CS5785 Homework 3

The homework is generally split into programming exercises and written exercises.
This homework is due on **November 9, 2018 at 11:59 PM EST**. Upload your homework to CMS. Please upload the submission as a single `.zip` file. A complete submission should include:

1. A write-up as a single `.pdf` file
2. Source code for all of your experiments (AND figures) 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.

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" and and "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 Slack 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 ever get stuck, the best way is to ask on Slack. That way, your solutions will be available to the other students in the class.
- Your instructor and TAs will offer office hours[1], 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. in this assignment. Any reference or copy of public code repositories should be properly cited in your submission (examples include Github, Wikipedia, Blogs).

[1] https://cornelltech.github.io/cs5785-fall-2017/contact.html

PROGRAMMING EXERCISES

1. **Sentiment analysis of online reviews.**

   In this assignment you will use several machine learning techniques from the class to identify and extract subjective information in online reviews. Specifically, the task for this assignment is **sentiment analysis**. According to *Wikipedia*, sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document. It has been shown that people's attitudes are largely manifested in the language they adopt. This assignment will walk you through the mystery and help you better understand our posts online!

   Important: Use your own implementations for this question. Any bag-of-words or PCA implementation is **NOT** allowed.

   (a) Download Sentiment Labelled Sentences Data Set. There are three data files under the root folder. *yelp_labelled.txt*, *amazon_cells_labelled.txt* and *imdb_labelled.txt*. Parse each file with the specifications in *readme.txt*. Are the labels balanced? If not, what's the ratio between the two labels? Explain how you process these files.

   (b) **Pick your preprocessing strategy.** Since these sentences are online reviews, 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**).

   - Lowercase all of the words.
   - Lemmatization of 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.

   (c) **Split training and testing set.** In this assignment, for each file, please use the first 400 instances for each label as the *training set* and the remaining 100 instances as *testing set*. In total, there are 2400 reviews for training and 600 reviews for testing.

   (d) **Bag of Words model.** Extract features and then represent each review using bag of words model, i.e., every word in the review becomes its own element in a feature vector. In order to do this, first, make one pass through all the reviews in the *training set* (**Explain why** we can't use *testing set* at this point) and build a dictionary of unique words. Then, make another pass through the review in both the *training set* and *testing set* and count up the occurrences of each word in your dictionary. The ith element of a review's feature vector is the number of occurrences of the ith dictionary word in the review. Implement the bag of words model and report feature vectors of any two reviews in the training set.

   (e) **Pick your postprocessing strategy.** Since the vast majority of English words will not appear in most of the reviews, most of the feature vector elements will be 0. This suggests that we need a postprocessing or normalization strategy that combats the huge variance of the elements in the feature vector. You may want to use one of the following strategies. Whatever choices you make, explain **why** you made the decision.

   - *log-normalization.* For each element of the feature vector x, transform it into $f(x) = log(x+1)$.

- *l*1 *normalization.* Normalize the *l*1 norm of the feature vector, $\hat{\mathbf{x}} = \frac{\mathbf{x}}{|\mathbf{x}|}$.
- *l*2 *normalization.* Normalize the *l*2 norm of the feature vector, $\hat{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$.
- *Standardize* the data by subtracting the mean and dividing by the variance.

(f) **Sentiment prediction.** Train a logistic regression model (*you can use existing packages here*) on the training set and test on the testing set. Report the classification accuracy and confusion matrix. Inspecting the weight vector of the logistic regression, what are the words that play the most important roles in deciding the sentiment of the reviews? Repeat this with a Naive Bayes classifier and compare performance.

(g) **N-gram model.** Similar to the bag of words model, but now you build up a dictionary of n-grams, which are contiguous sequences of words. For example, "Alice fell down the rabbit hole" would then map to the 2-grams sequence: ["Alice fell", "fell down", "down the", "the rabbit", "rabbit hole"], and all five of those symbols would be members of the n-gram dictionary. Try $n = 2$, repeat (d)-(g) and report your results.

(h) **PCA for bag of words model.** The features in the bag of words model have large redundancy. Implement PCA to reduce the dimension of features calculated in (e) to 10, 50 and 100 respectively. Using these lower-dimensional feature vectors and repeat (f), (g). Report corresponding clustering and classification results. (**Note:** You should implement PCA yourself, but you can use **numpy.svd** or some other SVD package. Feel free to double-check your PCA implementation against an existing one)

(i) **Algorithms comparison and analysis.** According to the above results, compare the performances of *bag of words*, *2-gram* and *PCA for bag of words*. Which method performs best in the prediction task and why? What do you learn about the language that people use in online reviews (e.g., expressions that will make the posts positive/negative)? *Hint:* Inspect the clustering results and the weights learned from logistic regression.

