Directions:

- You can work individually or with one partner. If you work in a pair, both partners will receive the same grade.
- Detailed submission instructions are on the course website: http://courses.cs.washington.edu/csep521
  If you work in pairs, only one member should submit all the relevant files.
- Concise, clear and precise answers to all questions are appreciated (and expected).
- Remember to assign your PDF pages to questions when submitting your solutions on Gradescope. Note that one can assign multiple pages to a single question (and you must assign all pages associated with a question to that question) and a single page to multiple questions (if one solves 2 questions on the same page, which we would strongly prefer that you do not do)
- Get started early!

Assignment courtesy of Greg Valiant and Tim Roughgarden.

1. Similarity Metrics (22 points)

Goal: The goal of this part is to understand better the differences between distance metrics, and to think about which metric makes the most sense for a particular application.

Description: In this part you will look at the similarity between the posts on various newsgroups. We’ll use the well-known 20 newsgroups dataset. You will use a version of the dataset where every article is represented by a bag-of-words: a vector indexed by words, with each component indicating the number of occurrences of that word. You will need 3 files: data50.csv, label.csv, and group.csv. All of these can be downloaded from the course website. In data50.csv there is a sparse representation of the bags-of-words, with each line containing 3 fields: articleId, wordId, and count. To find out which group an article belongs to, use the file label.csv, where for articleId i, line i in label.csv contains the groupId. Finally the group name is in group.csv, with line i containing the name of group i.

We’ll use the following similarity metrics, where \( x = (x_1, \ldots, x_k) \) and \( y = (y_1, \ldots, y_k) \) are two bags of words (so \( x_i \) is the number of occurrences of word \( i \) in the document represented by \( x \)):

- Jaccard Similarity \( J(x, y) \):
  \[
  J(x, y) := \frac{\sum_{i=1}^{k} \min(x_i, y_i)}{\sum_{i=1}^{k} \max(x_i, y_i)}.
  \]

- \( L_2 \) Similarity:\(^2\)
  \[
  L_2(x, y) = -||x - y||_2 = -\sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}.
  \]

\(^1\)http://qwone.com/~jason/20Newsgroups/

\(^2\) While we typically talk about \( L_2 \) or Euclidean distance, to make sure that a high number means a higher similarity, we negate the distances.
Cosine Similarity:

\[ S_C(x, y) = \sum_{i=1}^{k} x_i \cdot y_i \left/ \left( ||x||_2 \cdot ||y||_2 \right) \right. \]

Note that Jaccard and Cosine similarity are numbers between 0 and 1, while \( L_2 \) similarity is between \(-\infty \) and 0 (with higher numbers indicating more similarity).

(a) (3 points) Make sure you can import the given datasets into whatever language you’re using. For example, if you’re using python, read the data50.csv file and store the information in an appropriate way. Remember that the total number of words in the corpus is huge, so you probably want to work with a sparse representation of your data (e.g., you don’t want to waste space on words that don’t occur in a document). If you’re using MATLAB, you can simply import the data using the GUI.

(b) (12 points) Implement the three similarity metrics described above. For each metric, prepare the following plot. The plot will look like a \( 20 \times 20 \) matrix. Rows and columns are index by newsgroups (in the same order). For each entry \((A, B)\) of the matrix (including the diagonal), compute the average similarity over all ways of pairing up one article from A with one article from B. After you’ve computed these 400 numbers, plot your results in a heatmap. Make sure that you label your axes with the group names and pick an appropriate colormap to represent the data: the rainbow colormap may look fancy, but a simple color map from white to blue may be a lot more insightful. Make sure to include a legend.

(c) (7 points) Based on your three heatmaps, which of the similarity metrics seems the most reasonable, and why would you expect these metrics to be better suited to this data? Are there any pairs of newsgroups that are very similar? Would you have expected these to be similar?

Deliverables: All of your code. Three heat maps for (b), your discussion/explanations for (c).

2. Nearest-neighbor classification using dimension reduction (24 points)

A "nearest-neighbor" classification system is conceptually extremely simple, and often is very effective. Given a large dataset of labeled examples, a nearest-neighbor classification system will predict a label for a new example, \( x \), as follows: it will find the element of the labeled dataset that is closest to \( x \) in whatever metric makes the most sense for that dataset and then output the label of this closest point. [As you can imagine, there are many natural extensions of this system; for example considering the labels of the \( r > 1 \) closest neighbors.] From a computational standpoint, naively, finding the closest point to \( x \) might be time consuming if the labeled dataset is large, or the points are very high dimensional. In the next two parts, you will explore two ways of speeding up this computation: using dimension reduction, and, in the next problem, using locality sensitive hashing.

Goal: The goal of this part is to get a feel for the trade-off in dimensionality reduction between the quality of approximation and the number of dimensions used.

