Assignment 4: Naïve Bayes Classifier and Deep Learning for Natural Language Processing

UVA CS4774

Machine Learning Foundation

Owen Richards

<u>Naïve Bayes Classifier</u>

Q1) This is the verification of correct implementation of the preprocessing for the dataset and building a vocabulary of unique words from the training corpus being ranked with their frequency.

```
--------------------
number of unique words:  26060
the most common 10 words were: ['film', 'movi', 'one', 'like', 'charact', 'make', 'get', 'time', 'scene', 'even']
the least common 10 words were: ['looooot', 'schnazzi', 'timex', 'indiglo', 'jessalyn', 'gilsig', 'ruber', 'jaleel', 'balk
i', 'guesswork']
number of unique words in vocabulary:  100
--------------------
```

Fig 1: Verification of Correct Preprocessing

Q2) To build the bag of words an additional transfer function was created to convert a text document into a feature vector. This function is shown in the figure below where the parameters are review and vocabulary. This function puts the text documents into a feature vector named foo in this case which is the length of vocabulary.

```
def transfer(review, vocabulary):
    foo = [0]*len(vocabulary)
    for word in review:
        if word in vocabulary:
            index = vocabulary.index(word)
            foo[index] += 1
    return foo
```

Fig 2: Transfer Function

Q3) Using the transfer function above, a Bag of Words function was created that stores the features into matrix Xtrain and Xtest. Additionally, ytrain and ytest stores the labels. Both x_train_bow and x_test_bow are numpy arrays with bag of words representation.

```python
def getBOWRepresentation(x_train, x_test, vocabulary):
    '''
    converts into Bag of Words representation
    each column is a feature(unique word) from the vocabulary
    x_train_bow : a numpy array with bag of words representation
    '''
    # --------your code here-------------
    vocabulary = list(vocabulary)
    x_train_bow = []
    x_test_bow = []
    for text_data in x_train:
        x_train_bow.append(transfer(text_data, vocabulary))
    for text_data in x_test:
        x_test_bow.append(transfer(text_data, vocabulary))
    x_train_bow = np.array(x_train_bow)
    x_test_bow = np.array(x_test_bow)

    return x_train_bow, x_test_bow
```

Fig 3: Get BOW Representation

Q4) The results of:

thetaPosTrue, thetaNegTrue = *naiveBayesBernFeature_train*(Xtrain, ytrain)

