View in Colaboratory (https://colab.research.google.com/github/wikiabhi/Text-Classification/blob/master/Text_Classification.ipynb)

In [0]:
```
import numpy as np
import pandas as pd
import os
import time
```

In [2]:
```
start_time = time.time()
start_time
```

Out[2]:
```
1528366468.3325632
```

In [3]:
```
import nltk
nltk.download('stopwords')
```

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

Out[3]:
```
True
```

In [0]:
```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

In [0]:
```
# Our own list of some block words to be avoided
block_words = ['newsgroups', 'xref', 'path', 'from', 'subject', 'sende
```

In [6]:
```
## Download the dataset
import urllib.request
urllib.request.urlretrieve ("https://archive.ics.uci.edu/ml/machine-le
```

Out[6]:
```
('a.tar.gz', <http.client.HTTPMessage at 0x7f361d78eac8>)
```

In [0]:
```
#Extracting the dataset
import tarfile
tar = tarfile.open("a.tar.gz")
tar.extractall()
tar.close()
```

In [8]:
```
##Make a list of the folders in the dataset
directory = [f for f in os.listdir('./20_newsgroups') if not f.startsw
directory
```

Out[8]:
```
['misc.forsale',
 'sci.med',
 'talk.politics.guns',
 'alt.atheism',
 'talk.politics.mideast',
 'soc.religion.christian',
 'comp.os.ms-windows.misc',
 'rec.autos',
 'sci.crypt',
 'comp.sys.ibm.pc.hardware',
 'sci.electronics',
 'comp.graphics',
 'rec.sport.baseball',
 'rec.motorcycles',
 'talk.politics.misc',
 'comp.sys.mac.hardware',
 'sci.space',
 'talk.religion.misc',
 'rec.sport.hockey',
 'comp.windows.x']
```

In [0]:
```
# Create a dictionary of words with their frequency
vocab = {}
for i in range(len(directory)):
    ##Create a list of files in the given dictionary
    files = os.listdir('./20_newsgroups/' + directory[i])

    for j in range(len(files)):
        ##Path of each file
        path = './20_newsgroups/' + directory[i] + '/' + files[j]

        ##open the file and read it
        text = open(path, 'r', errors='ignore').read()

        for word in text.split():
            if len(word) != 1:
                ##Check if word is a non stop word or non block word(w
                if not word.lower() in stop_words:
                    if not word.lower() in block_words:
                        ##If word is already in dictionary then we jus
                        if vocab.get(word.lower()) != None:
                            vocab[word.lower()] += 1

                        ##If word is not in dictionary then we put tha
                        else:
                            vocab[word.lower()] = 1

# vocab
```

In [0]:
```
import operator
sorted_vocab = sorted(vocab.items(), key= operator.itemgetter(1), reve
# sorted_vocab
```

In [0]:
```
# Dictionary containing the most occuring k-words.
kvocab={}

# Frequency of 1000th most occured word
z = sorted_vocab[2000][1]

for x in sorted_vocab:
    kvocab[x[0]] = x[1]

    if x[1] <= z:
        break
```

In [12]:
```
#@title
sorted_vocab[0:100]
```

Out[12]:
```
[('subject:', 20486),
 ('from:', 20417),
 ('date:', 20137),
 ('newsgroups:', 20081),
 ('message-id:', 20050),
 ('lines:', 20042),
 ('path:', 20029),
 ('article', 12108),
 ('people', 8415),
 ('university', 8203),
 ('know', 7695),
 ('think', 7205),
 ("i'm", 5823),
 ('distribution:', 4406),
 ('time', 4336),
 ('it.', 4185),
 ('anyone', 3976),
 ('world', 3602),
 ('right', 3326),
 ('believe', 3309),
 ('still', 3290),
 ('something', 3190),
 ('computer', 3157),
 ('system', 3137),
 ("i've", 3114),
 ('15', 2881),
 ('god', 2881),
 ('back', 2840),
 ('news', 2836),
 ("can't", 2836),
 ('state', 2787),
 ('work', 2692),
 ('someone', 2610),
 ('>in', 2610)
```

