PROGRAMMING EXERCISES

1. **Data generating distribution and convergence of linear regression.**

   (a) Generate a synthetic dataset with the following data generating process:

   $$Y \sim \alpha + \beta X + \epsilon,$$

   where X is drawn from a normal distribution with mean 168 and standard deviation 30, and ϵ is drawn from a normal distribution with mean 0 and standard deviation 20. **Let $\alpha = 20$, and $\beta = 0.5$.**

   [Hint: use function 'numpy.random.normal', and X and Y should be a column vector. Think carefully what the shape of X and epsilon should be when generating their values.]

   (b) Run 5 linear regressions with sample sizes $n = 10^2, 10^3, 10^4, 10^5, 10^6$, respectively. Report the coefficients and R^2 of each run.

   (c) What do you observe in part (b)? Do the coefficients of your linear regressions converge to the α and β? Does your R^2 converge to 0? If not, what number does it converge to?

   (d) Recall that the analytical form of R^2 is given by

   $$R^2 = 1 - \frac{\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{n}(y^{(i)} - \bar{y})^2},$$

   where $\bar{y} = \sum_{i=1}^{n} \frac{1}{n} y^{(i)}$ is the empirical mean of the labels. Recall that the sample variance of a random variable X is calculated by $\text{var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - \bar{x})^2$, where $\bar{x}$ is the sample mean. Assuming that the coefficient of the linear regression converges to those in true data generating distribution, establish that when the data generation is linear, R^2 of a linear regression converge to $1 - \frac{\text{var}(\epsilon)}{\text{var}(Y)}$.

   (e)Show that the coefficients of the linear regression converges to α and β in the data generating process.

   (f) In the example above, you have established that if 1) the data generating process is linear and 2) we fit the data using linear regression, then the R^2 converges to $1 - \frac{\text{var}(\epsilon)}{\text{var}(Y)}$. Given this information,

   do you think there exists another model such that you obtain a better (higher) R^2 asymptotically (when the data that you get goes to infinity)? If so, show such an example. If not, please argue why. (You do not need to show this mathematically.)

   (g) Based on this exercise, what are some important characteristics of the data generating distribution necessary for the presence of a "good" model?

2. **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:

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

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 there exists some words that only appear in some classes but never in other classes in the *training* set. This might lead the model into thinking it is impossible for the word to appear in those other classes it never appears in in the *training* set, even though that word might actually appear in those classes in the *development* set.

Concretely, suppose word j never appears in class k in the *training* set. Since the maximum likelihood estimate of ψ_{jk} is $\frac{n_{jk}}{n_k}$ 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, during training, we set

$$P(x_j = 1 | y = k) = \psi_{jk} = \frac{n_{jk}}{n_k} = 0$$

Then, any bag of words x that contains word j, i.e., $x_j = 1$, will have

$$P_\theta(x|y=k) = \prod_{j=1}^{d} P(x_j|y=k) = 0$$

We obviously do not want this, because this implies we only assign positive probability to the bag of words whose words have appeared in the class k at least once.

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 class k document with word j by some number of "virtual" occurrences α, and increment the number of times we have seen class k document in general by 2α (so half of the pseudo documents have word j in it). Mathematically,

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

Thus, the Naive Bayes model will behave as if every word or document has been seen at least α times.

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 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. To get 2-grams, you can set `ngram_range=(2,2)` for CountVectorizer. 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.

(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.