is shown in the figure below.

```
thetaPosTrue = [0.4045584045584046, 0.5156695156695157, 0.5569800569800569, 0.6908831908831908, 0.905982905982906, 0.25925
925925925924, 0.6153846153846154, 0.5028490028490028, 0.36752136752136755, 0.801994301994302, 0.44871794871794873, 0.26495
726495726496, 0.7421652421652422, 0.6253561253561254, 0.9088319088319088, 0.5925925925925926, 0.5498575498575499, 0.481481
48148148145, 0.584045584045584, 0.3803418803418803, 0.3732193732193732, 0.7122507122507122, 0.5384615384615384, 0.38176638
17663818, 0.4757834757834758, 0.7606837606837606, 0.6096866096866097, 0.48575498575498577, 0.3931623931623915, 0.33618233
618233617, 0.4045584045584046, 0.31054131054131057, 0.3974358974358974, 0.43162393162393164, 0.34045584045584043, 0.333333
3333333333, 0.2948717948717949, 0.46153846153846156, 0.3717948717948718, 0.42592592592592593, 0.30484330484330485, 0.46296
296296296297, 0.452991452991453, 0.3333333333333333, 0.46438746438746437, 0.3803418803418803, 0.5384615384615384, 0.405982
905982906, 0.5897435897435898, 0.41452991452991456, 0.37606837606837606, 0.5569800569800569, 0.5954415954415955, 0.5327635
327635327, 0.39886039886039887, 0.39316239316239315, 0.7350427350427351, 0.5797720797720798, 0.31196581196581197, 0.256410
2564102564, 0.33760683760683763, 0.35185185185185186, 0.5256410256410257, 0.32763532763532766, 0.5669515669515669, 0.28205
128205128205, 0.35327635327635326, 0.5584045584045584, 0.39316239316239315, 0.3190883190883191, 0.5042735042735043, 0.3347
5783475783477, 0.3547008547008547, 0.4358974358974359, 0.33760683760683763, 0.3262108262108262, 0.33760683760683763, 0.378
9173789173789, 0.34045584045584043, 0.24786324786324787, 0.33048433048433046, 0.44017094017094016, 0.5569800569800569, 0.3
7037037037037035, 0.3418803418803419, 0.3418803418803419, 0.42165242165242167, 0.39316239316239315, 0.37037037037037035, 0
.45726495726495725, 0.3660968660968661, 0.3603988603988604, 0.4287749287749288, 0.3091168091168091, 0.4045584045584046, 0.
32051282051282054, 0.405982905982906, 0.35327635327635326, 0.3817663817663818, 0.5256410256410257]
thetaNegTrue = [0.5185185185185185, 0.5256410256410257, 0.5527065527065527, 0.7222222222222222, 0.8746438746438746, 0.2920
22792022792, 0.5270655270655271, 0.3418803418803419, 0.3945868945868946, 0.8461538461538461, 0.3333333333333333, 0.5042735
042735043, 0.7037037037037037, 0.6680911680911681, 0.8931623931623932, 0.5925925925925926, 0.5199430199430 2, 0.51994301994
302, 0.5071225071225072, 0.23931623931623933, 0.3660968660968661, 0.717948717948718, 0.5270655270655271, 0.361823361823361
83, 0.5541310541310541, 0.8062678062678063, 0.5826210826210826, 0.47863247863247865, 0.45014245014245013, 0.33190883190883
19, 0.43162393162393164, 0.33903133903133903, 0.3831908831908832, 0.42592592592592593, 0.31054131054131057, 0.356125356125
3561, 0.3176638176638177, 0.46153846153846156, 0.33475783475783477, 0.37464387464387466, 0.3148148148148148, 0.39316239316
239315, 0.46438746438746437, 0.31054131054131057, 0.48148148148148145, 0.3418803418803419, 0.5897435897435898, 0.390313390
3133903, 0.6253561253561254, 0.3504273504273504, 0.31054131054131057, 0.5113960113960114, 0.47863247863247865, 0.485754985
75498577, 0.4031339031339031, 0.3817663817663818, 0.6965811965811965, 0.5242165242165242, 0.33048433048433046, 0.262108262
1082621, 0.3660968660968661, 0.3817663817663818, 0.48148148148148145, 0.3076923076923077, 0.5498575498575499, 0.3376068376
0683763, 0.31339031339031337, 0.5071225071225072, 0.36324786324786323, 0.34045584045584043, 0.36324786324786323, 0.3190883
190883191, 0.2792022792022792, 0.3945868945868946, 0.31196581196581197, 0.3005698005698006, 0.32051282051282054, 0.3547008
547008547, 0.3575498575498576, 0.28062678062678065, 0.292022792022792, 0.3575498575498576, 0.5056980056980057, 0.299145299
14529914, 0.25925925925925924, 0.2934472934472934, 0.4159544159544159 6, 0.3319088319088319, 0.3575498575498576, 0.31908831
90883191, 0.39173789173789175, 0.396011396011396, 0.2692307692307692, 0.3646723646723647, 0.4658119658119658, 0.3034188034
188034, 0.39173789173789175, 0.30484330484330485, 0.34615384615384615, 0.37464387464387466]
--------------------
```

Fig 4: Results of thetaPosTrue and thetaNegTrue

Q5) The resulting accuracy of the naiveBayrsBernFeature_test function is shown in the figure below.

```
--------------------
BNBC classification accuracy = 0.685
--------------------
--------------------
```

Fig 5: Resulting Accuracy of naiveBayrsBernFeature_test Function

Q6) The accuracy of using the "sklearnnaive_bayes.MultinomialNB" from the scikit learn package to perform training and testing is shown in the figure below. The accuracy of this is in the line Sklearn MultinomialNB accuracry. Additionally, the MNBC accuracy can be seen in the figure also.

```
 0.009801100600654606, 0.005395101248066756, 0.007589109088947236, 0.005916627702046541, 0.006689925547602777, 0.007445239
722332122]
--------------------
MNBC classification accuracy = 0.7416666666666667
Sklearn MultinomialNB accuracy = 0.7416666666666667
```

