Text Classification Using Naive Bayes

Neba Nfonsang
University of Denver

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import re
import html
from nltk.corpus import stopwords
import string

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline
```

Load the Data

In [2]:
```
data = pd.read_csv("spam_text_messages.csv")
data.head()
```

Out[2]:

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

Data Preprocessing

In [3]:
```
# initialize stopwords
sw = set(stopwords.words("english"))
list(sw)[0:10] # view the first 10 stopwords
```

Out[3]:
```
['why',
 'because',
 'am',
 'hers',
 'if',
 'through',
 'ain',
 "weren't",
 'down',
 'themselves']
```

In [4]:
```
# view puntuations and special characters that need to be removed
print(set(string.punctuation))
```

```
{'!', ',', '\\', '/', '%', '"', '`', ':', '^', ')', '?', ']', '~', '>', '@', '&',
'{', '[', '_', '*', '-', ';', '$', '#', '|', '}', '(', '+', '=', "'", '<', '.'}
```

In [5]:

```
# a function that cleans text and removes stop words


def clean(text, stopwords):
    # remove tags like <tab>
    text = re.sub(r'<[^<>]*>', ' ', text)
    # split text on whitespace
    text_list = text.split()
    text_words = []

    punctuation = set(string.punctuation)

    # keep #tags and @mentions
    ## punctuation.remove("#")
    ## punctuation.remove("@")

    for word in text_list:
        # remove punctuation marks at the beginning
        # of each word
        while len(word) > 0 and word[0] in punctuation:
            word = word[1:]

        # remove punctuation marks at the end of each word
        while len(word) > 0 and word[-1] in punctuation:
            word = word[:-1]

        # a rule to eliminate most urls
        if len(word) > 0 and "/" not in word:
            # eliminate stopwords
            if word.lower() not in stopwords:
                # append the word to the text_words list
                text_words.append(word.lower())
        cleaner_text = " ".join(text_words)
    return cleaner_text
```

In [6]:
```
# let's check how one of the messages look like before cleaning
data["Message"][305]
```

Out[6]:
```
'SMS. ac Blind Date 4U!: Rodds1 is 21/m from Aberdeen, United Kingdom. Check Him out http://img. (http://img.) sms. ac/W/icmb3cktz8r7!-4 no Blind Dates send HIDE'
```

In [7]:
```
# Apply the clean() function to the data and pass in the stopword argument

data["Message"] = data["Message"].apply(clean, stopwords=sw)
data.head()
```

Out[7]:

	Category	Message
0	ham	go jurong point crazy available bugis n great ...
1	ham	ok lar joking wif u oni
2	spam	free entry 2 wkly comp win fa cup final tkts 2...
3	ham	u dun say early hor u c already say
4	ham	nah think goes usf lives around though

In [8]:
```
#checking text after cleaning
data["Message"][305]
```

Out[8]:
```
'sms ac blind date 4u rodds1 aberdeen united kingdom check sms blind dates send hide'
```

In [9]:
```
# check the shape of the data
data.shape
```

Out[9]:
```
(5572, 2)
```

In [10]:
```
# select only text with more than 50 words for training
data = data[data["Message"].str.len() > 50]
data.shape
```

Out[10]:
```
(2251, 2)
```

In [11]:
```
data.head()
```

Out[11]:

	Category	Message
0	ham	go jurong point crazy available bugis n great ...
2	spam	free entry 2 wkly comp win fa cup final tkts 2...
5	spam	freemsg hey darling 3 week's word back i'd lik...
7	ham	per request melle melle oru minnaminunginte nu...
8	spam	winner valued network customer selected receiv...

Split the Data

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(data["Message"],
                                                    data["Category"],
                                                    test_size=0.3)

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

```
(1575,) (1575,)
(676,) (676,)
```

Transform Text Data to Feature Vectors

In [13]:
```
tfidf = TfidfVectorizer(ngram_range=(1,2), stop_words="english", min_df=10,
                        max_features=None)
X_train_tfidf= tfidf.fit_transform(X_train)
X_train_tfidf.toarray()
```

Out[13]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [14]:
```
X_train_tfidf.toarray().shape
```

Out[14]:
```
(1575, 464)
```

Train the Model

Note that there are several types of Naive Bayes constructors dependent on the nature of the data:

- MultinomialNB() is used for text classification when data is represented as a feature vector.
- ComplementNB() is an adaptation of the standard MultinomialNB() for imbalance data.
- GaussianNB() is used if the features are assumed numerical and are assumed to follow a Gaussian or normal distribution.
- BernoulliNB() is used when each feature follows a Bernoulli distribution. That is, the data or all features are binary with values 0 or 1.
- CategoricalNB() is used when each feature has it's own categorical distribution.

In [15]:
```
# construct and fit model
clf = MultinomialNB()
clf = clf.fit(X_train_tfidf, y_train)
```

In [16]:
```
# make prediction on training set
clf.predict(X_train_tfidf)
```

Out[16]:
```
array(['ham', 'ham', 'ham', ..., 'ham', 'ham', 'ham'], dtype='<U4')
```

In [17]:
```
# compute accuracy on training set
clf.score(X_train_tfidf, y_train)
```

Out[17]:
```
0.9701587301587301
```

Model Evaluation

In [18]:
```
X_test_tfidf= tfidf.transform(X_test)
X_test_tfidf.toarray()
```

Out[18]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [19]:
```
clf.score(X_test_tfidf, y_test)
```

Out[19]:
```
0.9497041420118343
```

Cross Validation

In [20]:
```
tfidf = TfidfVectorizer(ngram_range=(1,2), stop_words="english", min_df=10,
                        max_features=None)
X_train_tfidf= tfidf.fit_transform(X_train)

