Name: Chitipolu Sri Sudheera

SMS SPAM / HAM

Text Preprocessing and Machine Learning Modeling

Context¶

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.

Content¶

The files contain one message per line. Each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text.
This corpus has been collected from free or free for research sources at the Internet:
-> A collection of 425 SMS spam messages was manually extracted from the Grumbletext Web site. This is a UK forum in which cell phone users make public claims about SMS spam messages, most of them without reporting the very spam message received. The identification of the text of spam messages in the claims is a very hard and time-consuming task, and it involved carefully scanning hundreds of web pages. The Grumbletext Web site is: [Web Link]. -> A subset of 3,375 SMS

randomly chosen ham messages of the NUS SMS Corpus (NSC), which is a dataset of about 10,000 legitimate messages collected for research at the Department of Computer Science at the National University of Singapore. The messages largely originate from Singaporeans and mostly from students attending the University. These messages were collected from volunteers who were made aware that their contributions were going to be made publicly available. The NUS SMS Corpus is avalaible at: [Web Link]. -> A list of 450 SMS ham messages collected from Caroline Tag's PhD Thesis available at [Web Link]. -> Finally, we have incorporated the SMS Spam Corpus v.0.1 Big. It has 1,002 SMS ham messages and 322 spam messages and it is public available at: [Web Link]. This corpus has been used in the following academic researches:

Problem Statement¶

use this dataset to build a prediction model that will accurately classify which texts are spam

Import Libraries¶

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
%config InlineBackend.figure_format='retina'
```

Import data¶

In [3]:

```
data=pd.read_csv("C:/Users/ADMIN/Desktop/spam.csv",encoding='latin-1')
```

In [4]:

```
data.head()
```

Out[4]:

	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
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Let's drop the unwanted columns, and rename the column name appropriately.

In [6]:

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

In [7]:

```
data.tail()
```

Out[7]:

	label	text
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will Ì_ b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

In [9]:

```
data.label.value_counts()
```

Out[9]:

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

In [10]:

```
# convert label to a numerical variable
data['label_num']=data.label.map({'ham':0,'spam':1})
```

In [11]:

```
data.head()
```

Out[11]:

	label	text	label_num
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

Train Test Split¶

Before performing text transformation, let us do train test split. Infact, we can perform k-Fold cross validation. However, due to simplicity, I am doing train test split.

In [15]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(data['text'],data['label'],test_size=0.
2,random_state=10)
```

In [16]:

```
print(X_train.shape)

print(X_test.shape)

print(Y_train.shape)

print(Y_test.shape)
```

```
(4457,)
(1115,)
(4457,)
(1115,)
```

Text Transformation¶

Various text transformation techniques such as stop word removal, lowering the texts, tfidf transformations, prunning, stemming can be performed using sklearn.feature_extraction libraries. Then, the data can be convereted into bag-of-words.
For this problem, Let us see how our model performs without removing stop words.

In [21]:

```
from sklearn.feature_extraction.text import CountVectorizer

vect=CountVectorizer()
```

Note : We can also perform tfidf transformation.

In [22]:

```
vect.fit(X_train)
```

Out[22]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

vect.fit function learns the vocabulary. We can get all the feature names from vect.get_feature_names( ).
Let us print first and last twenty features

In [23]:

```
print(vect.get_feature_names()[0:20])

print(vect.get_feature_names()[-20:])
```

```
['00', '000', '000pes', '008704050406', '0089', '0121', '01223585236', '01223585334',
'0125698789', '02', '0207', '02072069400', '02073162414', '02085076972', '021', '03',
'04', '0430', '05', '050703']
['zyada', 'åð', 'åòharry', 'åòit', 'åômorrow', 'åôrents', 'ì_', 'ì¼1', 'ìä', 'ìï', 'ó_
', 'û_', 'û_thanks', 'ûªm', 'ûªt', 'ûªve', 'ûï', 'ûïharry', 'ûò', 'ûówell']
```

In [24]:

```
X_train_df = vect.transform(X_train)
```

Now, let's transform the Test data.

