CS 189 Introduction to Machine Learning
Fall 2018 HW9

This homework is due **Wednesday, December 5 at 10pm.**

2 Running Time of k-Nearest neighbor Search Methods

The method of k-nearest neighbors is a very simple idea that forms a fundamental conceptual building block of machine learning, and plays a role in many ML algorithms. A classic example is the k-nearest neighbor classifier, which is a non-parametric classifier that finds the k closest examples in the training set to the test example, and then outputs the most common label among them as its prediction. Generating predictions using this classifier requires an algorithm to find the k closest examples in a possibly large and high-dimensional dataset, which is known as the k-nearest neighbor search problem. More precisely, given a set of n points, $\mathcal{D} = \{\mathbf{x}_1 \ldots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$ and a query point $\mathbf{z} \in \mathbb{R}^d$, the problem requires finding the k points in $\mathcal{D}$ that are the closest to $\mathbf{z}$ in Euclidean distance.

This problem explores the computational complexity of nearest-neighbor methods to show how naive implementations perform very poorly as the dimensionality of the problem grows, but more sophisticated use of randomized techniques can do better.

Overall Hint: In this problem, reading later parts will help you know what you need to do in earlier parts in case you can't figure it out. So, read ahead before asking a question.

(a) First, we will try out a k-nearest neighbor classifier on the accessibility dataset collected earlier in the semester. Like in the previous homework, you should load, clean, and split the dataset into a test and train set, with 3000 datapoints in the test set. We will try two different classifiers: one which computes Euclidean distance in image feature space, and one which computes Euclidean distance on latitude and longitude. **Plot test set accuracy curves of the two k-nearest neighbor classifiers for $k \in \{1, 5, 10, 15, 20, 25, 30\}$.** You may use scikit-learn methods.

Solution:

```
from sklearn.neighbors import KNeighborsClassifier

ks = [1, 5, 10, 15, 20, 25, 30]
accs = []
for k in ks:
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(X_train, y_train)

    y_pred = neigh.predict(X_test)
    acc = sum(y_pred == y_test)/len(y_pred)
    print(acc)
    accs.append(acc)

accs_latlong = []
for k in ks:
```

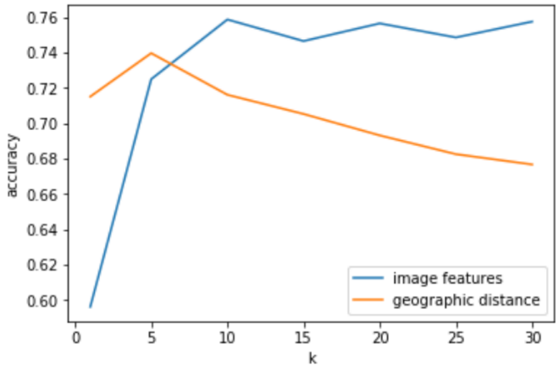


Figure 1: Accuracy vs. k for distance in feature space and distance in latitude/longitude.

```
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(latlons_train, y_train)

    y_pred = neigh.predict(latlons_test)
    acc = sum(y_pred == y_test)/len(y_pred)
    print(acc)
    accs_latlong.append(acc)
```

(b) Now, let's consider the computational complexity of this algorithm. First, we consider the naïve exhaustive search algorithm, which computes the distance between $\mathbf{z}$ and all points in $\mathcal{D}$ and then returns the k points with the shortest distance. This algorithm first computes distances between the query and all points, then finds the k shortest distances using quickselect[1]. **What is the (average case) time complexity of running the overall algorithm for a single query?**

Solution: $O(d)$: time spent on computing the distance between the query and each point
$O(dn)$: time spent on computing distances between the query and all points
$O(n)$: time spent on finding the k shortest distances using quickselect on average.

The overall time complexity is $O(dn+n)$, which can be simplified to $O(dn)$, since $dn \geq n$.

(c) Decades of research have focused on devising a way of preprocessing the data so that the k-nearest neighbors for each query can be found efficiently. "Efficient" means the time complexity of finding the k-nearest neighbors is lower than that of the naïve exhaustive search algorithm – meaning that the complexity must be *sublinear* in n.

