NLP-disaster-NB

October 19, 2025

1 NLP Disaster Tweets Continued

1.1 Bernoulli Naive Bayes

1.1.1 Import preprocessed data

I imported preprocessed data using the same preprocessing steps from HW2. A summary of the preprocessing steps is shown below.

1. Lowercase all letters
2. Only keep alphanumerics, # and @
3. Keep the http or https part of links, get rid of the rest of the link
4. Keep the @ part of @usernames, get rid of the actual username
5. Keep the # of #topics, keep both topic and # but separate them
6. Lemmatize all words
7. Remove common stop words (and, or, the ..)

```
[2]: import pandas as pd

     train = pd.read_csv("/home/jiwonjjeong/cornell-tech/CT-applied-machine-learning/
       ↪hw-2/data-from-NLP-disaster-tweets/preprocessed/train.csv")
     test = pd.read_csv("/home/jiwonjjeong/cornell-tech/CT-applied-machine-learning/
       ↪hw-2/data-from-NLP-disaster-tweets/preprocessed/test.csv")
     val = pd.read_csv("/home/jiwonjjeong/cornell-tech/CT-applied-machine-learning/
       ↪hw-2/data-from-NLP-disaster-tweets/preprocessed/validate.csv")
     print(train.shape)
     train[10:20]
```

```
(5329, 5)
```

```
[2]:        id       keyword                     location  \
     10  10691         wreck    ?currently writing a book?
     11   7148      mudslide       Malibu/SantaFe/Winning!
     12   2017     casualties                Canadian bread
     13    262     ambulance                     Amsterdam
     14   4468   electrocuted                           NaN
     15   4176         drown                           NaN
     16  10243       volcano        West Coast, Cali USA
     17    681        attack                        Mumbai
     18   1399     body%20bag                      New York
```

```
19    296    annihilated              Higher Places
```

```
                                         text  target
10  friggin wreck destiel suck read vine descripti…       0
11  sterlingscott red carpet fundrais oso mudslid …       1
12  @ honest peopl want go rampag let use hand fee…       0
13  http twelv fear kill pakistani air ambul helic…       1
14  got electrocut last night work first time life…       0
15  older nativ australian believ ocean creat urin…       0
16             architect behind kany west volcano http       0
17  india shud give ani evid pakthey share terrori…       1
18  auth loui vuitton brown saumur 35 cross bodi s…       0
19  episod trunk annihil freiza cleanest shit ever…       0
```

1.1.2 Create feature vectors using Naive-Bayes assumption

I will assume that each word can be considered independently, allowing me to represent the inclusion of each word as its own individual feature. Below, I am creating the word dictionary to use as features based on the training set words.

```python
[3]: # Get set of unique words in the training set
     unique_words = set(' '.join(train['text']).split())
     len(unique_words)
     words = list(unique_words)
```

```python
[4]: # Generate 10339 dimensional vector to represent each sample and embedd using␣
     ↪naive bayes assumption
     train_dict = {}
     val_dict = {}
     test_dict = {}
     for word in words:
         train_dict[f'has_{word}'] = train['text'].str.contains(word, na=False).
      ↪astype(int)
         val_dict[f'has_{word}'] = val['text'].str.contains(word, na=False).
      ↪astype(int)
         test_dict[f'has_{word}'] = test['text'].str.contains(word, na=False).
      ↪astype(int)
```

```python
[5]: # Concatenate the dictionary into dataframe
     train_no_target = pd.DataFrame(train_dict)
     val_no_target = pd.DataFrame(val_dict)
     test = pd.DataFrame(test_dict)
     train = pd.concat([train_no_target,train["target"]], axis=1)
     val = pd.concat([val_no_target,train["target"]], axis=1)
     train.head()
```

```
[5]:      has_wasnt   has_cree   has_rescueadoptionloc   has_tcop5zicjudxo   has_presley  \
      0           0          0                       0                   0             0
      1           0          0                       0                   0             0
      2           0          0                       0                   0             0
      3           0          0                       0                   0             0
      4           0          0                       0                   0             0

         has_sel   has_damnwa   has_hitter   has_humanitarian   has_antonio   …  \
      0        0            0            0                  0             0   …
      1        0            0            0                  0             0   …
      2        0            0            0                  0             0   …
      3        0            0            0                  0             0   …
      4        0            0            0                  0             0   …

         has_stuffin   has_declin   has_vermilion   has_pjnet   has_convert   has_cfc  \
      0            0            0               0           0             0         0
      1            0            0               0           0             0         0
      2            0            0               0           0             0         0
      3            0            0               0           0             0         0
      4            0            0               0           0             0         0

         has_homeland   has_nh   has_bruv   target
      0             0        0          0        0
      1             0        0          0        1
      2             0        0          0        1
      3             0        0          0        1
      4             0        0          0        0

      [5 rows x 10340 columns]
```

