CSC 4780/6780
Fall 2022
Homework 08

October 9, 2022

This homework is due at 11:59 pm on Sunday, Oct 16. It must be uploaded to iCollege by then. No credit will be given for late submissions. A solution will be released by noon on Monday, Oct 17.

it is always a good idea to get this done and turned in early. You can turn it in as many times as you like – iCollege will only keep the last submission. If, for some reason, you are unable to upload your solution, email it to me before the deadline.

Incidentally, I rarely check my iCollege mail, but I check my `dhillegass@gsu.edu` email all the time. Send messages there.

If you are reading this, the very first thing you should do is rename the directory it is in: `HW08_Jones_Fred`. Those of you who leave it `HW08_last_first` are messing up the grading process.

1 Sentiment Analysis Using a Multinomial Naive Bayesian Classifier

A Bayesian classifier is a wonderful thing: given an input, you get a probability for every possible output.

A naive Bayesian classifier makes the simplifying assumption that every dimension of the input vector is independent.

In this exercise, you will use a naive Bayesian classifier to classify a tweet using the "Bag of Words" approach.

I've supplied you with a collection of tweets to airlines that have been labeled "positive", "negative", or "neutral". You will develop a system that will be able to label the sentiment of a tweet with about 80% accuracy.

1.1 Prep the data

You are given `Tweets.csv` which is a real data set from Kaggle: `https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment`

You are also given a program `tweet_prep_data.py` that:

- Reads `Tweets.csv` using the `csv` library. (In this assignment, you are not allowed to use pandas. Sometimes you will want to use the `csv` library so that you don't have to have the whole dataset in memory at the same time.)
- Discards tweets where the labeler was not at least 50
- Ignores all stop words and names of airlines
- Saves out a list of the 2000 most common words that remain.
- Splits the remaining data into `train_tweet.csv` and `test_tweet.csv`. About 10% of the tweets will end up in `test_tweet.csv`.
- Prints out the 32 most common words.

If you haven't already, you will need to install nltk and its English stopwords:

```
> pip3 install nltk
> python3
Type "help", "copyright", "credits" or "license" for more information.
>>>> import nltk
>>>> nltk.download('stopwords')
```

You do not need to change `tweet_prep_data.py` at all. Run it to create `vocablist_tweet.pkl`, `train_tweet.csv`, and `test_tweet.csv`.

(Incidentally, in this assignment you won't need to change `util.py` at all either.)

Here's what it should look like when you run it:

```
> python3 tweet_prep_data.py
Kept 14404 rows, discarded 236 rows
Most common 32 words are ['flight', 'get', 'cancelled', 'thanks', 'service',
      'help', 'time', 'customer', 'im', 'us', 'hours', 'flights', 'hold', 'amp',
      'plane', 'thank', 'cant', 'still', 'one', 'please', 'need', 'would', 'delayed',
      gate', 'back', 'flightled', 'call', 'dont', 'bag', 'hour', 'got', 'late']
Wrote 2000 words to vocablist_tweet.pkl.
```

(Why is "flightled" on this list? I have no idea. Real data is sometimes weird.)

1.2 Use the training data

You will need to complete the program called `tweet_train.py` that

- Reads in `vocablist_tweet.pkl`.
- Goes through `train_tweet.csv` row by row, counting the words These counts will be used to create a word frequency vector for each sentiment.
- It will also count the tweets for each sentiment, so that it can say things like "63.3% of all these tweets are negative.". These frequencies will act as your priors.
- Take the log of all the word frequencies and the sentiment frequencies. Save them both to a single file named `frequencies_tweet.pkl`.
- Print out the 10 most positive words and the 10 most negative (as determined by the difference between the Sentiment 0 frequency and the Sentiment 2 frequency).

Words that don't appear at all for a sentiment should be treated as if they appeared 0.5 times.

When it runs, it should look something like this:

```
> python3 tweet_train.py
Skipped 81 tweets: had no words from vocabulary
*** Tweets by sentiment ***
    0 (negative): 63.5%
    1 (neutral): 20.5%
    2 (positive): 16.0%
Positive words:
    thanks thank great love awesome best much good amazing guys
Negative words:
    flight cancelled hours hold delayed call get flightled hour dont
```

1.3 Test with the testing data

You will need to complete the program called `test_tweet.py` that

- Reads in `vocablist_tweet.pkl` and `frequencies_tweet.pkl`.
- Goes through `test_tweet.csv` row by row, using Bayesian inference to guess if it is positive, negative, or neutral.
- At the end, it should give some statistics on its performance, like accuracy and a confusion matrix. This should include a baseline of "How many would the system get right if it ignored the data and just guessed the most common class?"

