CS5785 Homework 2

The homework is generally split into programming exercises and written exercises.

This homework is due on **Sept 30, 2021 at 11:59 PM ET**. Upload your homework to Gradescope (Canvas->Gradescope). There are two assignments for this homework in Gradescope. Please note a complete submission should include:

1. A write-up as a single `.pdf` file. ⟶ Submit to "Homework 2- Write Up".
2. Source code for all of your experiments (AND figures) zipped into a single .zip file, in `.py` files if you use Python or `.ipynb` files if you use the IPython Notebook. If you use some other language, include all build scripts necessary to build and run your project along with instructions on how to compile and run your code. **If you use the IPython Notebook to create any graphs, please make sure you also include them.** ⟶ Submit to "Homework 2- Code".

The write-up should contain a general summary of what you did, how well your solution works, any insights you found, etc. On the cover page, include the class name, homework number, and team member names. You are responsible for submitting clear, organized answers to the questions. You could use online LaTeX templates from Overleaf, under "Homework Assignment" or "Project / Lab Report".

Please include all relevant information for a question, including text response, equations, figures, graphs, output, etc. If you include graphs, be sure to include the source code that generated them. Please pay attention to Canvas for relevant information regarding updates, tips, and policy changes. You are encouraged (but not required) to work in groups of 2.

IF YOU NEED HELP

There are several strategies available to you.

- If you get stuck, we encourage you to post a question on the Discussions section of Canvas. That way, your solutions will be available to other students in the class.
- The professor and TAs offer office hours, which are a great way to get some one-on-one help.
- You are allowed to use well known libraries such as `scikit-learn`, `scikit-image`, `numpy`, `scipy`, etc. for this assignment (including implementations of machine learning algorithms), unless we explicitly say that you cannot in a particular question. Any reference or copy of public code repositories should be properly cited in your submission (examples include Github, Wikipedia, Blogs).

PROGRAMMING EXERCISES

1. **Binary Classification on Text Data.** In this problem, you will implement several machine learning techniques from the class to perform classification on text data. Throughout the problem, we will be working on the NLP with Disaster Tweets Kaggle competition, where the task is to predict whether or not a tweet is about a real disaster.

   (a) **Download the data.** Download the training and test data from Kaggle, and answer the following questions: (1) how many training and test data points are there? and (2) what percentage of the training tweets are of real disasters, and what percentage is not? Note that the meaning of each column is explained in the data description on Kaggle.

   (b) **Split the training data.** Since we do not know the correct values of labels in the test data, we will split the **training data from Kaggle** into a *training* set and a *development* set (a *development* set is a held out subset of the labeled data that we set aside in order to fine-tune models, before evaluating the best model(s) on the test data). Randomly choose 70% of the data points in the training data as the *training* set, and the remaining 30% of the data as the *development* set. Throughout the rest of this problem we will keep these two sets fixed. The idea is that we will train different models on the *training* set, and compare their performance on the *development* set, in order to decide what to submit to Kaggle.

   (c) **Preprocess the data.** Since the data consists of tweets, they may contain significant amounts of noise and unprocessed content. You **may or may not** want to do one or all of the following. Explain the reasons for each of your decision (**why or why not**).

   - Convert all the words to lowercase.
   - Lemmatize all the words (i.e., convert every word to its root so that all of "running," "run," and "runs" are converted to "run" and and all of "good," "well," "better," and "best" are converted to "good"; this is easily done using nltk.stem).
   - Strip punctuation.
   - Strip the stop words, e.g., "the", "and", "or".
   - Strip @ and urls. (It's Twitter.)
   - Something else? Tell us about it.

   (d) **Bag of words model.** The next task is to extract features in order to represent each tweet using the binary "bag of words" model, as discussed in lectures. The idea is to build a vocabulary of the words appearing in the dataset, and then to represent each tweet by a feature vector x whose length is the same as the size of the vocabulary, where $x_i = 1$ if the i'th vocabulary word appears in that tweet, and $x_i = 0$ otherwise. In order to build the vocabulary, you should choose some threshold M, and only include words that appear in at least k different tweets; this is important both to avoid run-time and memory issues, and to avoid noisy/unreliable features that can hurt learning. Decide on an appropriate threshold M, and discuss how you made this decision. Then, build the bag of words feature vectors for both the *training* and *development* sets, and report the total number of features in these vectors.

   In order to construct these features, we suggest using the CountVectorizer class in `sklearn`. A couple of notes on using this function: (1) you should set the option "`binary=True`" in order

to ensure that the feature vectors are binary; and (2) you can use the option "`min_df=M`" in order to only include in the vocabulary words that appear in at least M different tweets. Finally, make sure you fit CountVectorizer only once on your training set and use the same instance to process both your training and development sets (don't refit it on your development set a second time).

Important: at this point you should only be constructing feature vectors for each data point using the text in the "text" column. You should ignore the "keyword" and "location" columns for now.