Many efficient algorithms for k-nearest neighbor search rely on a divide-and-conquer strategy known as space partitioning. The idea is to divide the feature space into cells and maintain a data structure that keeps track of the points that lie in each. Then, to find the k-nearest neighbors of a query, these algorithms look up the cell that contains the query and obtain the subset of points in $\mathcal{D}$ that lie in the cell and adjacent cells. Adjacent cells must be included in case the query point is in the corner of its cell. Then, exhaustive search is performed on this subset to find the k points that are the closest to the query.

[1] Quickselect is a counterpart of quicksort that just picks the top k in an unordered list. Instead of taking $O(n \log n)$ like quicksort on average, it takes $O(n)$. Look-up quickselect if you want, but in principle, you should be able to derive it if you understand the principle behind quicksort. Just realize that there is no point in recursively sorting things that for sure aren't going to be in the top k.

For simplicity, we'll consider the special case of $k = 1$ in the following questions, but note that the various algorithms we'll consider can be easily extended to the setting with arbitrary k. We first consider a simple partitioning scheme, where we place a Cartesian grid (a rectangular grid consisting of hypercubes) over the feature space.

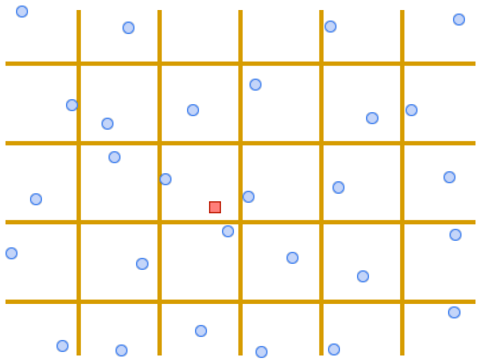


Figure 2: Illustration of the space partitioning scheme we consider. The data points are shown as blue circles and the query is shown as the red square. The cell boundaries are shown as gold lines.

How many cells need to be searched in total if the data points are one-dimensional? Two-dimensional? d-dimensional? If each cell contains one data point, what is the time complexity for finding the 1-nearest neighbor in terms of d, assuming accessing any cell takes constant time?

Solution: In 1D, there are two adjacent cells, so three cells need to be searched. In 2D, there are eight adjacent cells, so nine cells need to be searched. By induction, it is easy to see that in d dimensions, there are 3^d cells that need to be searched in $\mathbb{R}^d$ since you need to search the neighbors in each new dimension, both above and below you. This multiplies the number of cells by three with each additional dimension. The reason we have to look at all the neighbors is that our query point might be in a corner, and the points in the adjacent cells might also be in corners. So, the exact configuration matters if we want to find the true nearest neighbor.

Since each cell contains a data point, there are 3^d data points that need to be searched exhaustively. Because computing the distance for each data point takes $\Theta(d)$ time and finding the minimum of 3^d distances takes $O(3^d)$ time, the time complexity is $O(d3^d + 3^d) = O(d3^d)$.

(d) In low dimensions, the divide-and-conquer method provides a significant speedup over naïve exhaustive search. However, in moderately high dimensions, its time complexity can grow large quickly. In the high dimensional case, we modify our divide-and-conquer algorithm to use the naïve exhaustive search instead. This behavior arises in many settings, and is known as *the curse of dimensionality*. How do we overcome the curse of dimensionality? Since it arises from the need to search adjacent cells, what if we don't have cells at all?

Consider a new approach that simply projects all data points along a uniformly randomly chosen direction and keeps all projections of data points in a sorted list. To find the 1-nearest neighbor, the algorithm projects the query along the same direction used to project the data points and uses binary search to find the data point whose projection is closest to that of the query. Then it marches along the list to obtain $\tilde{k}$ points whose projections are the closest to the projection of the query. Finally, it performs exhaustive search over these points and returns the point that is the closest to the query. This is a simplified version of an algorithm known as Dynamic Continuous Indexing (DCI).

Because this algorithm is randomized (since it uses a randomly chosen direction), there is a non-zero probability that it returns the incorrect results. We are therefore interested in how many points we need to exhaustively search over to ensure the algorithm succeeds with high probability.

We first consider the probability that a data point that is originally far away appears closer to the query under projection than a data point that is originally close. Without loss of generality, we assume that the query is at the origin. Let $\mathbf{v}^l \in \mathbb{R}^d$ and $\mathbf{v}^s \in \mathbb{R}^d$ denote the far (long) and close (short) vectors respectively, and $\mathbf{u} \in S^{d-1} \subset \mathbb{R}^d$ is a vector drawn uniformly randomly on the unit sphere which serves as the random direction. Then the event of interest is when $\{|\langle \mathbf{v}^l, \mathbf{u}\rangle| \leq |\langle \mathbf{v}^s, \mathbf{u}\rangle|\}$.