scores = cross_val_score(estimator=MultinomialNB(), X=X_train_tfidf, y=y_train, cv=5)
print("Average cross validation score: ", scores.mean())
print("Standard deviation of cross validation scores: ", scores.std())
```

```
Average cross validation score:  0.9612698412698413
Standard deviation of cross validation scores:  0.010509806576029759
```

Grid Search Cross Validation

In [21]:
```
# check hyperparameters that can be optimized
tfidf.get_params()
```

Out[21]:
```
{'analyzer': 'word',
 'binary': False,
 'decode_error': 'strict',
 'dtype': numpy.float64,
 'encoding': 'utf-8',
 'input': 'content',
 'lowercase': True,
 'max_df': 1.0,
 'max_features': None,
 'min_df': 10,
 'ngram_range': (1, 2),
 'norm': 'l2',
 'preprocessor': None,
 'smooth_idf': True,
 'stop_words': 'english',
 'strip_accents': None,
 'sublinear_tf': False,
 'token_pattern': '(?u)\\b\\w\\w+\\b',
 'tokenizer': None,
 'use_idf': True,
 'vocabulary': None}
```

In [22]:
```
pipe = Pipeline([("tfidf",TfidfVectorizer(stop_words="english")),
                 ("nb", MultinomialNB())])


param_grid = [{"tfidf__min_df":[5, 20],
               "tfidf__ngram_range":[(1, 1), (1, 2), (1, 3), (1, 5), (1, 7)]}]

grid = GridSearchCV(estimator=pipe , param_grid =param_grid, cv=5)
grid.fit(X_train, y_train)
```

Out[22]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('tfidf',
                                        TfidfVectorizer(stop_words='english')),
                                       ('nb', MultinomialNB())]),
             param_grid=[{'tfidf__min_df': [5, 20],
                          'tfidf__ngram_range': [(1, 1), (1, 2), (1, 3), (1, 5),
                                                 (1, 7)]}])
```

In [23]:
```
# find best hyperparameter values
grid.best_params_
```

Out[23]:
```
{'tfidf__min_df': 5, 'tfidf__ngram_range': (1, 2)}
```

In [24]:
```
# training accuracy
grid.score(X_train, y_train)
```

Out[24]:
```
0.9752380952380952
```

In [25]:
```
# test accuracy
grid.score(X_test, y_test)
```

Out[25]:
```
0.9630177514792899
```

Note: Combining Text from Different Columns

- Sometimes, if your text data has several columns with meaningful text, you may combine the columns with meaningful text into a single column of text.

In [26]:
```
articles = ["""Deterministic point inclusion methods for
             computational applications with complex geometry""",
             """Terascale direct numerical simulations of turbulent
             combustion using S3D"""
            ]


abstracts = ["""A fundamental problem in computation is finding practical
             and efficient algorithms for determining if a query point is
             contained within a model of a three-dimensional solid""",
            """
            Computational science is paramount to the understanding of
            underlying processes in internal combustion engines of the
            future that will utilize non-petroleum-based alternative fuels,
            """
            ]

df = pd.DataFrame(zip(articles, abstracts),
                  columns=["articles", "abstracts"])
df
```

Out[26]:

	articles	abstracts
0	Deterministic point inclusion methods for \n ...	A fundamental problem in computation is findin...
1	Terascale direct numerical simulations of turb...	\n Computational science is paramoun...

In [27]:
```
# combine articles and abstracts into a single text

df["text"] = df["articles"] + " " + df["abstracts"]
df.drop(["articles", "abstracts"], axis="columns")
```

Out[27]:

	text
0	Deterministic point inclusion methods for \n ...
1	Terascale direct numerical simulations of turb...

In [28]:
```
# apply the clean function to a single case
clean(df["text"][0], sw)
```

Out[28]:
```
'deterministic point inclusion methods computational applications complex geometry fun
damental problem computation finding practical efficient algorithms determining query
point contained within model three-dimensional solid'
```