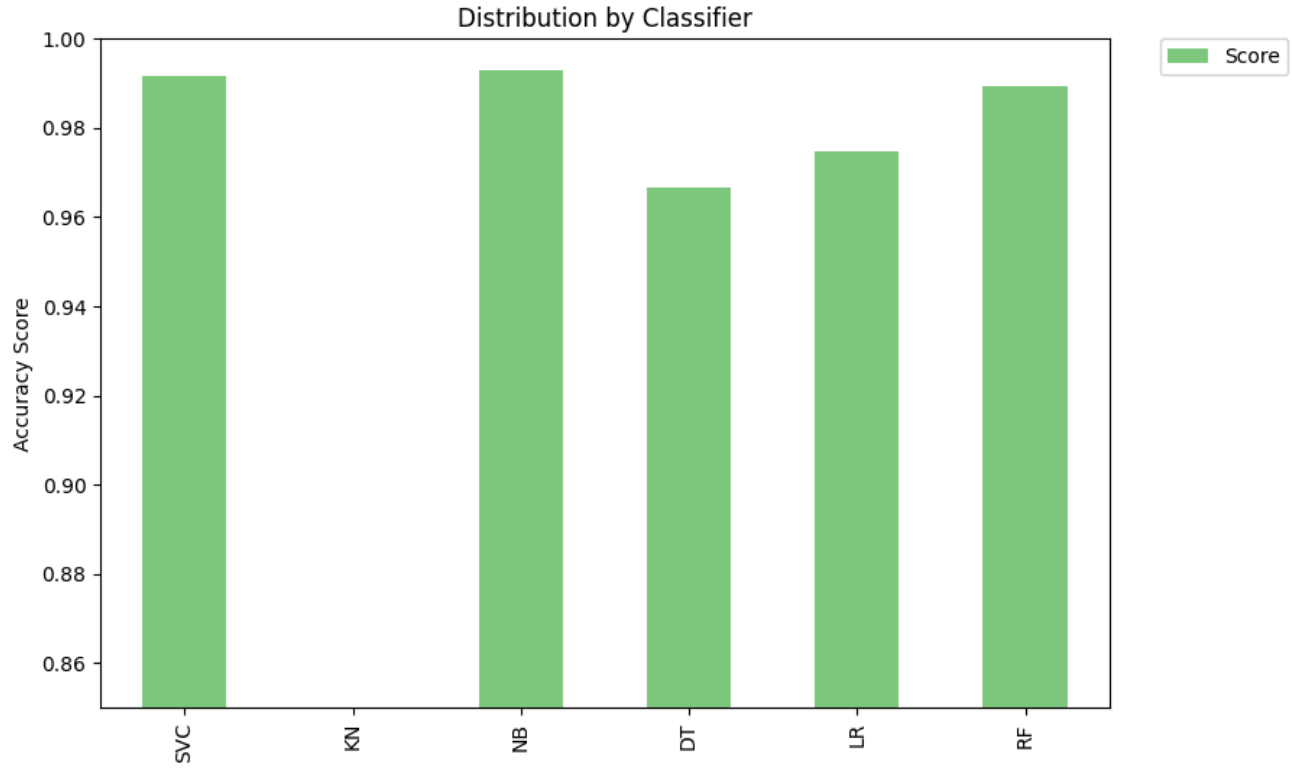
	Score_
SVC	0.991627
KN	0.226077
NB	0.992823
DT	0.966507
LR	0.974880
RF	0.989234

Next steps: code predictions_word_vectors | recommended | New interactive sheet

Plot of scores TFIDF Vectorizer algorithm

```
predictions.plot(kind='bar', ylim=(0.85,1.0), figsize=(9,6), align='center', colormap="Ac
plt.xticks(np.arange(6), predictions.index)
plt.ylabel('Accuracy Score')
plt.title('Distribution by Classifier')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

```
<matplotlib.legend.Legend at 0x7a29c6fec340>
```



Plot of scores custom vocabulary

```
predictions_word_vectors.plot(kind='bar', ylim=(0.85,1.0), figsize=(9,6), align='center',
plt.xticks(np.arange(6), predictions_word_vectors.index)
plt.ylabel('Accuracy Score')
plt.title('Distribution by Classifier - Word Vectors')
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

```
<matplotlib.legend.Legend at 0x7a298e79b910>
```