CS5785 Homework 2

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

This homework is due on **October 22, 2020 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. 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 use 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 or 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 *train* set and a *dev* set (a *dev* set is a 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 *train* set, and the remaining 30% of the data as the *dev* 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 *train* set, and compare their performance on the *dev* 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 garbage. 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".
   - 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 *train* and *dev* 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.

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) **Implement a naive Bayes classifier.** Implement a Naive Bayes classifier, using the Bernoulli Naive Bayes model as discussed in lectures, in order to predict the probability that each tweet is of a real disaster or not. Train this classifier on the *train* set, and report its mean $F1$-score on the *dev* set.

Note that the $F1$-score, also known as F-score, is defined according to

$$F1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}},$$

where

$$\text{precision} = \frac{\text{TPR}}{\text{TPR} + \text{FPR}}$$
$$\text{recall} = \frac{\text{TPR}}{\text{TPR} + \text{FNR}}.$$

For more information on this metric see F1-score.

Important: For this question you should implement the classifier yourself, without using any existing machine learning libraries such as `sklearn`. Using basic libraries such as `numpy` is acceptable.

(f) **Logistic regression prediction.** Train a logistic regression model using the bag of words feature vectors on the *train* set, and report its mean $F1$-score on the *dev* set. Inspecting the weight vector of the logistic regression, what are the most important words for deciding whether a tweet is about a real disaster or not?

For this question, we suggest using the LogisticRegression implementation in `sklearn`. You can access the weight vector of the trained model using the `coef_` attribute.

(g) **Linear SVM prediction.** Train a linear SVM model on the bag of words feature vectors, for a variety of regularization hyperparameter values. For each value of the hyperparameter, train the classifier on the *train* set, and compute its $F1$-score on the *dev* set. Create a plot of the resultant classification $F1$-scores on the *dev* set for different hyperparameter values. What hyperparameter value results in the best classification perormance, and how does this best linear SVM classifier compare with the logistic regression classifier from the previous section? Next, inspecting the weight vector from the SVM classifier, what are the most important words for deciding whether a tweet is about a real disaster or not? Are these the same or different from the most important words for logistic regression?

For this question, we suggest using the LinearSVC implementation in `sklearn`, which is an efficient implementation of the linear SVM classifier using the primal approach discussed in lectures. Note that the regularization hyperparameter value is referred to as C for this implementation. If you use this implementation, we suggest trying the following C values:

[0.01, 0.1, 1.0, 10.0, 100.0]. As with the `sklearn` logistic regression model, you can access the weight vector of the trained model using the `coef_` attribute.

(h) **Non-linear SVM prediction.** Repeat the previous question, using a non-linear SVM model with a Gaussian kernel. Note that in this case, since we are using a kernelized SVM there is no weight vector for the bag of words features, so you should skip the part about inspecting the weight vector.

For this question, we suggest using the SVC implementation in `sklearn`, which is an efficient implementation of the kernelized SVM classifier using the dual approach discussed in lectures. Again, the regularization hyperparameter value is referred to as C for this implementation. If you use this implementation, we again suggest trying the following C values: [0.01, 0.1, 1.0, 10.0, 100.0]. Note also that they refer to the Gaussian kernel as the RBF kernel, and the hyperparameter we referred to as σ they refer to as `gamma`. You may use the default value for `gamma`, although you may be able to achieve better performance by experimenting with different values.

(i) **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 *train* and *dev* 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 *train* set. Discuss how you chose the threshold M, and report the total number of 1-grams and 2-grams in your vocabulary. In addition, randomly sample 10 2-grams from your vocabulary, and print them out. Then, repeat parts (e)-(h), and report the results. 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.

(j) **Incorporating the additional columns.** In what we have done so far, we have only used the words in the "text" column for features. In addition, however, the dataset contains a "keyword" column, and a "location" column. Come up with a strategy for incorporating the information in these columns into your feature vectors. Two examples of how you might do this would be

- Append the words appearing in these columns (if any) to the tweet, and re-calculate the bag of words or N-grams features.
- Create new categorical features corresponding to the different values these columns could take, and append these to the bag of words or N-grams features.

In doing this, you could choose to either use bag of words or N-grams for the text features,

depending on which you found to be more appropriate based on your prior analysis. Discuss your chosen strategy, and why you chose it. In addition, repeat parts (f)-(h) using your chosen strategy to construct feature vectors, and report the results. Discuss how and why the results differ from those computed without using these columns.

(k) **Finalizing your model.** Out of all of the joint approaches you have considered so far, for both constructing feature vectors and training a classifier on these vectors, which do you believe is the best for accurately predicting whether a tweet is of a real disaster or not? Justify your choice. Then, using your chosen best approach, re-build your feature vectors and re-train your classifier using the entire Kaggle training data (*i.e.* using all of the data in both the *train* and *dev* sets), and 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.