(e) **Logistic regression.** In this question, we will be training logistic regression models using bag of words feature vectors obtained in part (d). We will use the $F1$-score as the evaluation metric.

Note that the $F1$-score, also known as F-score, is the harmonic mean of precision and recall. Recall that precision and recall are:

$$\text{precision} = \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false positives}} \quad \text{recall} = \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false negatives}}.$$

$F1$-score is the **harmonic mean** (or, see it as a weighted average) of precision and recall:

$$F1 = \frac{2}{\text{precision}^{-1} \cdot \text{recall}^{-1}} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

We use $F1$-score because it gives a more comprehensive view of classifier performance than accuracy. For more information on this metric see F1-score.

We ask you to train the following classifiers. We suggest using the LogisticRegression implementation in `sklearn`.

i. Train a logistic regression model without regularization terms. You will notice that the default `sklearn` logistic regression utilizes L2 regularization. You can turn off L2 regularization by changing the `penalty` parameter. Report the F1 score in your *training* and in your *development* set. Comment on whether you observe any issues with overfitting or underfitting.

ii. Train a logistic regression model with L1 regularization. `Sklearn` provides some good examples for implementation. Report the performance on both the *training* and the *development* sets.

iii. Similarly, train a logistic regression model with L2 regularization. Report the performance on the *training* and the *development* sets.

iv. Which one of the three classifiers performed the best on your *training* and *development* set? Did you observe any overfitting and did regularization help reduce it? Support your answers with the classifier performance you got.

v. Inspect the weight vector of the classifier with L1 regularization (in other words, look at the θ you got after training). You can access the weight vector of the trained model using the `coef_` attribute of a LogisticRegression instance. What are the most important words for deciding whether a tweet is about a real disaster or not?

(f) **Bernoulli Naive Bayes.** Implement a Bernoulli Naive Bayes classifier to predict the probability of whether each tweet is about a real disaster. Train this classifier on the *training* set, and report its $F1$-score on the *development* set.

Important: For this question you should implement the classifier yourself similar to what was shown in class, without using any existing machine learning libraries such as `sklearn`. You may only use basic libraries such as `numpy`.

As you work on this problem, you may find that some words in the vocabulary occur in the *development* set but are not in the *training* set. As a result, the standard Naive Bayes model learns to assign them an occurrence probability of zero, which becomes problematic when we observe this "zero probability" event on our *development* set.

The solution to this problem is a form of regularization called Laplace smoothing or additive smoothing. The idea is to use "pseudo-counts", i.e. to increment the number of times we have seen each word or document by some number of "virtual" occurrences α. Thus, the Naive Bayes model will behave as if every word or document has been seen at least α times.

More formally, the ψ_{jk} parameter of Bernoulli Naive Bayes is the probability of observing word j within class k. Its normal maximum likelihood estimate is

$$\psi_{jk} = \frac{n_{jk}}{n_k},$$

where n_k is the number of documents of class k and n_{jk} is the number of documents of class k that contain word j. In Laplace smoothing, we increment each counter n_{jk} by α (thus we count each word an extra α times), and the resulting estimate for ψ_{jk} becomes:

$$\psi_{jk} = \frac{n_{jk} + \alpha}{n_k + 2\alpha}.$$

It's normal to take $\alpha = 1$.

(g) **Model comparison.** You just implemented a generative classifier and a discriminative classifier. Reflect on the following:

- Which model performed the best in predicting whether a tweet is of a real disaster or not? Include your performance metric in your response. Comment on the pros and cons of using generative vs discriminative models.
- Think about the assumptions that Naive Bayes makes. How are the assumptions different from logistic regressions? Discuss whether it's valid and efficient to use Bernoulli Naive Bayes classifier for natural language texts.

(h) **N-gram model.** The N-gram model is similar to the bag of words model, but instead of using individual words we use N-grams, which are contiguous sequences of words. For example, using $N = 2$, we would says that the text "Alice fell down the rabbit hole" consists of the sequence of 2-grams: ["Alice fell", "fell down", "down the", "the rabbit", "rabbit hole"], and the following sequence of 1-grams: ["Alice", "fell", "down", "the", "rabbit", "hole"]. All eleven of these symbols may be included in the vocabulary, and the feature vector x is defined according to $x_i = 1$ if the i'th vocabulary symbol occurs in the tweet, and $x_i = 0$ otherwise. Using $N = 2$, construct feature representations of the tweets in the *training* and *development*

tweets. Again, you should choose a threshold M, and only include symbols in the vocabulary that occur in at least M different tweets in the *training* set. Discuss how you chose the threshold M, and report the total number of 1-grams and 2-grams in your vocabulary. In addition, take 10 2-grams from your vocabulary, and print them out.

Then, implement a logistic regression and a Bernoulli classifier to train on 2-grams. You may reuse the code in (e) and (f). You may also choose to use or not use a regularization term, depending on what you got from (e). Report your results on *training* and *development* set. Do these results differ significantly from those using the bag of words model? Discuss what this implies about the task.

