TEXT CLASSIFICATION NAÏVE BAYES

NAÏVE BAYES TEXT CLASSIFIRS INTRODUCTION:

NAÏVE BAYES CLASSIFIER:

The Naive Bayes classifier is a simple probabilistic classifier which is based on Bayes theorem with strong and naïve independence assumptions.

TEXT CLASSIFICATION:

*Input*:

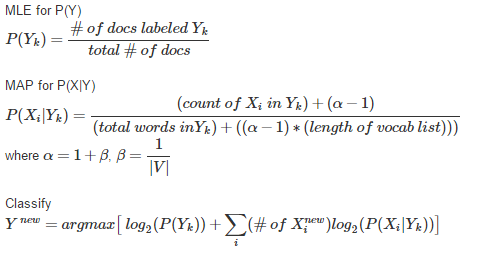
a document *d*

a fixed set of classes *C* ={*c*1, *c*2,…, *cJ*}

*Output*:

a predicted class *c* ∈ *C*

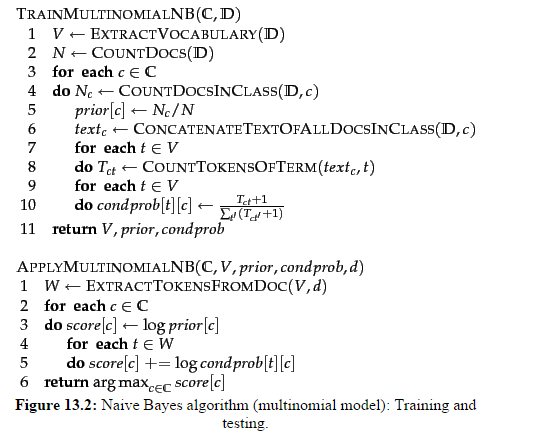
Hence with the help of training data I must calculate the MLE, MAP on the training data and classify on the testing data.



ALGORITHIM FOR NAÏVE BAYES:

With the help of Stanford nlp tutorial I got the idea of algorithmic approach of training and testing the data for Naïve Bayes Classification. But they have used Laplace Smoothing but I Calculated β value using the above formula

β=1/|Vocabulary|



HIGH LEVEL CODE DESCRIPTION:

**Python Code: np11.py** (NAÏVE BAYES IMPLEMENTATION,ACCURACY, CONFUSION MATRIX)

Usage of numpy and scipy python packages helped me to complete the project successfully as I initially started with traditional for loops which took lot of time to execute and then understood the usage of Indexing and Numpy arrays my program is now running fast.

1. Read all the dataset using “np.genfromtxt” to read the train.data, test.data, train.label, test.label, vocabulary.text.
2. We take the train data file and train label file to calculate the MLE and MAP.
3. From the Train.label file as line number indicated the doc id and the label we calculate the MLP using the above formula number of word in given class/total number of documents and store a matrix of 20X1 Matrix of MLE .Hence MLE of each label of newsgroup.
4. Using the above mentioned Formula calculated the MAP for each label hence the MAP Matrix follows(20X61188)[(class labels X Word Id)](β=1/|v|).
5. Then creating a sparse matrix to performing Classification on the Test data using MLE and MAP Matrix.
6. Using the Above formula to classify I performed the operations

In the following code testdatamatrix\_transpose matrix is my sparse matrix of (7505X61188)

testdatamatrix\_transpose=testdatamatrix.transpose()  
testdatacalc=maplog2\*testdatamatrix\_transpose  
testdatacalc=testdatacalc+np.matlib.repmat(mle,1,7505)  
argmaxval=(testdatacalc.argmax(axis=0)+1)  
argmaxval = argmaxval.transpose()

1. From the argmax value and then I can calculate the Confusion Matrix and with the help of confusionmatrix I can easily find out the accuracy(% of correctly classified).

accuracy=sum(confusionmatrix.diagonal())/float(7505)  
accuracy=accuracy\*100

**Python Code: nb\_plot\_different\_beta values.py**

(DIIFFERNT BETA VALUE CHANGE FROM .00001 to 1)