Description: You may have noticed that it takes some time to compute all the distances in the previous part, though it should not take more than a minute or two. In this part we will implement a dimension reduction technique to reduce the running time, which can be used to also speed up classification. In the following, \( k \) will refer to the original dimension of your data (which for this dataset is 19,575, the number of words that occur in these articles), and \( d \) will refer to the target dimension.

- Random Projection: Given a set of \( k \)-dimensional vectors \( \{v^{(1)}, v^{(2)}, \ldots\} \) (each \( v^{(i)} \) is itself a \( k \)-dimensional vector) define a \( d \times k \) matrix \( M \) by drawing each entry randomly (and independently) from a normal distribution of mean 0 and variance 1. The \( d \)-dimensional reduced vector corresponding to \( v^{(i)} \)

If you feel that the computations are taking too long, then you should look into optimizing your code and in particular consider caching repeated computations.
is given by the matrix-vector product $Mv^{(i)}$. We can think of the matrix $M$ as a set of $d$ random $k$-dimensional vectors $\{w^{(1)}, w^{(2)}, \ldots, w^{(d)}\}$ (the rows of $M$), and then the $j$-th coordinate of the reduced vector $Mv^{(i)}$ is the inner product between the vectors $v^{(i)}$ and $w^{(j)}$. If you need to review the basics of matrix-vector multiplication, see the links on the course webpage.

(a) (5 points) Baseline Classification: Implement the baseline cosine-similarity nearest-neighbor classification system that, for any given document, finds the document with largest cosine similarity, and returns that newsgroup/label. (These computations should be exact, i.e., you need to exactly compute the cosine similarity between each pair of documents in the 20 newsgroups dataset.)

Compute the $20 \times 20$ matrix whose entry $(A, B)$ is defined by the fraction of articles in group $A$ that have their nearest neighbor in group $B$.

Plot these results in a heatmap.

What is the average accuracy (i.e., what percentage of the 1000 articles have their nearest neighbor in the same news-group?)

(b) (3 points) Your plots for Part 1(b) were symmetric – why is the matrix in (a) not symmetric?

(c) (10 points) Implement the random projection dimension reduction function and plot the nearest-neighbor visualization as in part (a) for cosine similarity and $d = 10; 25; 50; 100$.

What is the average accuracy for each of these settings?

For which values of the target dimension are the results comparable to the original dataset?

(d) (6 points) What is the time it takes to reduce the dimensionality of the data? Suppose that you are trying to build a very fast article classification system, and have an enormous dataset of $n$ labeled tweets/articles. What is the overall Big-Oh runtime of classifying a new article, as a function of $n$ (the number of labeled datapoints), $k$ (the original dimension of each datapoint), and $d$ (the reduced dimension)? (Classifying a new article means returning the label of the nearest datapoint in the dataset.)

Now suppose you are instead trying to classify tweets; the bag-of-words representation is still a $k$ dimensional vector, but now each tweet has, say, only 50 words (which is much much less than $k$). Explain how you could exploit the sparsity of the data to improve the runtime of the naive cosine-similarity nearest-neighbor classification system (from part (a)). How does this runtime compare to that of a dimension-reduction nearest-neighbor system (as in the first step of this part) that reduces the dimension to $d = 50$? [For this part, we expect a theoretical analysis – you do not need to implement these algorithms and measure their runtimes empirically.]

Deliverables: Code, figures, classification performance for part (a), brief explanation for part (b), code, classification performance for part (c), discussion and analysis for part (d).

3. Locality Sensitive Hashing (38 points)

Goal: The goal of this part is to implement a basic Locality-Sensitive-Hashing\(^5\) nearest-neighbor classification system, and experiment with the tradeoff between bucket size and the number of hash tables. This part is largely an illustration that such techniques can be applied for fast classification. A larger dataset would have illustrated this much better (though Parts 1 and 2 would have taken much longer).

\(^4\) Again, this means that each tweet is represented by a length $k$ vector, say $v$, where $v_i = c$ means that the $i$-th word in the corpus appears $c$ times in the tweet. ($c$ can be 0, and in fact will almost always be 0)

\(^5\) Locality Sensitive Hashing will be discussed in class on February 7.
Description: You will implement the Random Hyperplane Hashing LSH scheme, which has the property that vectors with larger cosine similarity will have a higher probability of colliding (i.e. hashing to the same value). [You will be able to reuse much of the code from Part 2.] The hashing scheme, and associated nearest-neighbor classification system, is defined as follows:

- **Hyperplane Hashing**: Construct \( \ell \) hash tables in the following manner: for the \( i \)'th hashtable, define a \( d \times k \) matrix \( M_i \) by drawing each entry randomly (and independently) from a normal distribution of mean 0 and variance 1. The \( i \)'th hashvalue of the \( k \)-dimensional vector \( v \) is defined as the binary vector \( \text{sgn}(M_i v) \in \{0, 1\}^d \) where each positive coordinate of \( M_i v \) is replaced by a “1” and each nonpositive coordinate by a “0”. Note that each hashtable has \( 2^d \) buckets, and each data point is placed in exactly one bucket in each of the \( \ell \) hashtables.