In [27]:

```
X_test_df=vect.transform(X_test)
```

In [28]:

```
type(X_test_df)
```

Out[28]:

```
scipy.sparse.csr.csr_matrix
```

Visualizations¶

In [29]:

```
ham_words=''
spam_words=''
spam=data[data.label_num==1]
ham=data[data.label_num==0]
```

In [30]:

```
import nltk
from nltk.corpus import stopwords
```

In [31]:

```
for val in spam.text:
    text=val.lower()
    tokens=nltk.word_tokenize(text)
    #tokens = [word for word in tokens if word not in stopwords.words('english')]
    for words in tokens:
        spam_words=spam_words+words+''

for val in ham.text:
    text=val.lower()
    tokens=nltk.word_tokenize(text)
    for words in tokens:
        ham_words=ham_words+words+''
```

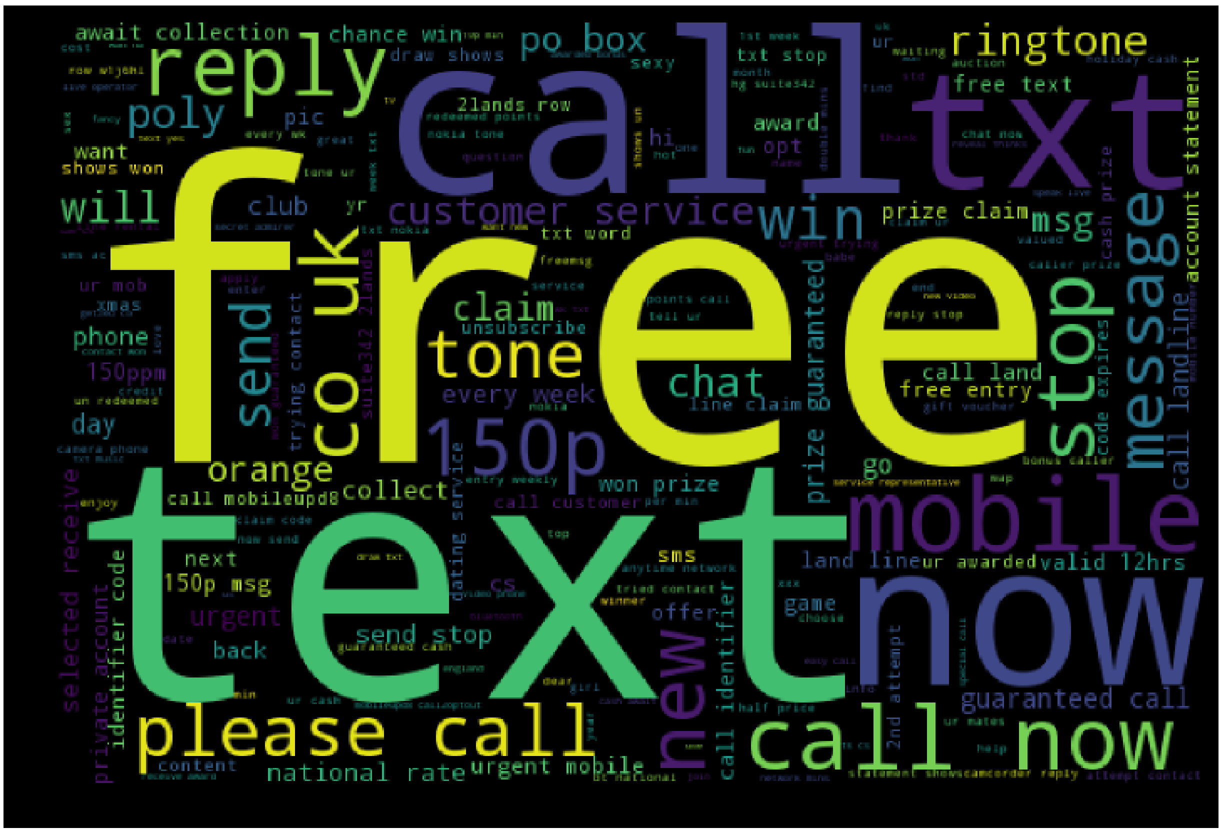
In [38]:

```
from wordcloud import WordCloud
# Generate a word cloud image
spam_wordcloud = WordCloud(width=600, height=400).generate(spam_words)
```

```
ham_wordcloud = WordCloud(width=600, height=400).generate(ham_words)
```