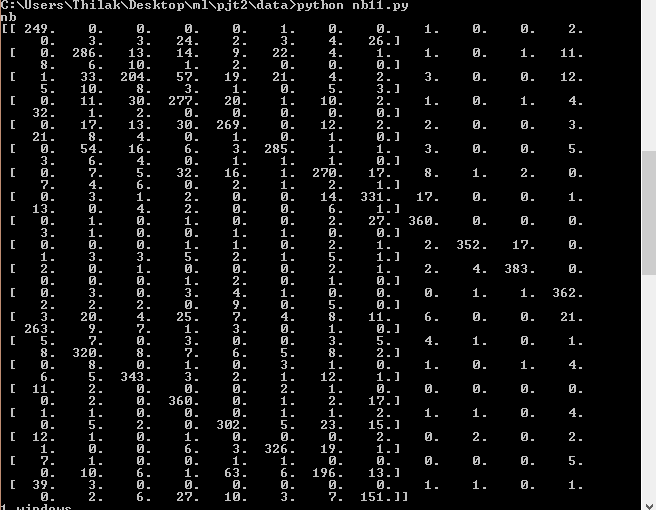
This code is similar to the code description of nb11.py.

The difference is I am looping the code for different β value ranging from .00001 to 1

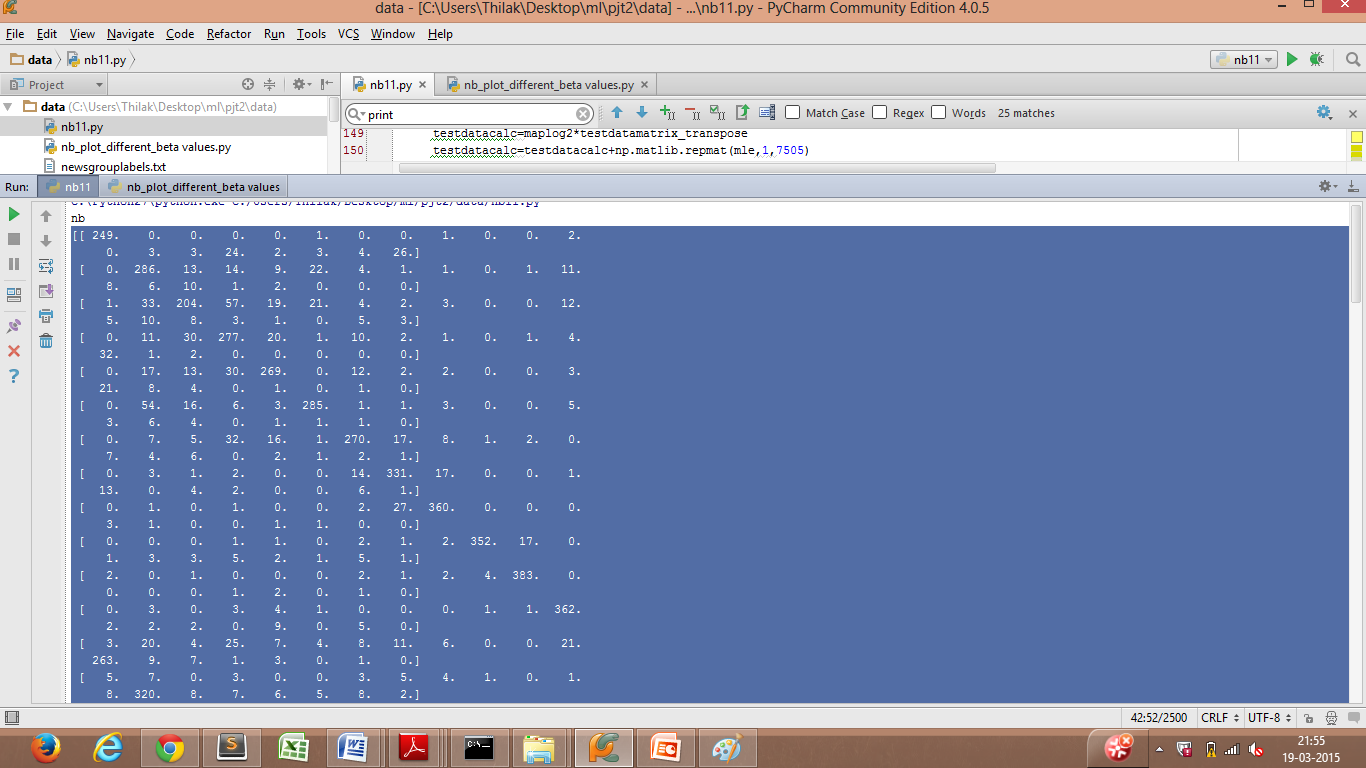
**Python Code: nb\_100words.py**

Print 100 words with Highest Measure

**CONFUSION MATRIX:**

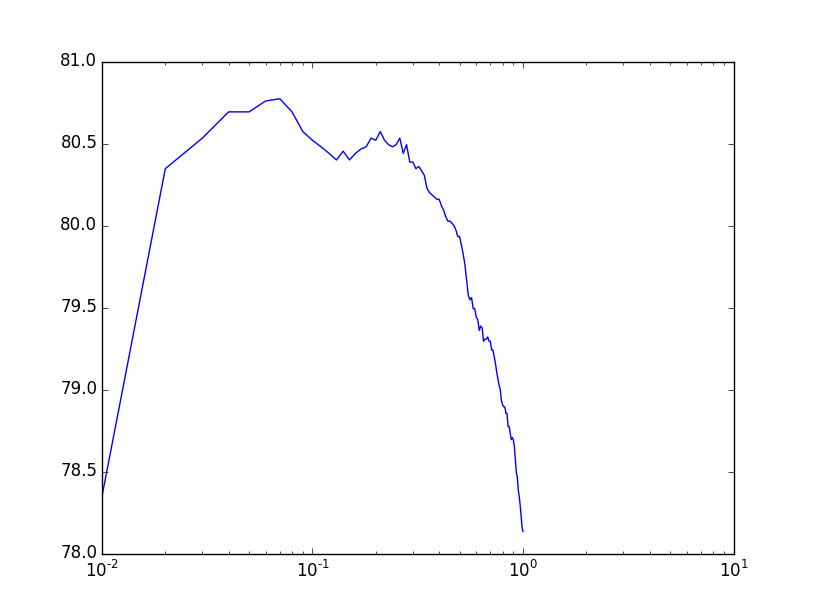
****

In PyCharm Ide the confusion Matrix is

****

With different Values of β ranging from 0.0001 to 1(Laplace Smoothing). I get varied Accuracy Results.The Semilogx graph is followed.( **nb\_plot\_different\_beta values.py** code does this**)**

Lower β values tends to Over fitting of testing data and Higher β value tends to Under fitting.



.

Following are the array print of each β value and its accuracy.

β value:

[0.01001, 0.02001, 0.030010000000000002, 0.040010000000000004, 0.050010000000000006, 0.06001000000000001, . . . , 0.9900100000000007, 1.0000100000000007]

Accuracy:

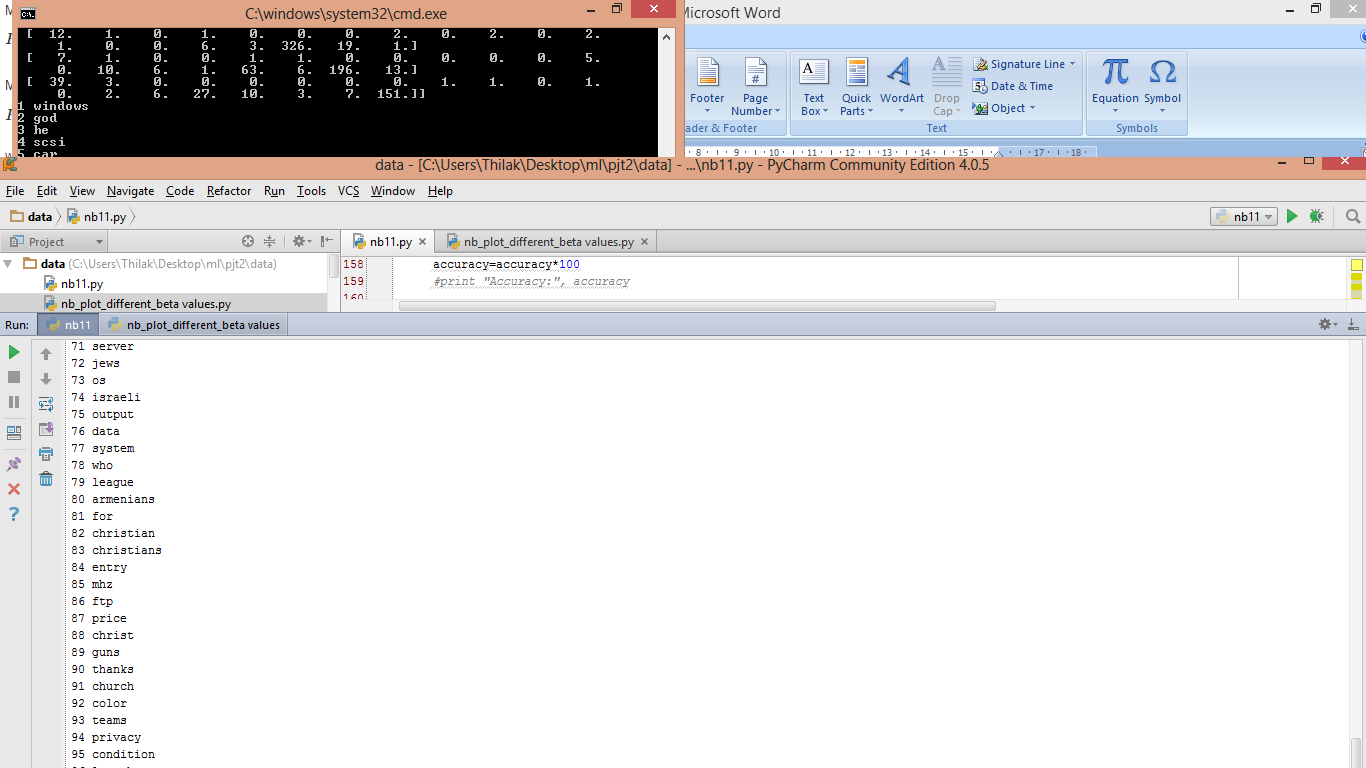
[78.347768154563624, 80.346435709526986, 80.532978014656891, 80.692871419053972, 80.692871419053972, . . . , 78.16122584943372, 78.134576948700868]

Entire List is pasted in Appendix

A Probabilistic method to rank the words for the classier we calculate the Likelihood P(WORD/LABEL) which can also denotes Term Frequency. Here the Posterior is P(Label/Word) is the Likelihood and Prior and thus Using the **Information Gain Knowledge** Ixy=H(X)-H(X/Y) that is Calculation of H(X) and H(Y) using P(Label/Word)\*log2P((Label/Word)) and 1- P(Label/Word)\*log2(1-P((Label/Word)) and thus using it from Michel Book.

Another way is using through **TF-IDF Calculation used using SCIKIT Python learning package we can get the TFIDF Vector to determine the top 100 words**.

**nb\_100words.py** codedoes the same logic of the probabilistic approach of Information gain concept and also through TFIDF Calculation to get the top 100 words rank for the classifier.





The entire list is in Appendix.

REFERNCES:

CODE:

1. http://nlp.stanford.edu/IR-book/html/htmledition/text-classification-and-naive-bayes-1.html

2. www.docs.spicy.org, for referencing the python syntaxes

3. Referenced Piazza codes, discussions and formula for implementing MAP and Categorize it.

4. http://blog.christianperone.com/?p=1747 to understand TFIDF.