2. **Clustering for text analysis.** In this problem, you will analyze all the articles from the journal Science in the year 2000. (Thanks to JSTOR for providing the data.) Many of the parameters of this analysis will be left for you to decide. For these files you will need to use `science2k-vocab.npy` and `science2k-titles.npy`, which are vectors of terms and titles respectively.

   (a) The file `science2k-doc-word.npy` contains a 1373×5476 matrix, where each row is an article in Science described by 5476 word features. The articles and words are in the same order as in the vocabulary and titles files above. You can read this file using

   ```
   numpy.load("science2k-doc-word.npy")
   ```

   To obtain the features, we performed the following transformation. First, we computed per-document smoothed word frequencies. Second, we took the log of those frequencies. Finally, we centered the per-document log frequencies to have zero mean.

   Cluster the documents using k-means and various values of k (go up to at least $k = 20$).

   Select a value of k.

   For that value, report the top 10 words of each cluster in order of the largest positive distance from the average value across all data. More specifically, if $\bar{\mathbf{x}}$ is the 5476-vector of average values across documents and $\mathbf{m}_i$ is the ith mean, report the words associated with the top

components in $\mathbf{m}_i - \bar{\mathbf{x}}$. Report the top ten documents that fall closest to each cluster center. You can find the titles in the `science2k-titles.dat` file.

Comment on these results. What has the algorithm captured? How might such an algorithm be useful?

(b) The file `science2k-word-doc.txt` is similar, but capture *term-wise* rather than document-wise features. That is, for each *term*, we count the frequency as the number of *documents* that term appears in rather than the other way around. This allows us to characterize individual terms.

This matrix is 5476×1373, where each row is a term in Science described by 1373 "document" features. These are transformed document frequencies (as above). Repeat the analysis above, but cluster terms instead of documents. The terms are listed in `science2k-vocab.txt`

Comment on these results. How might such an algorithm be useful? What is different about clustering terms from clustering documents?

3. **EM algorithm and implementation**

(a) The parameters of Gaussian Mixture Model (GMM) can be estimated via the EM algorithm. Show that the alternating algorithm for k-means (in Lec. 11) is a special case of the EM algorithm and show the corresponding objective functions for E-step and M-step.

(b) Download the Old Faithful Geyser Dataset. The data file contains 272 observations of (*eruption time*, *waiting time*). Treat each entry as a 2 dimensional feature vector. Parse and plot all data points on 2-D plane.

(c) Implement a bimodal GMM model to fit all data points using EM algorithm. Explain the reasoning behind your termination criteria. For this problem, we assume the covariance matrix is spherical (i.e., it has the form of $\sigma^2 I$ for scalar σ) and you can randomly initialize Gaussian parameters. For evaluation purposes, please submit the following figures:

- Plot the trajectories of two mean vectors in 2 dimensions (i.e., coordinates vs. iteration).
- Run your program for 50 times with different initial parameter guesses. Show the distribution of the total number of iterations needed for algorithm to converge.

(d) Repeat the task in (c) but with the initial guesses of the parameters generated from the following process:

- Run a k-means algorithm over all the data points with $K = 2$ and label each point with one of the two clusters.
- Estimate the first guess of the mean and covariance matrices using maximum likelihood over the labeled data points.

Compare the algorithm performances of (c) and (d).

4. **Multidimensional scaling for genetic population differences.** In this exercise, we will look at a dataset of 42 human geographic populations collected by Cavalli-Sforza *et al.* in *The History and Geography of Human Genes*, 1994. There are many ways to measure genetic similarity between populations. This work uses the *modified Nei's distance,* which compares allele frequencies at specific locations within the human genome. Other ways to measure genetic similarity include the edit distance between DNA sequences or Euclidean distance of gene expressions.

This dataset contains comparisons between every pair of populations in the study as a distance matrix. You can use the following code to interact with it:

```
import numpy as np
data = np.load("mds-population.npz")
print data['D'] # Distance matrix
print data['population_list'] # List of populations
```

Here, $D_{i,j}$ is the dissimilarity between population i and j. Higher scores indicate more dissimilarity.

Since this distance is not based on an underlying vector representation, we cannot directly use traditional techniques like classification or clustering to analyze it. Our first step will be to *embed* this distance matrix into an m-dimensional vector space.