(l) **Reflecting on interpretability.** Suppose that you were constructing a model for this task as part of a consulting job, where you not only cared about classification performance, but wanted an interpetable model, such that you could explain to your clients how it made its decisions, and they could trust the model. Would you still choose the same approach, or a different one? Discuss why or why not, and which approach you think would be best in this setting.

WRITTEN EXERCISES

1. **Maximum-margin classifiers.** Suppose we are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label.

X_1	X_2	Y
3	4	Red
2	2	Red
4	4	Red
1	4	Red
2	1	Blue
4	3	Blue
4	1	Blue

   (a) Sketch the observations and the maximum-margin separating hyperplane.

   (b) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1, and β_2.

   (c) On your sketch, indicate the margin for the maximal margin hyperplane.

   (d) Indicate the support vectors for the maximal margin classifier.

   (e) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

   (f) Sketch a hyperplane that separates the data, but is not the maximum-margin separating hyperplane. Provide the equation for this hyperplane.

   (g) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

2. **Naive Bayes with binary features.** You are working at a hospital, and tasked with developing a classifier to predict whether patients of some hospital have the flu or not based on their symptoms. Suppose that you have access to the following information about the distribution of patients entering the hospital:

- 20% of the patients have the flu, and 80% do not
- Out of the patients who have the flu:
  - 75% have both coughing and sneezing
  - 5% have sneezing but no coughing
  - 5% have coughing but no sneezing
  - 15% have neither coughing nor sneezing
- Out of the patients who do not have the flu:
  - 4% have both coughing and sneezing
  - 1% have sneezing but no coughing
  - 1% have coughing but no sneezing
  - 94% have neither coughing nor sneezing

(a) Suppose the hospital is presented with a patient who has both coughing and sneezing. According to the full generative model presented above, what is the probability that the patient doesn't have the flu?

(b) Now, suppose we wish to train a naive Bayes classifier using the above data. What is the probability that the above patient doesn't have the flu according to a naive Bayes model?

(c) Do the above approaches give significantly different probabilities that the above patient doesn't have the flu? 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.

3. **Kernelized gradient descent.** In this problem, we will walk you through an alternative method for performing kernelized ridge regression, via gradient descent. Recall that, given feature map ϕ, the loss function of kernel ridge regression is given by

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}\left(y_i - \theta^T\phi(x_i)\right)^2 + \frac{\lambda}{2}\|\theta\|^2, \tag{1}$$

and that the update rule for minimizing equation 1 via gradient descent is

$$\theta_{t+1} = \theta_t - \alpha\left(\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \theta_t^T\phi(x_i)\right)\phi(x_i) + \lambda\theta_t\right), \tag{2}$$

where α is the learning rate. The idea of the kernelized gradient descent approach is that instead of storing the explicit parameter value θ_t at each time step, we will represent it in the form

$$\theta_t = \sum_{i=1}^{n}\beta_i^{(t)}\phi(x_i), \tag{3}$$

and just store the vector $\beta^{(t)} = (\beta_1^{(t)}, \beta_2^{(t)}, \ldots, \beta_n^{(t)})$, which we will update at each time step. Note that we could initialize $\beta^{(0)}$ randomly, or just set it to zero.

(a) Suppose that θ_t takes the form as in equation 3. Show that the parameter value θ_{t+1}, obtained from the gradient descent update as in equation 2, also takes the form of equation 3, with an updated vector of constants $\beta^{(t+1)}$. Specifically, defining the matrix L according to $L_{i,j} = \phi(x_i)^T\phi(x_j)$, provide an equation for $\beta^{(t+1)}$ in terms of $\beta^{(t)}$, L, y, and α. Note that for full credit, this equation shouldn't explicitly involve the feature map ϕ other than through L.

(b) Let K be the kernel function, which satisfies $K(x,x') = \phi(x)^T\phi(x')$. Suppose that θ_t takes the form as in equation 3, and let x_{test} be some test data point. Provide an equation for the predicted value $\hat{y} = \theta_t{}^T x_{\text{test}}$, in terms of $\beta^{(t)}$, K, x_{test}, and $\{x_1,\ldots,x_n\}$. Note that for full credit, this equation shouldn't explicitly involve the feature map ϕ other than through K.

(c) Suppose that ϕ is an infinite-dimensional feature map, but that $K(x,x')$ is easy to compute. Provide an explanation of why this does not prevent us from performing gradient descent and computing predicted values $\hat{y} = \theta^T\phi(x)$, even though θ and $\phi(x)$ are infinite-dimensional.