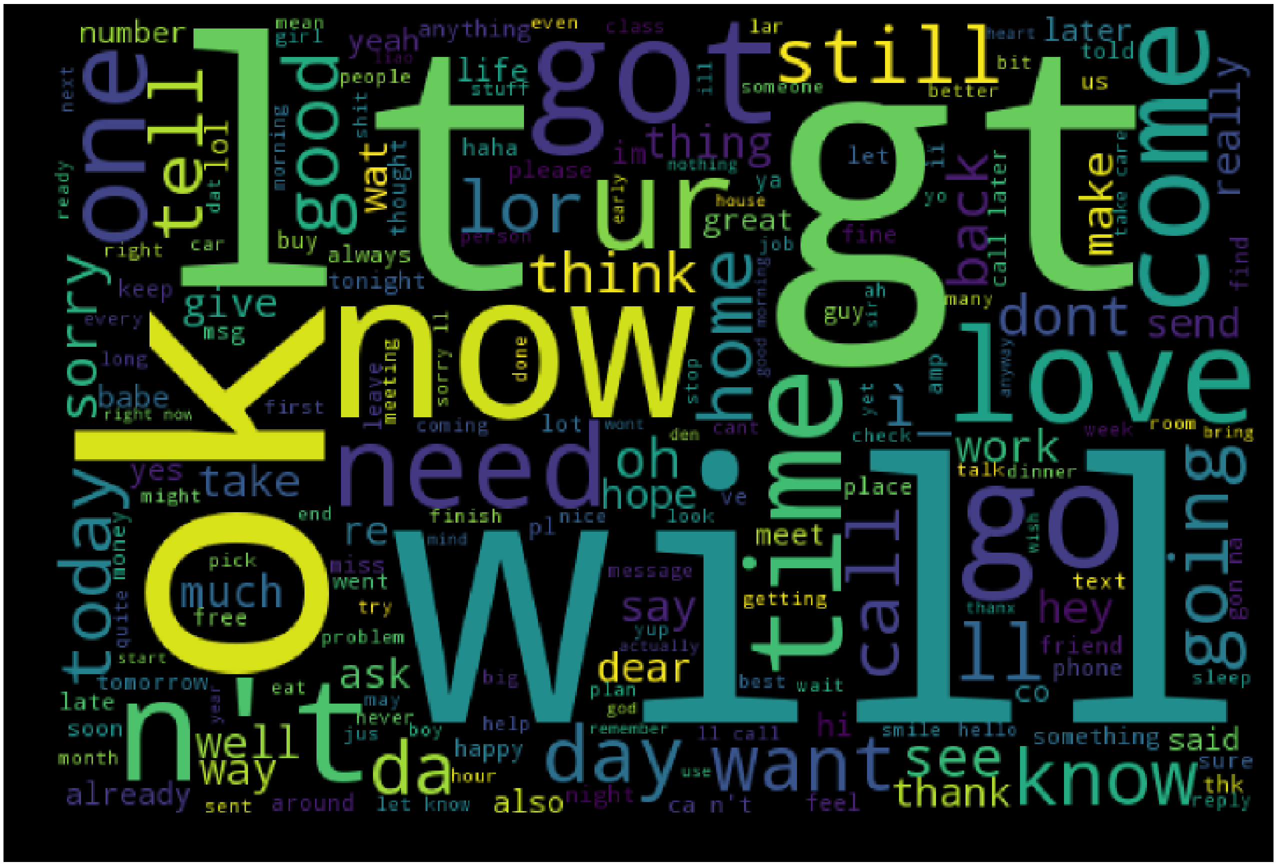
In [39]:

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



In [40]:

```
#Ham word cloud
plt.figure( figsize=(10,8), facecolor='k')
plt.imshow(ham_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```



Machine Learning models:¶

1 Multinomial Naive Bayes¶

Generally, Naive Bayes works well on text data. Multinomail Naive bayes is best suited for classification with discrete features.

In [39]:

```
prediction=dict()
from sklearn.naive_bayes import MultinomialNB
model=MultinomialNB()
model.fit(X_train_df,Y_train)
```

Out[39]:

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [40]:

```
prediction['multinomial']=model.predict(X_text_df)
```

In [41]:

```
print(prediction)
```

```
{'multinomial': array(['ham', 'ham', 'ham', ..., 'ham', 'ham', 'ham'], dtype='<U4')}
```

In [42]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [43]:

```
accuracy_score(Y_test,prediction['multinomial'])
```

Out[43]:

```
0.9883408071748879
```

2 Logistic Regression¶

In [44]:

```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(X_train_df,Y_train)
```

Out[44]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [45]:

```
prediction["Logistic"] = model.predict(X_test_df)
```

In [47]:

```
accuracy_score(Y_test,prediction["Logistic"])
```

Out[47]:

```
0.9802690582959641
```

3 k -NN classifier¶

In [49]:

```
from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier(n_neighbors=5)
model.fit(X_train_df,Y_train)
```

Out[49]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

In [50]:

```
prediction['knn']=model.predict(X_text_df)
```

In [51]:

```
accuracy_score(Y_test,prediction["knn"])
```

Out[51]:

```
0.9121076233183857
```

4 Ensemble classifier¶

In [52]:

```
from sklearn.ensemble import RandomForestClassifier

model=RandomForestClassifier()

model.fit(X_train_df,Y_train)
```

Out[52]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [53]:

```
prediction["random_forest"] = model.predict(X_test_df)
```

In [55]:

```
accuracy_score(Y_test,prediction["random_forest"])
```

Out[55]:

```
0.9695067264573991
```

In [57]:

```
from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier()

model.fit(X_train_df,Y_train)
```

Out[57]:

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None)
```

In [58]:

```
prediction["adaboost"] = model.predict(X_test_df)
```

In [60]:

```
accuracy_score(Y_test,prediction["adaboost"])
```

Out[60]:

```
0.967713004484305
```

ParameterTuning using GridSearchCV¶

Based, on the above four ML models, Naive Bayes has given the best accuracy. However, Let's try to tune the parameters of k -NN using GridSearchCV

In [62]:

```
from sklearn.model_selection import GridSearchCV

k_range=np.arange(1,30)

k_range
```

Out[62]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

In [63]:

```
param_grid=dict(n_neighbors=k_range)

print(param_grid)
```

```
{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])}
```

In [64]:

```
model=KNeighborsClassifier()

grid=GridSearchCV(model,param_grid)

grid.fit(X_train_df,Y_train)
```

Out[64]:

```
GridSearchCV(cv=None, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
i',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 1
2, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [66]:

```
grid.best_estimator_
```

Out[66]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

In [67]:

```
grid.best_params_
```

Out[67]:

```
{'n_neighbors': 1}
```

In [68]:

```
grid.best_score_
```

Out[68]:

```
0.9461521202602647
```

In [69]:

```
grid.grid_scores_
```

```
C:\Users\ADMIN\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761: Dep
recationWarning: The grid_scores_ attribute was deprecated in version 0.18 in favor of
the more elaborate cv_results_ attribute. The grid_scores_ attribute will not be avail
able from 0.20
  DeprecationWarning)
```

Out[69]:

```
[mean: 0.94615, std: 0.00449, params: {'n_neighbors': 1},
 mean: 0.92259, std: 0.00288, params: {'n_neighbors': 2},
 mean: 0.92349, std: 0.00226, params: {'n_neighbors': 3},
 mean: 0.90554, std: 0.00117, params: {'n_neighbors': 4},
 mean: 0.90621, std: 0.00065, params: {'n_neighbors': 5},
 mean: 0.89410, std: 0.00060, params: {'n_neighbors': 6},
 mean: 0.89455, std: 0.00062, params: {'n_neighbors': 7},
 mean: 0.88580, std: 0.00141, params: {'n_neighbors': 8},
 mean: 0.88602, std: 0.00142, params: {'n_neighbors': 9},
 mean: 0.88198, std: 0.00262, params: {'n_neighbors': 10},
 mean: 0.88198, std: 0.00262, params: {'n_neighbors': 11},
 mean: 0.87660, std: 0.00210, params: {'n_neighbors': 12},
 mean: 0.87705, std: 0.00230, params: {'n_neighbors': 13},
 mean: 0.87256, std: 0.00223, params: {'n_neighbors': 14},
 mean: 0.87278, std: 0.00253, params: {'n_neighbors': 15},
 mean: 0.87009, std: 0.00051, params: {'n_neighbors': 16},
 mean: 0.87009, std: 0.00051, params: {'n_neighbors': 17},
 mean: 0.86830, std: 0.00028, params: {'n_neighbors': 18},
 mean: 0.86852, std: 0.00030, params: {'n_neighbors': 19},
 mean: 0.86718, std: 0.00110, params: {'n_neighbors': 20},
 mean: 0.86718, std: 0.00110, params: {'n_neighbors': 21},
 mean: 0.86650, std: 0.00059, params: {'n_neighbors': 22},
 mean: 0.86650, std: 0.00059, params: {'n_neighbors': 23},
 mean: 0.86650, std: 0.00059, params: {'n_neighbors': 24},
 mean: 0.86650, std: 0.00059, params: {'n_neighbors': 25},
 mean: 0.86650, std: 0.00059, params: {'n_neighbors': 26},
 mean: 0.86650, std: 0.00059, params: {'n_neighbors': 27},
 mean: 0.86605, std: 0.00004, params: {'n_neighbors': 28},
 mean: 0.86605, std: 0.00004, params: {'n_neighbors': 29}]
```

Model Evaluation¶

In [71]:

```
print(classification_report(Y_test,prediction['multinomial'],target_names=['Ham','Spam
']))
```

```
             precision    recall  f1-score   support

        Ham       0.99      0.99      0.99       965
       Spam       0.97      0.95      0.96       150

avg / total       0.99      0.99      0.99      1115
```

In [72]:

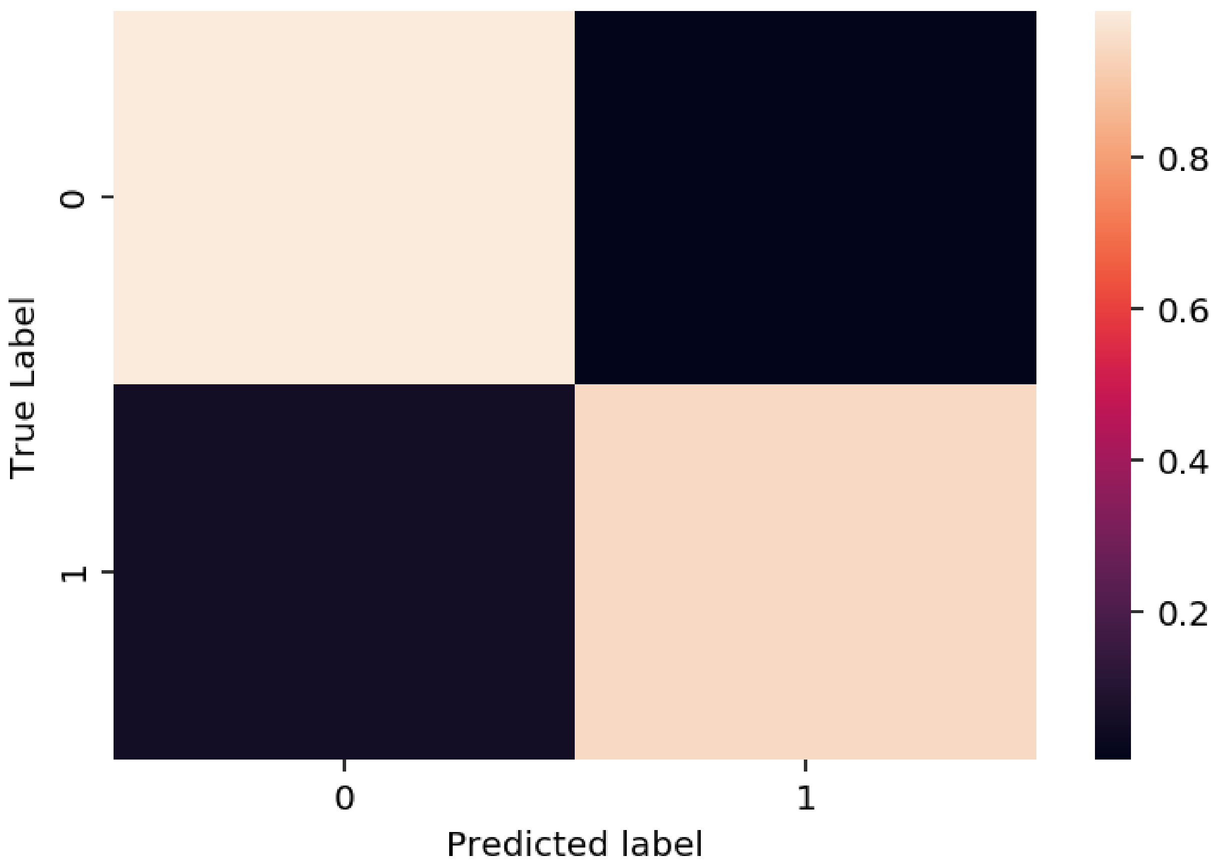
```
cm=confusion_matrix(Y_test,prediction['multinomial'])
cm_normalized=cm.astype('float')/cm.sum(axis=1)[:,np.newaxis]
```

In [73]:

```
sns.heatmap(cm_normalized)
plt.ylabel('True Label')
plt.xlabel('Predicted label')
```

Out[73]:

```
Text(0.5,15,'Predicted label')
```



In [74]:

```
print(cm)
```

```
[[960   5]
 [  8 142]]
```

By seeing the above confusion matrix, it is clear that 5 Ham are mis classified as Spam, and 8 Spam are misclassified as Ham. Let'see what are those misclassified text messages. Looking those messages may help us to come up with more advanced feature engineering.

In [75]:

```
pd.set_option('display.max_colwidth',-1)
```

I increased the pandas dataframe width to display the misclassified texts in full width.

Misclassified as Spam¶

In [77]:

```
X_test[Y_test < prediction["multinomial"] ]
```

Out[77]:

```
573     Waiting for your call.
4727    I (Career Tel) have added u as a contact on INDYAROCKS.COM to send FREE SMS. T
o remove from phonebook - sms NO to  &lt;#&gt;
5475    Dhoni have luck to win some big title.so we will win:)
4860    Nokia phone is lovly..
1259    We have sent JD for Customer Service cum Accounts Executive to ur mail id, For
details contact us
Name: text, dtype: object
```

Misclassified as Ham¶

In [78]:

```
X_test[Y_test > prediction["multinomial"] ]
```

Out[78]:

```
5035    You won't believe it but it's true. It's Incredible Txts! Reply G now to learn
truly amazing things that will blow your mind. From O2FWD only 18p/txt
2574    Your next amazing xxx PICSFREE1 video will be sent to you enjoy! If one vid is
not enough for 2day text back the keyword PICSFREE1 to get the next video.
3130    LookAtMe!: Thanks for your purchase of a video clip from LookAtMe!, you've bee
n charged 35p. Think you can do better? Why not send a video in a MMSto 32323.
68      Did you hear about the new \Divorce Barbie\"? It comes with all of Ken's stuff
!"
2662    Hello darling how are you today? I would love to have a chat, why dont you tel
l me what you look like and what you are in to sexy?
4211    Missed call alert. These numbers called but left no message. 07008009200
3572    You won't believe it but it's true. It's Incredible Txts! Reply G now to learn
truly amazing things that will blow your mind. From O2FWD only 18p/txt
3979    ringtoneking 84484
Name: text, dtype: object
```

It seems length of the spam text is much higher than the ham. Maybe we can include length as a feature. In addition to unigram, we can also try bigram features