(a) First, use multidimensional scaling (MDS) to coerce D into a 2-dimensional vector representation. You can use the functions in `sklearn.manifold` for this.

   i. MDS will attempt to output a set of points $\mathbf{x} \in \mathscr{R}^{42,m}$ such that the Euclidean distance between every pair of points approximates the Nei's distance between these populations, or $\|x_i - x_i\|_2 \approx D_{i,j}$. What assumptions are being made? Under what circumstances could this fail? How could we measure how much information is being lost? Please explain.

   ii. One way of increasing the quality of the output is by increasing the dimensionality of the MDS result. How many dimensions are necessary to capture most of the variation in the data? There are several ways of making this judgment: for instance, you might sample some quality measure between x and D while varying m, or you might count the nonzero singular values of D, or inspect the singular values of x at some high dimension like $m = 20$. Briefly explain your method and justify why it makes sense.

   iii. Use MDS to embed the distance matrix into only two dimensions and show the resulting scatterplot. Label each point with the name of its population.

(b) *k-means on 2D embedding.* Select an appropriate k and run k-means on the scatterplot. Show the resulting clusters.

   Since we are working with only two dimensions, this clustering is likely to lose a lot of high-dimensional structure. Do you agree with the resulting clustering? What information seems to be lost?

(c) *Comparing hierarchical clustering with K-Means.* Use hierarchical clustering to cluster the original distance matrix. We suggest using the functions in `scipy.cluster.hierarchy` for this, as this library can plot the resulting graph structure. Show the resulting tree as a *dendrogram*, labeling the x axis with the categorical population names and the y axis with the Nei's distance between clusters. (It's also possible to do this with `sklearn`, but traversing the resulting structure is much harder).

   To turn the resulting tree into a flat clustering of points, cut off the dendrogram at a certain distance by merging all subclusters within this distance together. This can be done with the `scipy.cluster.hierarchy.fcluster` function. Select a distance cutoff that roughly corresponds with your chosen k earlier, balancing the number of points in each cluster with the number of resulting clusters. Visualize the resulting clustering by coloring the corresponding points on your 2D MDS embedding. How does this clustering compare with the k-means clustering you computed earlier?

(d) *Compare k-medoids with k-means.* Repeat the above experiment, but applying k-medoid clustering on the original distance matrix. Show the resulting clusters on a 2D scatterplot. Are there any significant differences between the clustering chosen by k-medoids compared to k-means?

WRITTEN EXERCISES

1. **Decision trees.** Suppose we modify the tree-growing algorithm presented in class to use the impurity function

$$I(r) = \min\{r, 1-r\}. \tag{1}$$

Let us call this the *min-error* impurity function.

As usual, for a split where p_1 positive and n_1 negative examples reach the left branch and p_2 positive and n_1 negative examples reach the right branch, the weighted impurity of the split will be

$$(p_1+n_1)\cdot I\left(\frac{p_1}{p_1+n_1}\right) \quad + \quad (p_2+n_2)\cdot I\left(\frac{p_2}{p_2+n_2}\right). \tag{2}$$

(a) Suppose that each branch of this split is replaced by a leaf labeled with the more frequent class among the examples that reach that branch. Show that the number of training mistakes made by this truncated tree is exactly equal to the weighted impurity given above. Thus, using the min-error impurity is equivalent to growing the tree greedily to minimize training error.

(b) Suppose the dataset looks like the following. There are three $\{0,1\}$-valued attributes, and one $\{-,+\}$-valued class label y.

a_1	a_2	a_3	y
0	0	0	+
1	1	0	+
0	1	0	+
1	0	1	-
0	0	1	-
0	1	0	-
1	1	0	-
1	1	1	-
1	0	0	-
1	1	0	-

Which split will be chosen at the root when the Gini index impurity function is used? Which split will be chosen at the root when min-error impurity is used? Explain your answers.

(c) Under what general conditions on p_1, n_1, p_2, and n_2 will the weighted min-error impurity of the split be strictly smaller than the min-error impurity before making the split (i.e., of all the examples taken together)?

(d) What do your answers to the last two parts suggest about the suitability of min-error impurity for growing decision trees?

2. **Bootstrap aggregation ("bagging")** Suppose we have a training set of N examples, and we use bagging to create a bootstrap replicate by drawing N samples **with replacement** to form a new

training set. Because each sample is drawn with replacement, some examples may be included in this bootstrap replicate multiple times, and some examples will be omitted from it entirely.

As a function of N, compute the expected fraction of the training set that does not appear at all in the bootstrap replicate. What is the limit of this expectation as $N \to \infty$?