- Besides a guess, the Bayesian Classifier gives us a probability that it is correct. If we discard the results that it is less sure of, we would expect it's accuracy to increase. Make a plot showing this.

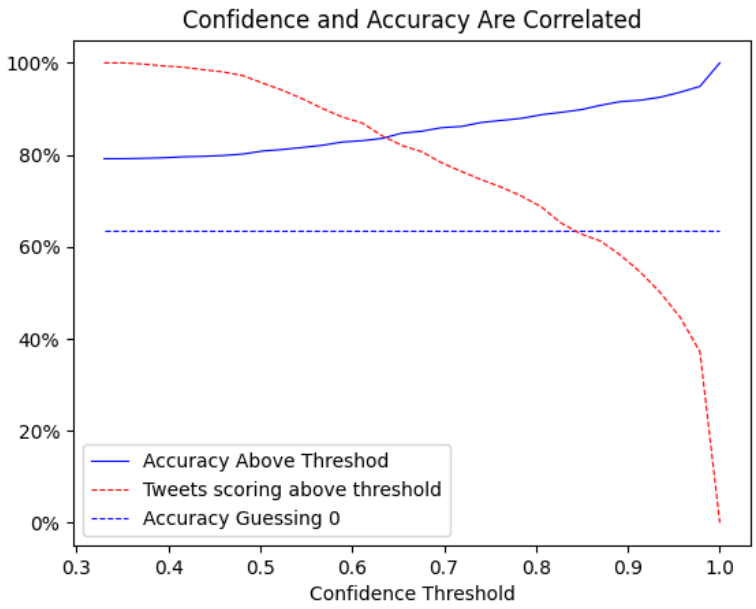
When it runs, it should look like this:

```
> python3 tweet_test.py
Would get 63.5% accuracy by guessing "0" every time.
Skipped 9 rows for having none of the common words
1468 lines analyzed, 1162 correct (79.2% accuracy)
Confusion:
[[843  58  31]
 [118 159  34]
 [ 42  23 160]]
```

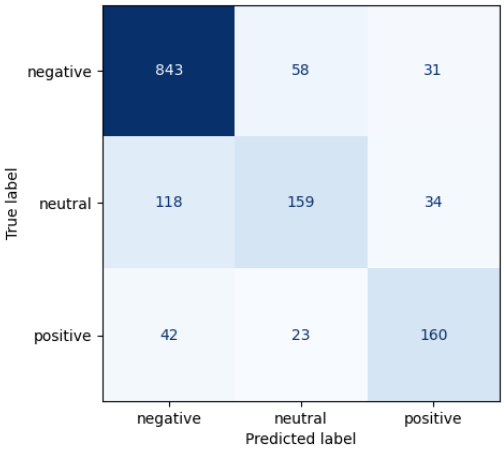
Remember: If W is the log word frequency matrix (one row per sentiment) and $\vec{c}$ is the word count vector (a column vector), then the log likelihood is computed with a matrix multiplication:

$$W\vec{c}$$

Your `test_tweet.py` should also create a plot called `confidence_tweet.png` that looks like this:



And a confusion matrix that looks like this:


negative
neutral
positive
True label
843
58
31
118
159
34
42
23
160
negative
neutral
positive
Predicted label

2 Extra Credit

Another really useful type of classifier is the Gaussian Naive Bayesian Classifier. I like it so much that I will give you a bonus 8 points if you implement one. On the exercise, there is a lot of guidance in the template files. On this bonus problem, not so much. But they have a lot in common.

This is completely optional! Don't kill yourself trying to get it done.

2.1 gn_train.py

You are given `train_gn.csv`. You will write a program called `gnclassifier.py`.

The data in `train_gn.csv` looks like this:

```
X0,X1,X2,X3,Y
3.42,43.62,11.68,10.17,PY9
3.07,88.66,10.01,20.09,PY9
1.69,16.20,7.81,10.70,TK1
1.04,35.51,8.37,16.85,RM9
3.50,59.01,10.24,10.73,PY9
```

The last column is the class of that datapoint. That is what you are trying to predict.

You will compute the mean and standard deviation of each attribute for each class

You will also figure out the prior for each class.

You will store the labels, the priors, the means, and the standard deviations in `parameters_gn.pkl`.