```
('>in', 2610),
('government', 2534),
('problem', 2528),
('23', 2522),
('another', 2516),
('read', 2516),
('usa', 2496),
('information', 2480),
('>the', 2452),
('number', 2424),
("that's", 2382),
('things', 2378),
('part', 2323),
('fri,', 2307),
('point', 2297),
('little', 2294),
('22', 2284),
('windows', 2265),
('>i', 2253),
('tue,', 2241),
('file', 2208),
('data', 2155),
('question', 2126),
('probably', 2112),
('years', 2106),
('different', 2100),
('available', 2095),
('(usenet', 2079),
('space', 2079),
('it,', 2073),
('around', 2072),
('long', 2053),
('tell', 2048),
('least', 2006),
('best', 1997),
('program', 1995),
('software', 1976),
('public', 1961),
('power', 1958),
('thu,', 1883),
('thing', 1875),
('drive', 1870),
('run', 1869),
('support', 1864),
('however,', 1826),
("i'd", 1825),
('18', 1804),
('rather', 1801),
('enough', 1792),
('case', 1791),
('hard', 1786),
('keep', 1770),
('fact', 1767),
('25', 1758),
```

```
('let', 1757),
('science', 1753),

('called', 1751),
('great', 1742),
('...', 1738),
('call', 1725),
('looking', 1709),
('mon,', 1690),
('found', 1683),
('real', 1676),
('nothing', 1671),
('26', 1661),
('quite', 1634)]
```

In [0]:
```
features_list = list(kvocab.keys())

## Create a Dataframe containing features_list as columns
df = pd.DataFrame(columns = features_list)


## Filling the x_train values in dataframe

for i in range(len(directory)):
    ##Create a list of files in the given dictionary
    files = os.listdir('./20_newsgroups/' + directory[i])

    for j in range(len(files)):
        ##Insert a row at the end of Dataframe with all zeros
        df.loc[len(df)] = np.zeros(len(features_list))

        ##Path of each file
        path = './20_newsgroups/' + directory[i] + '/' + files[j]

        ##open the file and read it
        text = open(path, 'r', errors='ignore').read()


        for word in text.split():
            if word.lower() in features_list:
                df[word.lower()][len(df)-1] += 1


# df.head()
```

In [0]:
```
## Making the 2d array of x
x = df.values

## Feature list
f_list = list(df)
```

In [15]:
```
## Creating  y array containing labels for classification

y = []

for i in range(len(directory)):
    ##Create a list of files in the given dictionary
    files = os.listdir('./20_newsgroups/' + directory[i])

    for j in range(len(files)):
        y.append(i)

y = np.array(y)
y.shape
```

Out[15]:
```
(19997,)
```

In [0]:
```
## Splitting the whole dataset for training and testing
from sklearn import model_selection
x_train, x_test, y_train, y_test = model_selection.train_test_split(x,
```

Implement Multinomial Naive Bayes from sklearn

In [17]:
```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)

train_score = clf.score(x_train, y_train)
test_score = clf.score(x_test, y_test)

train_score, test_score
```

Out[17]:
```
(0.8797759551910382, 0.834)
```

In [18]:
```
## Calculating time on our local machine
end_time = time.time()
total_time = end_time - start_time
total_time
```

Out[18]:
```
1700.6345119476318
```

Implementing Multinomial Naive Bayes from scratch

In [0]:
```
def fit(x_train, y_train):

    ## dictionary containing the count of words
    count = {}

    ## set of all classes
    set_class = set(y_train)

    for current_class in set_class:
        count[current_class] = {}
        count["total_data"] = len(y_train)

        ##Rows whose class is current_class
        current_class_rows = (y_train == current_class)

        x_train_current = x_train[current_class_rows]
        y_train_current = y_train[current_class_rows]

        sums = 0
        for i in range(len(f_list)):
            ## For each class, calculating total frequency of a featu
            count[current_class][f_list[i]] = x_train_current[:,i].sum
            sums = sums + count[current_class][f_list[i]]

        ##Calculating total count of words of a class
        count[current_class]["total_count"] = sums

    return count
```

In [0]:
```
def probability(dictionary, row, current_class):
    ## class_prob = log of probability of the current class = log(no c
    class_prob = np.log(dictionary[current_class]["total_count"]) - np
    total_prob = class_prob


    for i in range(len(row)):
        ##Numerator
        word_count = dictionary[current_class][f_list[i]] + 1
        ## Denominator
        total_count = dictionary[current_class]["total_count"] + len(f
        ## Add 1 to numerator and len(row) in denominator for laplace