- **Classification**: Suppose each original datapoint \( v \) has already been hashed (to bucket \( \text{sgn}(M_i v) \) of the \( i \)'th hashtable, for each \( 1 \leq i \leq \ell \)). Then, to predict the label of a (new) query vector \( q \) do the following:
  (i) compute its \( \ell \) hashvalues (bucket \( \text{sgn}(M_i q) \) of the \( i \)'th hashtable, for each \( 1 \leq i \leq \ell \));
  (ii) consider the set \( S_q \) of the original datapoints that were placed in at least one of these \( \ell \) buckets;
  (iii) among all points in \( S_q \), find the data point \( x \) that is most similar to the query \( q \) (using brute-force search over \( S_q \)); and
  (iv) label \( q \) with \( x \)'s label.

(a) (10 points) Consider the \( i \)'th hash tables in the above scheme, corresponding to matrix \( M_i \). For two vectors \( q, x \in R^k \) that form an angle of \( \angle(q, x) = \theta < \pi/2 \) (i.e. \( q \) and \( x \) form an acute angle of \( \theta \)), what is the probability (over the randomness in the construction of the matrix \( M_i \)) that they hash to the same bucket in this \( i \)'th hash table? [Hint: for each of the \( d \) coordinates that define the hash of \( q \) and \( x \), what is the probability that they are equal, as a function of \( \theta \)?] Prove your claim in a few sentences.

(b) (2 points) If \( \angle(q, y) \geq 20 \), what can you say about the probability that \( q \) and \( y \) hash to the same bucket in the \( i \)'th hash table?

(c) (10 points) Suppose you have a dataset \( X = \{x^{(1)}, \ldots, x^{(n)}\} \) with \( n = 1,000,000 \) points, and consider the scheme described in this section ("Hyperplane Hashing" and "Classification" protocols), where you have \( \ell \) different hash tables, and, for a query point \( q \in R^k \), you check every point in your dataset \( X = \{x^{(1)}, \ldots, x^{(n)}\} \) that collides with \( q \) in at least one of the \( \ell \) different hash tables. Suppose you know that there is some point \( x^{(i)} \in X \) with angle at most 0.1 radians from \( q \) and that there are not too many points \( x^{(j)} \) with \( \angle(q, x^{(j)}) \in (0.1, 0.2] \). How should you pick \( d \) and \( \ell \) such that:
  (i) With probability at least 0.9, the point \( x^{(i)} \) with \( \angle(q, x^{(i)}) \leq 0.1 \) ends up hashing to the same bucket as \( q \) in at least one of the hash tables.
  (ii) The expected number of points \( x^{(\ell)} \) that have angle greater than 0.2 from \( q \) that you end up needing to consider (i.e. that hash into the same bucket in at least one of the \( \ell \) hash functions) is small.
  (iii) For a given datapoint, the total time to compute all \( \ell \) hashes is small.

Compute actual numeric values for \( d \) and \( \ell \) in the case that \( n = 1,000,000 \).

Note: There is no single right answer here (for example, I have not defined "small" in (ii) and (iii)). Your analysis here should be similar to the one done in class on February 7 and you’ll need to reference your answers to parts (a) and (b) of this question. Imagine that you are actually trying to design such a nearest-neighbor search algorithm for a company, and are asked to explain your choice of the parameters to your supervisor.

(d) (10 points) Implement the Locality Sensitive Hashing scheme and corresponding "Classification" protocol described at the beginning of this problem, for the newsgroups dataset, with \( \ell = 128 \) hash functions, and \( d = 5, 6, \ldots, 20 \). For each value of \( d \), compute the average classification error of the corresponding
scheme, and compute the average size of the set $S_q$ (averaged over the 1000 articles). Plot the average classification error versus the average number of articles inspected. Is there a "sweet spot" in terms of the tradeoff between classification error and the average size of $S_q$ (which in turn governs the running time of classification)?

(e) (6 points) Compare and contrast the performance properties the LSH-based nearest neighbor classification system in part (d) with those of the dimension-reduction-based one in Problem 2. What properties of an application would suggest that the former would be a better choice than the latter, and vice versa? Describe how you might combine the two approaches. For example, could dimensionality reduction help speed up the brute-force computation in the LSH classification system? When, if ever, might such a combination outperform both of the single-approach systems? Justify your answer.

Deliverables: Parts (a) and (b) require a short rigorous analysis. For part (c), a compelling/rigorous argument and short discussion. For part (d): code, average classification accuracy and average numbers of inspected datapoints, plot, and brief discussion. Part (e): discussion.