When it runs, it should look like this:

```
> python3 gn_train.py
Read 1198 samples with 4 attributes from train_gn.csv
Priors:
    D6X: 37.3%
    PY9: 35.7%
    RM9: 19.9%
    TK1: 5.0%
    ZZ4: 2.1%
D6X:
    Means -> X0: 4.0291     X1:33.4690     X2: 6.8480     X3: 7.3198
    Stdvs -> X0: 0.5664     X1:16.4635     X2: 4.1119     X3: 5.3221
PY9:
    Means -> X0: 3.0236     X1:62.7718     X2: 7.0495     X3:15.4692
    Stdvs -> X0: 0.5868     X1:23.4633     X2: 2.9422     X3: 5.7730
RM9:
```

```
    Means -> X0: 1.9467     X1:24.9554     X2: 6.9616     X3:10.4789
    Stdvs -> X0: 0.5615     X1: 6.8383     X2: 1.8640     X3: 2.7498
TK1:
    Means -> X0: 1.0353     X1:13.8293     X2: 7.0343     X3:10.5025
    Stdvs -> X0: 0.6032     X1: 9.0163     X2: 1.1721     X3: 2.5536
ZZ4:
    Means -> X0: 4.9596     X1:14.5060     X2: 5.5684     X3: 1.8624
    Stdvs -> X0: 0.5395     X1: 4.5257     X2: 6.3474     X3: 5.6984
Wrote parameters to parameters_gn.pkl
```

2.2 gn_train.py

You will create a second program called `gn_train.py`. It will read `parameters_gn.pkl`. (Be sure to take the log of the priors before adding them to the log likelihoods!)

Then it will go through each row of `test_gn.csv` and use a Gaussian Naive Bayes approach to predict the class for that row.

Print the probabilities of each class for the first ten rows of data.

Then you will produce the same sorts of metrics that you did for the tweets.

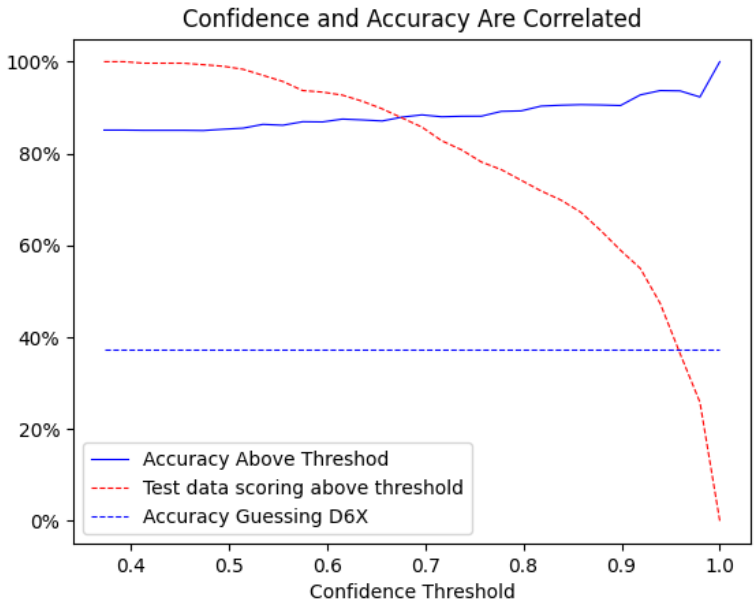
When it runs, the command line will look like this:

```
> python3 gn_test.py
Read parameters from parameters_gn.pkl
Can expect 37.3% accuracy by guessing "D6X" every time.
Read 302 rows from test_gn.csv
Here are 10 rows of results:
GT=RM9 -> D6X:  0.0%     PY9:  1.0%     RM9: 96.3%     TK1:  2.7%     ZZ4:  0.0%
GT=PY9 -> D6X:  0.0%     PY9:100.0%     RM9:  0.0%     TK1:  0.0%     ZZ4:  0.0%
GT=RM9 -> D6X:  0.0%     PY9:  0.8%     RM9: 88.9%     TK1: 10.3%     ZZ4:  0.0%
GT=D6X -> D6X: 89.3%     PY9: 10.5%     RM9:  0.2%     TK1:  0.0%     ZZ4:  0.0%
GT=D6X -> D6X: 97.1%     PY9:  2.8%     RM9:  0.0%     TK1:  0.0%     ZZ4:  0.1%
GT=PY9 -> D6X:  0.0%     PY9:100.0%     RM9:  0.0%     TK1:  0.0%     ZZ4:  0.0%
GT=RM9 -> D6X:  0.4%     PY9: 46.3%     RM9: 52.6%     TK1:  0.8%     ZZ4:  0.0%
GT=D6X -> D6X: 98.7%     PY9:  0.5%     RM9:  0.0%     TK1:  0.0%     ZZ4:  0.8%
GT=RM9 -> D6X:  0.0%     PY9:  0.1%     RM9: 55.3%     TK1: 44.6%     ZZ4:  0.0%
GT=D6X -> D6X: 99.3%     PY9:  0.6%     RM9:  0.0%     TK1:  0.0%     ZZ4:  0.1%

*** Analysis ***
302 data points analyzed, 257 correct (85.1% accuracy)
Confusion:
[[114  14   0   0   2]
 [ 12  76   7   0   0]
 [  3   0  56   2   0]
 [  0   0   2   9   0]
```