Again, we suggest using CountVectorizer to construct these features. In order to include both 1-gram and 2-gram features, you can set `ngram_range=(1,2)`. Note also that in this case, since there are probably many different 2-grams in the dataset, it is especially important carefully set `min_df` in order to avoid run-time and memory issues.

(5)

(i) **Determine performance with the *test* set** Re-build your feature vectors and re-train your preferred classifier (either bag of word or n-gram using either logistic regression or Bernoulli naive bayes) using the entire Kaggle training data (*i.e.* using all of the data in both the *training* and *development* sets). Then, test it on the Kaggle test data. Submit your results to Kaggle, and report the resulting $F1$-score on the test data, as reported by Kaggle. Was this lower or higher than you expected? Discuss why it might be lower or higher than your expectation.

WRITTEN EXERCISES

1. **Naive Bayes with Binary Features.** You are working at a hospital, and tasked with developing a classifier to predict whether patients have COVID based on their symptoms. Suppose that you have access to the following information about the distribution of patients entering the hospital:

   - 10% of the patients have COVID, and 90% do not
   - Out of the patients who have COVID:
     - 75% have both fever and coughing
     - 5% have fever but no coughing
     - 5% have coughing but no fever
     - 15% have neither fever nor coughing
   - Out of the patients who do not have COVID:
     - 4% have both fever and coughing
     - 1% have coughing but no fever
     - 1% have fever but no coughing
     - 94% have neither fever nor coughing

We can formulate this as a machine learning problem by modeling the symptoms via features $x = (x_1, x_2) \in \{0,1\}^2$, where x_1 is a binary indicator of whether the patient has fever and x_2 is a binary indicator of whether they are coughing, and the target y equals 1 if they have COVID and 0 otherwise.

Hint: You may want to remind yourself about Bayes' theorem before solving this problem.

(a) Suppose the hospital is presented with a patient who has both fever and coughing. According to the full data distribution presented above, what is the probability that the patient doesn't have COVID?

(b) Next, suppose we have trained a Naive Bayes classifier using a very large dataset sampled IID from the data distribution and we have found the best Naive Bayes model that approximates this data distribution. What is the probability that the above patient doesn't have COVID according to the naive Bayes model?

(c) Do the above approaches give significantly different probabilities that the patient doesn't have COVID? If so, why? Which approach do you think gives a more reasonable estimate? You should relate your answer to the different assumptions made by the two approaches.

2. **Categorical Naive Bayes.** Suppose we are working with a dataset $\mathscr{D} = \{x^{(i)}, y^{(i)} \mid y = 1,2,...,n\}$ in which the d-dimensional inputs x are *categorical*: each feature x_j takes one of L possible values: $x_j^{(i)} \in \{1,2,...,L\}$ for all i, j. If $L = 2$, then the features look like the binary bag-of-words vectors that we have seen in class; in this example, however, the features can take more than just two values. We also assume that the target y represents one of K possible classes: $y \in \{1,2,...,K\}$

In the Categorical Naive Bayes algorithm, we model this data via a probabilistic model $P_\theta(x, y)$.

- The distribution $P_\theta(y)$ is Categorical with parameters $\phi = (\phi_1, ..., \phi_K)$ and

$$P_\theta(y = k) = \phi_k$$

- The distribution of each feature x_j conditioned on $y = k$ is a Categorical distribution with parameters $\psi_{jk} = (\psi_{jk1}, ..., \psi_{jkL})$, where

$$P_\theta(x_j = \ell | y = k) = \psi_{jk\ell}.$$

The distribution over a vector of features x is given by

$$P_\theta(x|y = k) = \prod_{j=1}^{d} P_\theta(x_j|y = k),$$

which is just the Naive Bayes factorization of $P_\theta(x|y = k)$.

In other words, the prior distribution $P_\theta(y)$ in this model is the same as in Bernoulli Naive Bayes. The distribution $P_\theta(x|y = k)$ is a product of Categorical distributions, whereas in Bernoulli Naive Bayes it was the product of Bernoulli distributions.

The total set of parameters of this model is $\theta = (\phi_1, ...\phi_K, \psi_{111}, ...\psi_{dKL})$. We learn the parameters via maximum likelihood:

$$\max_\theta \frac{1}{n} \sum_{i=1}^{n} \log P_\theta(x^{(i)}, y^{(i)})$$

(a) Show that the maximum likelihood estimate for the parameters ϕ is

$$\phi^* = \frac{n_k}{n},$$

where n_k is the number of data points with class k.

(b) Show that the maximum likelihood estimate for the parameters $\psi_{jk\ell}$ is

$$\psi^*_{jk\ell} = \frac{n_{jk\ell}}{n_k},$$

where $n_{jk\ell}$ is the number of data points with class k for which the j-th feature equals ℓ.