1.1.3   Compute model parameters

I used the closed-form solution for the parameters for a Bernoulli Naive Bayes model. This results in an analytical solution for the parameters psis which helps calculate P(x|y) and parameters phis which helps calculate P(y).

I also used Laplace smoothing to make sure that words outside of the training/model vocabulary will give a P(x|y) probability of 0.5 (no bias towards either 0 or 1).

```python
[6]: # Compute model parameters
     # Code adapted from lecture notes
     import numpy as np
     alpha = 1
     n = train.shape[0]
     d = train.shape[1] - 1
     K = 2

     psis = np.zeros([K,d])
```

```
phis = np.zeros([K])

y_train = train["target"]
X_train = train.drop(columns=["target"])

for k in range(K):
    X_k = X_train[y_train == k]
    psis[k] = (X_k.sum(axis=0) + alpha)/ (2*alpha + X_k.shape[0])
    phis[k] = X_k.shape[0]/ float(n)

print(phis)
```

[0.56746106 0.43253894]

1.1.4 Compute predictions from model

Using the modeled distribution of inputs for a given class P(x|y=k) and the modeled distribution of the class P(y=k), I can compute the predicted P(y=k|x) for each class k by multiplying these two values. To avoid numerical instability, I added the logs of these probabilities.

Then, I determined that the predicted class for a given input should be the class that has the highest probability for that input.

```
[7]: # Compute predicitons from model
     # Code adapted from lecture notes
     # And code adapted from ChatGPT (2025)
     # Prompt: "how can i make sure i dont crash due to memory in this funciton..."
     def nb_predictions(x, psis, phis):
         x = x.to_numpy()

         n, d = x.shape
         K = psis.shape[0]

         logpy = np.log(phis).reshape([K])
         logpyx = np.zeros((K, n))

         psis = psis.clip(1e-14, 1-1e-14)
         log_psis = np.log(psis)
         log_one_minus_psis = np.log(1-psis)

         for i in range(n):
             xi = x[i]   # shape (d,)
             logpxy = xi * log_psis + (1-xi) * log_one_minus_psis   # shape (K,d)
             logpyx[:, i] = logpxy.sum(axis=1) + logpy

         return logpyx.argmax(axis=0), logpyx

     idx, logpyx = nb_predictions(X_train, psis, phis)
```

```
print(idx[:10])
```

```
[1 0 1 0 0 1 1 0 0 1]
```

1.2 Compare models

I applied the model to calculate the predicted class of the validation/development set. Then I used the F1 score to evaluate the classification model's precision and recall.

```
[8]:  # Apply to validation set
      X_val = val.drop(columns=["target"])
      idx_val, logpyx = nb_predictions(X_val, psis, phis)
      print(idx_val[:10])
```

```
[0 0 0 0 0 0 0 0 0 1]
```

```
[9]:  # Evaluate on validation set
      from sklearn.metrics import f1_score
      f1 = f1_score(idx_val, val["target"])
      print("F1 score:", f1)
```

```
F1 score: 0.2289348171701113
```

```
[14]:  # Compare models to HW2
```

Recall, the best model between the N-gram and bag of words approach had an F1 score of 0.743139407244786. The best model of overall was with the M=3 feature set using L2 regularization with C=1 with the bag of words model.

When the model was rebuilt using the entire training data and submitted to Kaggle, the F1 score rose to 0.7537796976241901.

By comparison, the Naive-Bayes approach had an F1 score of 0.2289348171701113. Thus, it seems that the Naive-Bayes approach is significantly worse at generalizing and predicting if tweets are disasters on new data.