5.p36.m to understand the conept of Indexing for faster implementation of Matrix Calculations.

DOCUMENT:

1. http://nlp.stanford.edu/IR-book/html/htmledition/text-classification-and-naive-bayes-1.html
2. <https://www.cs.cmu.edu/~tom/10701_sp11>

APPENDIX:

DIFFERENT BETA VALUEA ND ITS RESPECTIVE ACCURACY:

[78.347768154563624, 80.346435709526986, 80.532978014656891, 80.692871419053972, 80.692871419053972, 80.759493670886073, 80.772818121252499, 80.692871419053972, 80.572951365756168, 80.519653564290479, 80.479680213191202, 80.439706862091938, 80.399733510992675, 80.45303131245835, 80.399733510992675, 80.439706862091938, 80.466355762824776, 80.479680213191202, 80.532978014656891, 80.519653564290479, 80.572951365756168, 80.519653564290479, 80.493004663557628, 80.479680213191202, 80.493004663557628, 80.532978014656891, 80.439706862091938, 80.493004663557628, 80.386409060626249, 80.386409060626249, 80.346435709526986, 80.359760159893398, 80.33311125916056, 80.306462358427709, 80.226515656229182, 80.199866755496345, 80.186542305129919, 80.173217854763493, 80.159893404397067, 80.159893404397067, 80.119920053297804, 80.093271152564967, 80.053297801465689, 80.026648900732837, 80.026648900732837, 80.013324450366412, 80.0, 79.973351099267148, 79.933377748167885, 79.933377748167885, 79.880079946702196, 79.826782145236507, 79.760159893404392, 79.66688874083944, 79.573617588274487, 79.546968687541636, 79.560293137908062, 79.493670886075947, 79.493670886075947, 79.440373084610258, 79.427048634243832, 79.360426382411731, 79.387075283144569, 79.373750832778143, 79.293804130579616, 79.307128580946042, 79.307128580946042, 79.320453031312454, 79.293804130579616, 79.293804130579616, 79.240506329113927, 79.240506329113927, 79.200532978014664, 79.160559626915401, 79.107261825449697, 79.067288474350434, 79.027315123251157, 79.000666222518319, 78.934043970686204, 78.907395069953367, 78.894070619586941, 78.894070619586941, 78.854097268487678, 78.854097268487678, 78.774150566289137, 78.774150566289137, 78.734177215189874, 78.694203864090611, 78.707528314457036, 78.694203864090611, 78.654230512991347, 78.574283810792807, 78.494337108594266, 78.467688207861428, 78.387741505662888, 78.347768154563624, 78.294470353097935, 78.22784810126582, 78.16122584943372, 78.134576948700868]

[0.01001, 0.02001, 0.030010000000000002, 0.040010000000000004, 0.050010000000000006, 0.06001000000000001, 0.07001, 0.08001, 0.09000999999999999, 0.10000999999999999, 0.11000999999999998, 0.12000999999999998, 0.13001, 0.14001, 0.15001, 0.16001, 0.17001000000000002, 0.18001000000000003, 0.19001000000000004, 0.20001000000000005, 0.21001000000000006, 0.22001000000000007, 0.23001000000000008, 0.24001000000000008, 0.25001000000000007, 0.2600100000000001, 0.2700100000000001, 0.2800100000000001, 0.2900100000000001, 0.3000100000000001, 0.3100100000000001, 0.3200100000000001, 0.33001000000000014, 0.34001000000000015, 0.35001000000000015, 0.36001000000000016, 0.37001000000000017, 0.3800100000000002, 0.3900100000000002, 0.4000100000000002, 0.4100100000000002, 0.4200100000000002, 0.4300100000000002, 0.44001000000000023, 0.45001000000000024, 0.46001000000000025, 0.47001000000000026, 0.48001000000000027, 0.4900100000000003, 0.5000100000000003, 0.5100100000000003, 0.5200100000000003, 0.5300100000000003, 0.5400100000000003, 0.5500100000000003, 0.5600100000000003, 0.5700100000000003, 0.5800100000000004, 0.5900100000000004, 0.6000100000000004, 0.6100100000000004, 0.6200100000000004, 0.6300100000000004, 0.6400100000000004, 0.6500100000000004, 0.6600100000000004, 0.6700100000000004, 0.6800100000000004, 0.6900100000000005, 0.7000100000000005, 0.7100100000000005, 0.7200100000000005, 0.7300100000000005, 0.7400100000000005, 0.7500100000000005, 0.7600100000000005, 0.7700100000000005, 0.7800100000000005, 0.7900100000000005, 0.8000100000000006, 0.8100100000000006, 0.8200100000000006, 0.8300100000000006, 0.8400100000000006, 0.8500100000000006, 0.8600100000000006, 0.8700100000000006, 0.8800100000000006, 0.8900100000000006, 0.9000100000000006, 0.9100100000000007, 0.9200100000000007, 0.9300100000000007, 0.9400100000000007, 0.9500100000000007, 0.9600100000000007, 0.9700100000000007, 0.9800100000000007, 0.9900100000000007, 1.0000100000000007]