Fig 6: Sklean MultinomialNB Accuracy and MNBC accuracy

Additionally, below are my results for thetaPos and thetaNeg for reference.

```
thetaNeg = [0.012300830845592203, 0.011419630975074632, 0.013110096032802216, 0.021868143725497725, 0.03571557026220192, 0.
006438154156026328, 0.01233679818724598, 0.006635974535122109, 0.007211452001582563, 0.04593029529187498, 0.00627630111858
4325, 0.012894291982879546, 0.019044707405675647, 0.01744416070208251, 0.06279897852749704, 0.015268136532028917, 0.010700
284141999064, 0.01190519008740064, 0.010502463762903284, 0.004567852390029853, 0.006779843901737223, 0.022677408912707264,
 0.010124806675538611, 0.0064921051685069954, 0.012912275653706435, 0.026813653202891773, 0.01614933640254649, 0.010700284
141999064, 0.009063770096752148, 0.006060497068661655, 0.007679027443081682, 0.007319354026543898, 0.007463223393159012, 0
.009962953638096609, 0.005754774664604539, 0.0066000071934683305, 0.00595259504370032, 0.009459410854943711, 0.00613243175
19692115, 0.0075171744056396796, 0.005449052260547423, 0.008236521238715247, 0.010448512750422616, 0.005934611372873431, 0
.008901917059310146, 0.0062403337769305475, 0.013595655145128224, 0.007858864151350573, 0.014728626407222242, 0.0069416969
391792255, 0.005556954285508758, 0.011203826925151962, 0.009549329209078156, 0.009675214904866381, 0.008020717188792577, 0
.006653958205948998, 0.018127540193504297, 0.012264863503938423, 0.008272488580369025, 0.005215264539797863, 0.00600654605
6180988, 0.006869762255871668, 0.009891018954789051, 0.005503003273028091, 0.01276840628709132, 0.005682839981296982, 0.00
5574937956335647, 0.01082616983778729, 0.006707909218429666, 0.005934611372873431, 0.006510088839333885, 0.005377117577239
866, 0.00462180340251052, 0.00794878250548502, 0.005826709347912096, 0.005359133906412977, 0.005538970614681869, 0.0067798
43901737223, 0.006761860230910333, 0.005934611372873431, 0.005359133906412977, 0.007463223393159012, 0.010286659712980614,
 0.0049634931482214146, 0.004675754414991188, 0.006438154156026328, 0.008092651872100133, 0.005467035931374312, 0.00643815
4156026328, 0.005826709347912096, 0.0074092723806783445, 0.008002733517965687, 0.004747689098298744, 0.006707909218429666,
 0.009801100600654606, 0.005395101248066756, 0.007589109088947236, 0.005916627702046541, 0.006689925547602777, 0.007445239
722332122]
```

Fig 7: ThetaNeg Parameter

```
thetaPos = [0.007792550124384257, 0.01116987100281718, 0.011416992042702516, 0.017858613815713603, 0.036540964431044994, 0
.004711774493813737, 0.014843737129112506, 0.012421950938236215, 0.006705217548888779, 0.03616204550322081, 0.007809024860
376612, 0.004448178717936045, 0.01894594639120908, 0.013739929817624673, 0.07168157630273975, 0.014678989769188949, 0.0119
27708858465544, 0.010692103659038863, 0.013328061417815779, 0.008303266940147285, 0.006573419660949933, 0.0231634788052521
47, 0.010790952074992997, 0.006655793340911712, 0.009637720555528098, 0.023262327221206282, 0.01571689813670736, 0.0096871
44763505165, 0.006886439644804692, 0.005271915517553831, 0.007677226972437766, 0.005403713405492677, 0.006523995452972866,
 0.00909405426778036, 0.00558493550140859, 0.005568460765416235, 0.004810622909767871, 0.009159953211749781, 0.00660636913
2934645, 0.008583337452017331, 0.005255440781561475, 0.009044630059803292, 0.008863407963887378, 0.006606369132934645, 0.0
07545429084498921, 0.006441621773011087, 0.011878284650488477, 0.007413631196560075, 0.0142506466333877, 0.008500963772055
553, 0.006573419660949933, 0.011647638346595495, 0.013987050857510008, 0.011219295210794247, 0.008682185867971466, 0.00655
6944924957578, 0.01868235061533139, 0.015881645496630917, 0.006787591228850557, 0.0046623502858366694, 0.00517306710159969
65, 0.006145076525148684, 0.010972174170908911, 0.005749682861332147, 0.01186180991449612, 0.004612926077859602, 0.0066228
438689270005, 0.011795910970526697, 0.007528954348506565, 0.005024794477668495, 0.009884841595413434, 0.005255440781561475
, 0.006474571244995799, 0.009176427947742137, 0.0056014102374009455, 0.005370763933507965, 0.0064580965090034435, 0.006359
248093049309, 0.005502561821446811, 0.0051730671015996965, 0.00593090495724806, 0.009851892123428721, 0.011351093098733092
, 0.00604622810919455, 0.006128601789156329, 0.006919389116789403, 0.00797377222030017, 0.006573419660949933, 0.0072488838
36636518, 0.008995205851826225, 0.005749682861332147, 0.006309823885072242, 0.009324700571673339, 0.004464653453928401, 0.
007248883836636518, 0.005502561821446811, 0.006952338588774115, 0.006194500733125751, 0.0073477322525906524, 0.01082390154
697771]
```

Fig 7: ThetaPos Parameter

State-of-the-art Deep Learning for NLP: BERT

1) BERT model
    a. The performance of the BERT model on the dataset was 0.9729999899864197.
    b. The figure of both training and validation loss is shown below.

