

## Mini-Project #2

Due by 11:59 PM on Tuesday, April 13.

### Instructions

- You can work in groups of up to four students. If you work in a group, please submit one assignment via Gradescope (with all group members' names).
- Detailed submission instruction can be found on the course website (<https://web.stanford.edu/class/cs168/>) under "Coursework - Assignments" section. If you work in pairs, **only one member** should submit.
- Use 12pt or higher font for your writeup.
- Make sure the plots you submit are easy to read at a normal zoom level.
- If you've written code to solve a certain part of a problem, or if the part explicitly asks you to implement an algorithm, you must also include the code in your pdf submission.
- Code marked as able should be pasted into the relevant section. Keep variable names consistent with those used in the problem statement, and with general conventions. No need to include import statements and other scaffolding, if it is clear from context. Use the `verbatim` (or "minted") environment to paste code in LaTeX.

```
def example():  
    print "Your code should be formatted like this."
```

- **Reminder:** No late assignments will be accepted, but we will drop your lowest assignment grade.

### Part 1: Similarity Metrics

**Goal:** The goal of this part of the assignment 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.<sup>1</sup> 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`, v, 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  $\mathbf{x}$  and  $\mathbf{y}$  are two bags of words:

- Jaccard Similarity:  $J(\mathbf{x}, \mathbf{y}) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)}$ .
- $L_2$  Similarity<sup>2</sup>:  $L_2(\mathbf{x}, \mathbf{y}) = -\|\mathbf{x} - \mathbf{y}\|_2 = -\sqrt{\sum_i (x_i - y_i)^2}$ .

---

<sup>1</sup><http://qwone.com/~jason/20Newsgroups/>

<sup>2</sup>While we typically talk about  $L_2$  distance, to make sure that a higher number means a higher similarity we negate the distances.

- Cosine Similarity:  $S_C(\mathbf{x}, \mathbf{y}) = \frac{\sum_i x_i \cdot y_i}{\|\mathbf{x}\|_2 \cdot \|\mathbf{y}\|_2}$ .

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).

- (2 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 might 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.
- (8 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. (Note that the computation might take five or ten minutes, but shouldn't take much more.)
- (4 points) Based on your three heatmaps, which of the similarity metrics seems the most reasonable, and why would you expect that/those 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).

## Parts 2 and 3: A nearest-neighbor classification system

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  $x$ —closest 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: dimension reduction, and via *locality sensitive hashing*.

## Part 2: Dimension Reduction

**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, and  $d$  will refer to the target dimension.

- Random Projection: Given a set of  $k$ -dimensional vectors  $\{v_1, v_2, \dots\}$ , 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  $v_i$  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  $\{w_1, \dots, w_d\}$  (the rows of  $M$ ), and then the  $j$ th coordinate of the reduced vector  $Mv_i$  is the inner product between that  $v_i$

and  $w_j$ . If you need to review the basics of matrix-vector multiplication, see the primer on the course webpage.

- (a) (3 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. (Do each computation using brute-force search.)

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

Plot these results in a heatmap.

What is the average classification accuracy (i.e., what fraction of the 1000 articles have the same newsgroup/label as their closest neighbor)?

- (b) (2 points) Your plots for Part 1(b) were symmetric—why is the matrix in (a) not symmetric?
- (c) (7 points) Implement the random projection dimension reduction function and plot the nearest-neighbor visualization as in part (a) for cosine similarity and  $d = 10, 25, 50, 100, 200, 500, 1000$ .

What is the average classification error for each of these settings?

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

- (d) (5 points) Suppose you are trying to build a very fast article classification system, and have an enormous dataset of  $n$  labeled tweets/articles. What is the time it takes to reduce the dimensionality of the data? Give the Big-Oh runtime as a function of  $n$  (the number of labeled datapoints),  $k$  (the original dimension of each datapoint), and  $d$  (the reduced dimension). What is the overall Big-Oh runtime of classifying a new article? [Feel free to assume a naive matrix multiplication algorithm, as opposed to “fast matrix multiplication” algorithms, such as Strassen’s algorithm.]

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  $50 \ll k$  words. 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, plots, and classification performance for part (c), discussion and analysis for part (d).

## Part 3: Locality Sensitive Hashing

**Goal:** The goal of this part is to implement a basic Locality-Sensitive-Hashing nearest-neighbor classification system, and experiment with the tradeoff between bucket size and 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 better (though Parts 1 and 2 would have taken much longer :).

**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  $\ell$  hashtables 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 of 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  $i$ ). Then, to predict the label of a (new) query vector  $q$ , do the following: (i) compute its  $\ell$  hashvalues (bucket  $\text{sgn}(M_i u)$  of the  $i$ th hashtable); (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 of  $S_q$ , compute 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) (3 points) Consider the  $i$ th hash tables in the above scheme, corresponding to matrix  $M_i$ . For two vectors,  $x, y \in \mathcal{R}^k$  that form an angle of  $\text{angle}(x, y) = \theta < \pi/2$  radians (i.e.  $x$  and  $y$  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 function? [Hint: for each of the  $d$  coordinates that define the hash of  $x$  and  $y$ , what is the probability that they are equal, as a function of  $\theta$ ? To figure this out, it might be helpful to consider, geometrically, what it means for  $x$  and  $y$  to have the same sign when multiplied by a random vector. What does this look like in two dimensions? What random vectors will cause the inner products to have opposite signs?] Prove your claim in at most two sentences.
- (b) (6 points) Suppose you have a dataset  $X = \{x_1, x_2, \dots, x_n\}$  with  $n = 1,000,000$  points, and consider the scheme described in the “Hyperplane Hashing” and “Classification” protocols, where you have  $\ell$  different hash tables, and, for a query point  $y \in \mathcal{R}^k$ , you check every point in your dataset  $X = \{x_1, x_2, \dots, x_n\}$  that collides with  $y$  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  $y$ , and there are no points in the dataset that form an angle with  $y$  in the range  $(0.1, 0.2)$ . How should you pick  $d$  and  $\ell$  such that 1) With probability at least 0.9, the point  $x_i$  with  $\text{angle}(x_i, y) \leq 0.1$  ends up hashing to same bucket as  $y$  in at least one of the  $\ell$  hash tables, and 2) You *minimize* the total expected time it takes to find  $x_i$ , assuming that it takes time  $d \cdot k$  to compute each of the  $\ell$  hashes of query point  $y$ , and it takes time  $k$  to evaluate the distance between two  $k$ -dimensional vectors. Feel free to assume (pessimistically) that all  $n - 1$  points, other than  $x_i$ , have an angle of exactly 0.2 with  $y$ .

Compute actual numeric values for  $d$  and  $\ell$  in the case that  $n = 1,000,000$ ; if you like, in *addition* you may also give an answer as a function of  $n$ . Discuss how you arrived at your optimization, and briefly discuss any practical considerations that might come into play and cause the above theoretical analysis to be suboptimal in practice.

- (c) (6 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  $d = 5, 6, \dots, 20$ . 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)?
- (d) (4 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 better choice than the latter, or 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:** Part (a) a short rigorous analysis. Part (b), a compelling/rigorous argument and short discussion. Part (c): code, average classification accuracy and average numbers of inspected datapoints, plot, and brief discussion. Part (d): discussion.