WORDS(TFIDF)

'nhl'

'stephanopoulos'

'leafs'

'alomar'

'wolverine'

'crypto'

'lemieux'

'oname'

'rsa'

'ripem'

'athos'

'rbi'

'firearm'

'powerbook'

'pitcher'

'dyer'

'bruins'

'lciii'

'lindros'

'fprintf'

'ahl'

'azerbaijan'

'candida'

'iisi'

'args'

'baerga'

'gilmour'

'gfci'

'clh'

'pitchers'

'clemens'

'dodgers'

'gainey'

'sabretooth'

'liefeld'

'rlk'

'jagr'

'adb'

'hobgoblin'

'hawks'

'crypt'

'anonymity'

'aspi'

'countersteering'

'punisher'

'xfree'

'azerbaijani'

'cipher'

'recchi'

'sdpa'

'oilers'

'soderstrom'

'obp'

'argic'

'libxmu'

'jaeger'

'goalie'

'serdar'

'inning'

'sumgait'

'xmu'

'umu'

'gaza'

'denning'

'ioccc'

'obfuscated'

'rayshade'

'nsmca'

'xdm'

'ranck'

'dineen'

'stderr'

'dpy'

'cardinals'

'homicides'

'orbiter'

'mozumder'

'potvin'

'sandberg'

'uccxkvb'

'imake'

'plaintext'

'whalers'

'moncton'

'mydisplay'

'wip'

'hicnet'

'steveh'

'bontchev'

'karabakh'

'baku'

'canadiens'

'messier'

'bure'

'bikers'

'cryptographic'

'mutants'

'keown'

'ssto'

WORDS(IG)

1 windows

2 god

3 he

4 scsi

5 car

6 drive

7 space

8 team

9 dos

10 bike

11 file

12 of

13 that

14 mb

15 game

16 key

17 mac

18 jesus

19 window

20 dod

21 hockey

22 the

23 graphics

24 card

25 image

26 his

27 gun

28 encryption

29 sale

30 apple

31 government

32 season

33 we

34 games

35 israel

36 disk

37 files

38 ide

39 controller

40 players

41 shipping

42 chip

43 program

44 was

45 cars

46 nasa

47 win

48 year

49 were

50 they

51 turkish

52 motif

53 people

54 armenian

55 play

56 drives

57 bible

58 use

59 widget

60 pc

61 clipper

62 offer

63 jpeg

64 baseball

65 bus

66 my

67 nhl

68 software

69 is

70 db

71 server

72 jews

73 os

74 israeli

75 output

76 data

77 system

78 who

79 league

80 armenians

81 for

82 christian

83 christians

84 entry

85 mhz

86 ftp

87 price

88 christ

89 guns

90 thanks

91 church

92 color

93 teams

94 privacy

95 condition

96 launch

97 him

98 com

99 monitor

100 ram

101 memory