Fig 8: Training and Validation Loss vs Epoch Number

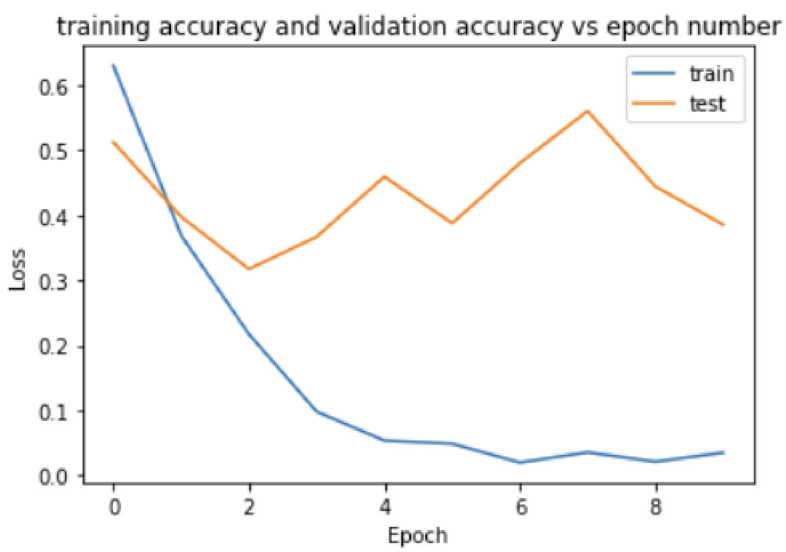

Fig 9: Training and Validation Accuracy vs Epoch Number

c. The batch size I used for the model type BERT was 8. The epoch number was 2, the learning rate was 0.00001 and the validation split was 0.1. Below is a snippet of the code specifying all my args.

```
if model_type == 'BERT':
  args = {
  'batch_size': 8,
  'validation_split': 0.1,
  'epoch': 2,
  'learning_rate': 0.00001
  }
```

Fig 10: Report of Arguments for BERT

2) The accuracy of the best model was 0.973 and this accuracy was done by the BERT model.
   In regards to the best accuracy that I got from the MLP model, that was 0.846666693687439. In figure 11 and 12 there is a graph of both training and validation loss, and training and validation accuracy.

Fig 11: Training and Validation Loss vs Epoch Number

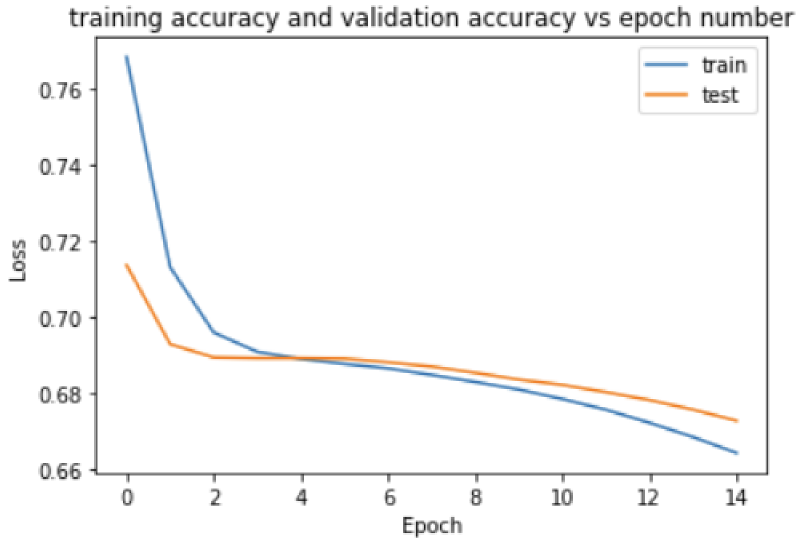

Fig 12: Training and Validation Accuracy vs Epoch Number

Finally, the arguments made for the MLP model was a batch size of 8, validation split of 0.1, a epoch of 2 and a learning rate of 0.03.