CS168, Spring 2024

Mini-Project #2

Due by 11am on Thursday, April 18.

Instructions

- You can work in groups of up to four students. If you work in a group, please submit one assignment via Gradescope (please have all group members link all group members to your submission).
- 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.[1] 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 $\mathbf{x}$ 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[2]: $L_2(\mathbf{x}, \mathbf{y}) = -||\mathbf{x} - \mathbf{y}||_2 = -\sqrt{\sum_i (x_i - y_i)^2}$.

[1] `http://qwone.com/~jason/20Newsgroups/`

[2] 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).

(a) (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.

(b) (10 points) Implement the three similarity metrics described above. For each metric, prepare the following plot. The plot will look like a 20×20 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 more than that.)

(c) (5 points) Based on your three heatmaps, which of the similarity metrics seems the most reasonable, and why would you expect 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 couple of minutes). 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 $\{v_1, v_2, \ldots\}$, 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 $\{w_1, \ldots, w_d\}$ (the rows of M), and then the jth 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) (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. (Do each computation using brute-force search.)

Compute the 20×20 matrix whose entry (A, B) is defined by the number of articles in group A that have their nearest neighbor in group B. (When computing an article's nearest neighbor, don't compute the similarity with itself, otherwise all the articles will be their own nearest neighbors, and this part would be meaningless.) Does it make sense why this should correspond to the accuracy of a nearest-neighbor classification system based on this dataset?

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) (4 points) Suppose each document were much, much longer. Would you need larger target dimensions in the random dimension reduction to accurately capture the similarity between this larger articles? Explain your answer in at most 3 or 4 sentences.

(e) (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), yes/no and brief explanation for (d), discussion and analysis for part (e).

Part 3: Locality Sensitive Hashing

Goal: The goal of this part is to think about a basic Locality-Sensitive-Hashing nearest-neighbor classification system which could be used to speed up the computations performed in Part 2. This part is purely theoretical analysis, and we split this up into a number of very small pieces—(a),(b),(c) and (d) all have correct one-sentence "proofs".

Description: Below is a description of a *Random Hyperplane Hashing* LSH scheme, which has the property that vectors with larger cosine similarity will have a higher probability of colliding. The hashing scheme, and associated nearest-neighbor classification system, is defined as follows:

- Hyperplane Hashing: Construct ℓ 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 ith 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 ℓ hashtables.

- Classification: Given a dataset X, suppose each original datapoint $v \in X$ has already been hashed (to bucket $\text{sgn}(M_i v)$ of the ith hashtable, for each $i = 1, 2, \ldots, \ell$). Then, to predict the label of a (new) query vector q, do the following: (i) compute its ℓ hashvalues (bucket $\text{sgn}(M_i u)$ of the ith hashtable); (ii) consider the set S_q of the original datapoints that were placed in at least one of these ℓ buckets; (iii) If $|S_q| \leq 10\sqrt{n}$ then go through the elements of S_q one by one computing the angle that each one forms with q and return the label of the closest one to q; if $|S_q| > 10\sqrt{n}$ do the above search but only look at the first $10\sqrt{n}$ elements of S_q.

(a) (3 points) Consider the ith hash table in the above scheme, corresponding to matrix M_i. For two vectors, $x, y \in \mathcal{R}^k$ that form an angle of $angle(x, y) = \theta < \pi$ radians, what is the probability (over the randomness in the construction of the matrix M_i) that they hash to the same bucket in this ith 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 θ? 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. If you set $\theta = 0.1$, numerically, what probability do you get?

(b) (2 points) In the next few parts, we'll let n denote the number of datapoints in our dataset X, and argue that we can pick ℓ and d in such a way that with high probability, 1) S_q will contain almost all "close" points—specifically all points whose angle with q is at most 0.1, and 2) S_q won't contain too many "far" points—specifically, S_q will contain at most $O(\sqrt{n})$ points with angle more than 0.2 with q, (and probably won't contain *any* points with angle more than 0.3, though we won't bother showing that).

Suppose there is a point $x \in X$ such that $angle(x, q) \leq 0.1$. Prove that

$$\Pr[x \in S_q] \geq 1 - \left(1 - 0.968^d\right)^\ell \geq 1 - e^{-\ell \cdot 0.968^d}.$$

[Hint: to get the final inequality, use the fact that $(1 - \delta) < e^{-\delta}$.]

(c) (2 points) Prove that the expected number of elements of S_q that have an angle with q of more than 0.2 radians is bounded as follows:

$$\mathbf{E}[|\{x \in S_q \text{ with } angle(x, q) > 0.2\}|] \leq n\left(1 - \left(1 - 0.937^d\right)^\ell\right) \leq n\ell \cdot 0.937^d.$$

[Hint: to get the final inequality, use the fact that for $\delta \in (0, 1)$ and $m \geq 1$ the following is true: $(1 - \delta)^m \geq 1 - m\delta$.]

(d) (1 points) Consider setting $d \approx 15 \log n$ so that $0.937^d \approx 1/n$, and hence $0.968^d \approx 1/\sqrt{n}$, and set $\ell = 5\sqrt{n}$. Using the previous two parts, show that if $angle(x, q) < 0.1$ then $\Pr[x \in S_q] > 0.99$, and the expected number of points in S_q with angle more than 0.2 from q is at most $5\sqrt{n}$.

(e) (3 points) If your dataset X consists of n point in $\mathcal{R}^k$, and the nearest neighbor of your query q does have angle at most 0.1 with the query point, then what should we expect the runtime of the Classification protocol to be, as a function of n and k, where we plugged in the values of d and ℓ from the previous part? When do we get a "win" over the naive brute-force nearest-neighbor search? Is there a regime in which it might make sense to use dimension reduction together with this scheme?

(f) (1 point) The above hashing based nearest neighbor search implicitly assumed that if there is a point with angle at most 0.1 from our query point, then we are satisfied if we find a point with angle at most 0.2 from the query point. Is this a reasonable goal? Discuss in two or three sentences. (Note: A slight extension of the scheme we described has similar runtime, but satisfies the stronger guarantee that if the nearest neighbor has angle α, then we expect to find a point with angle at most 2α, and we don't need to know α in advance...)

(g) (1 points) [Challenge] Assuming that there is some point with angle at most 0.1 from our query point, if our goal is to find a point whose angle is at most $0.1c$ from the query for some constant $c \geq 1$, what is the right variant of the approach, how should we set d and ℓ to minimize the runtime, and what is the runtime as a function of c and n? [Feel free to assume the original dimensionality, $k = O(\log n)$, and feel free to give a runtime of the form $O(n^{f(c)})$ where you ignore any constant terms and $log(n)$ terms.]

Deliverables: Part (a)-(d) short rigorous analyses. Part (e) and (f) short discussions. Part (g) algorithm sketch and sketch of analysis.