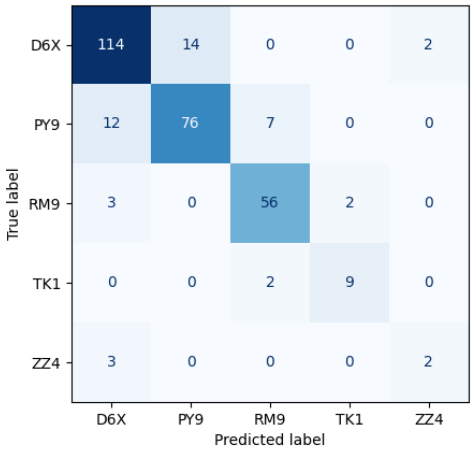
```
 [  3   0   0   0   2]]
Wrote confusion matrix plot to confusion_gn.png

*** Making a plot ****
Saved to "confidence_gn.png".
```

(`GT` stands for "Ground Truth". It is the right answer, what we are trying to predict.)

The plots will look like this:



And this:



2.3 Some handy math for the bonus problem

Remember that, by the naive Bayes assumption, the likelihood of a vector is just the product of the likelihoods of each dimension.

$$p(\vec{x}|y=j) = p_0(x_0|y=j)p_1(x_1|y=j)p_2(x_2|y=j)p_3(x_3|y=j)$$

By the Gaussian assumption, we are assuming that each of $p_i(x_i|y=j)$ is a normal distribution given by:

$$p_{j,i}(x_i) = \frac{1}{\sigma_{j,i}\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_{j,i}}{\sigma_{j,i}}\right)^2}$$

Once again, we are going to work in "log space", so we note that:

$$\log p(\vec{x}|y=j) = \log p_0(x_0|y=j) + \log p_1(x_1|y=j) + \log p_2(x_2|y=j) + \log p_3(x_3|y)$$

So we really want the log of $p_{j,i}$, which you could derive from the equation above. Here it is:

$$\log p_{j,i}(x_i) = -\log \sigma_{j,i} - \frac{\log(2\pi)}{2} - \frac{1}{2}\left(\frac{x_i - \mu_{j,i}}{\sigma_{j,i}}\right)^2$$

(And, in case I haven't said it, in this class you can always assume log means "natural logarithm".)

Don't forget the priors when you compute the posterior!

3 Criteria for success

If your name is Fred Jones, you will turn in a zip file called `HW08_Jones_Fred.zip` of a directory called `HW08_Jones_Fred`. It will contain:

- `tweet_prep_data.py`
- `tweet_train.py`
- `tweet_test.py`
- `test_tweet.csv`
- `train_tweet.csv`
- `Tweets.csv`

- util.py
- vocablist_tweet.pkl
- frequencies_tweet.pkl
- confidence_tweet.png
- confusion_tweet.png

If you do this bonus, also include the following in your zipfile:

- gn_train.py
- gn_test.py
- train_gn.csv
- test_gn.cvs
- confidence_gn.png
- confusion_gn.png

Be sure to format your python code with black before you submit it.

We will run your code like this:

```
cd HW08_Jones_Fred
python3 tweet_prep_data.py
python3 tweet_train.py
python3 tweet_test.py
```

For the bonus, we will run your code like this:

```
cd HW08_Jones_Fred
python3 gn_train.py
python3 gn_test.py
```

Do this work by yourself. Stackoverflow is OK. A hint from another student is OK. Looking at another student's code is *not* OK.

The template files for the python programs have import statements. Do not use any frameworks not in those import statements.

4 Reading

Our textbook, which I generally really like, has terrible coverage the naive Bayesian approach.

Here is a very good video explaining exactly what we are doing with tweet classification: `https://youtu.be/O2L2Uv9pdDA`

Same guy explaining the gaussian naive bayes approach: `https://youtu.be/H3EjCKtlVog`