I found this low score so surprising that I decided to check if my model was overfitting.

```
[11]:  # See F1 score for testing data
      from sklearn.metrics import f1_score
      f1 = f1_score(idx, train["target"])
      print("Testing F1 score:", f1)
```

```
Testing F1 score: 0.8132992327365729
```

Because the testing F1 score is very high while validation F1 score is low, it is observed that my NB model is overfitting on the testing data. I think one reason is for this is that I included all words in the vocabulary as features, even if it only appears once in the training data. This is likely created too many features compared to samples and induced overfitting on these poorly represented single-occurence words.

A better alternative is to only take the top 1000 highest frequency words, which is better done through the sklearn CountVectorizer library.

1.2.1 Comparison of generative and discriminative models

The previous bag of words and N-gram models are discriminative models, where they explicitly determine P(y|x) by modeling the data generative function between the input x and label y. This Bernoulli Naive-Bayes model is a generative model, where it models the distribution of inputs for each label. It is still able to perform discriminative-like predictions using the Bayes Rule (this is different from the NB assumption).

One positive of generative models is that it can perform discriminative-model-like predictions while also performing its task to generate new inputs with high probability of class k. Discriminative models do not have the capability to generate new inputs.

An additional positive is that generative models are more descriptive of the dataset. Discriminative models only classify classes of datasets by determining decision boundaries between classes. On the other hand, generative models show the distributions of the data. This means it can capture clustering of data within classes, showing their means and variances. This information about the distribution can help with more tasks like data imputation, handling missing data, and anomaly detection. Discriminative models do not have this additional capability.

On a performance level, discriminative models do better when there is more data close to the true decision boundary. So the difference in predictive power of a discriminative vs generative supervised learning model will depend on the dataset and underlying decision boundaries. For example, with less data, generative models will perform better.

On a negative for generative models, they require stronger assumptions to realistically learn distributions. Thus, there is a higher risk of inaccuracies coming for inappropriate, strong assumptions. The comparison of assumptions will be discussed in the next section.

Additionally, discriminative models are easier to implement and understand on an interpretative level for the task of classification prediction. We can simply look at the model parameter magnitudes to determine the most important and least important features with a discrinative model. By comparison, a generative model requires more steps (Bayes Rule) to perform predictive classification tasks. These complexities become harder to track and compare as we grow from a binary classification to a multiclassification task. By comparison, comparing magnitudes of parameters in a discrimnative classification is more straight-forward.

1.2.2 Asssumptions of generative and discriminative models

The Naive Bayes assumption is that the features for a given label can be considered independent from eachother. In this case, the NB assumption manifests as the presence of some specific word for non-disaster tweets like "happy" is independent from the word "help".

There is also an assumption of independence in disciminative models. Here, independence means that two different features each contribute an independent effect on the decision to classify a tweet as disaster or not. While this sounds very similar, in the N-gram and bag of words model, we were able to construct our features the a unique collection of words. For example, a vocabulary of 3 words would cause features to have the form of (0,0,0), (0,0,1), (0,1,0), (0,1,1),... (1,1,1). This means that there is no assumption of individual words having independent effect on the classification from eachother.

In terms of efficiency, the NB assumption is definitely important and very efficient for memory. In the example above, we saw that a vocabulary of 3 requires 9 different features for each combination

of the word being present or absent. By using the NB assumption, we can represent the same vocabulary of 3 words with only 3 features. With each feature being whether each word is present or not, independently. In a general form, the NB assumption allows reduction of parameters from $2^{(d-1)}$ to ~Kd, where d is the vocabulary size and K is the number of classes. Using the NB assumption allows the inclusion of more vocabulary while maintaining memory constraints.

In terms of validity, there is some appropriateness. For most unrelated words like "girl" and "computer", assuming independence of these word occurences is valid. However, we can also construct cases where words together create different meanings. For example, "not" and "happy" is very different from the independent occurence of "not" and "happy". Some words even change semantic meaning in a dependent way to another word. For example "bank" means different things when paired with "river" versus "Chase" (thus showing dependence between words). In general, independence is a poor assumption in natural language text since many words have context around other words.