        ## Log Probabilty of a word
        word_prob = np.log(word_count) - np.log(total_count)

        ##Calculating probability frequency number of times
        for j in range(int(row[i])):
            total_prob += word_prob

    return total_prob
```

In [0]:
```
ef predictSinglePoint(row, dictionary):
    classes = dictionary.keys()

    ##Initialising best_prob and best_class as very low count

    best_prob = -1000
    best_class = -1
    first_iter = True

    for current_class in classes:
        if(current_class == "total_data"):
            continue

        ##Calculating probabilty that the given row belong to current_cl
        prob_current_class = probability(dictionary, row, current_class)

        ##For first iteration we set the best_prob to be the probabilty
        ##For rest iteration, we check if the probabilty that row is of
        if(first_iter or prob_current_class > best_prob):
            best_prob = prob_current_class
            best_class = current_class

        first_iter = False

    ## Return the best class which has maximum probabilty.
    return best_class
```

In [0]:
```
def predict(x_test, dictionary):
    ## Initialise a list which contain the predictions
    y_pred_self = []

    ##Iterate through each row in x_test
    for j in range(len(x_test)):

        ##Calculate the prediction of the class to which the row belo
        pred_class = predictSinglePoint(x_test[j,:], dictionary)

        ##Append the predicted class to our list
        y_pred_self.append(pred_class)

    ##Return the list of predictions
    return y_pred_self
```

In [0]:
```
## Training the model
dictionary = fit(x_train, y_train)

##Testing the model
y_pred_self = predict(x_test, dictionary)
```

Accuracy Comparison between Sklearn MultinomialNB() and self implementation

In [0]:
```
from sklearn.metrics import accuracy_score
print("Accuracy for sklearn MultinomialNB() - ", test_score)
print("Accuracy for self-implemented Naive Bayes - ", accuracy_score(y
```

```
Accuracy for sklearn MultinomialNB() -  0.8394
Accuracy for self implemented Naive Bayes -  0.8422
```

Classification Report Comparison between Sklearn MultinomialNB() and self implementation

In [0]:
```
from sklearn.metrics import classification_report
print("Classification report for sklearn MultinomialNB()",classificati
print("Classification report for self-implemented Naive Bayes ",classi
```

```
Classification report for sklearn MultinomialNB()              preci
sion    recall  f1-score   support

          0       0.95      0.78      0.85       233
          1       0.83      0.90      0.86       253
          2       0.87      0.90      0.88       249
          3       0.69      0.77      0.73       240
          4       0.91      0.94      0.93       236
          5       0.80      0.88      0.84       240
          6       0.94      0.95      0.95       261
          7       0.96      0.89      0.93       269
          8       0.95      0.87      0.90       284
          9       0.72      0.61      0.66       248
         10       0.82      0.94      0.87       231
         11       0.93      0.82      0.87       233
         12       0.79      0.76      0.78       244
         13       0.89      0.82      0.86       256
         14       0.80      0.88      0.84       246
         15       0.70      0.86      0.77       253
         16       0.89      0.87      0.88       248
         17       0.96      1.00      0.98       281
         18       0.75      0.87      0.81       259
         19       0.62      0.42      0.50       236

avg / total       0.84      0.84      0.84      5000

Classification report for self implemented Naive Bayes
precision    recall  f1-score   support

          0       0.95      0.79      0.86       233
          1       0.85      0.90      0.87       253
```

2	0.87	0.90	0.89	249
3	0.69	0.78	0.73	240
4	0.93	0.94	0.93	236
5	0.80	0.87	0.83	240
6	0.94	0.95	0.94	261
7	0.96	0.90	0.93	269
8	0.94	0.87	0.91	284
9	0.72	0.62	0.67	248
10	0.83	0.94	0.88	231
11	0.92	0.84	0.88	233
12	0.79	0.77	0.78	244
13	0.89	0.83	0.86	256
14	0.81	0.88	0.84	246
15	0.70	0.86	0.77	253
16	0.89	0.87	0.88	248
17	0.96	1.00	0.98	281
18	0.77	0.86	0.81	259
19	0.62	0.43	0.51	236
avg / total	0.84	0.84	0.84	5000