Assuming that $\mathbf{0}$, $\mathbf{v}^l$ and $\mathbf{v}^s$ are not collinear,[2] consider the plane spanned by $\mathbf{v}^l$ and $\mathbf{v}^s$, which we will denote as P. For any vector $\mathbf{w}$, we use $\mathbf{w}^{\parallel}$ and $\mathbf{w}^{\perp}$ to denote the components of $\mathbf{w}$ in P and $P^{\perp}$ such that $\mathbf{w} = \mathbf{w}^{\parallel} + \mathbf{w}^{\perp}$.

If we use θ denote the angle of $\mathbf{u}^{\parallel}$ relative to $\mathbf{v}^l$, show that $\Pr\left(\left|\langle \mathbf{v}^l, \mathbf{u}\rangle\right| \leq \left|\langle \mathbf{v}^s, \mathbf{u}\rangle\right|\right) \leq \Pr\left(|\cos\theta| \leq \left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right)$.

Hint: For $\mathbf{w} \in \{\mathbf{v}^s, \mathbf{v}^l\}$, because $\mathbf{w}^{\perp} = 0$, $\langle \mathbf{w}, \mathbf{u}\rangle = \langle \mathbf{w}, \mathbf{u}^{\parallel}\rangle$.

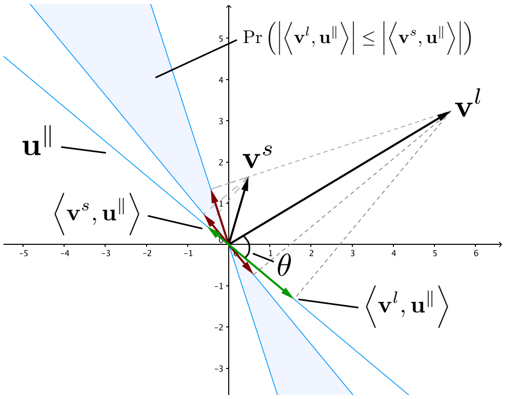

$\Pr\left(\left|\langle \mathbf{v}^l, \mathbf{u}^{\parallel}\rangle\right| \leq \left|\langle \mathbf{v}^s, \mathbf{u}^{\parallel}\rangle\right|\right)$
$\mathbf{v}^l$
$\mathbf{v}^s$
$\mathbf{u}^{\parallel}$
$\langle \mathbf{v}^s, \mathbf{u}^{\parallel}\rangle$
θ
$\langle \mathbf{v}^l, \mathbf{u}^{\parallel}\rangle$
-5
-4
-3
-2
-1
1
2
3
4
5
6
5
4
3
2
1
-1
-2
-3

Figure 3: Examples of "good" and "bad" projection directions. The blue lines denote possible projection directions $\mathbf{u}^{\parallel}$. The isolated blue line represents a "good" projection direction, since the projection of $\mathbf{v}^l$ is longer than the projection of $\mathbf{v}^s$ (both shown in green), thereby preserving the relative order between $\mathbf{v}^l$ and $\mathbf{v}^s$ in terms of their lengths after projection. Any projection direction within the shaded region is a "bad" projection direction, since the projection of $\mathbf{v}^l$ would not be longer than the projection of $\mathbf{v}^s$, thereby inverting the relative order between $\mathbf{v}^l$ and $\mathbf{v}^s$ after projection (shown in red).

Solution: Let ψ denote the angle of $\mathbf{u}^{\parallel}$ relative to $\mathbf{v}^s$.

[2] If $\mathbf{v}^l$ and $\mathbf{v}^s$ are collinear, random projection will essentially always be able to tell which is which so we don't bother to analyze that case. Understanding why will help you do this problem.

$$
\begin{aligned}
\Pr\left(\left|\langle \mathbf{v}^l, \mathbf{u}\rangle\right| \leq \left|\langle \mathbf{v}^s, \mathbf{u}\rangle\right|\right) &= \Pr\left(\left|\langle \mathbf{v}^l, \mathbf{u}^{\|}\rangle\right| \leq \left|\langle \mathbf{v}^s, \mathbf{u}^{\|}\rangle\right|\right) \\
&= \Pr\left(\left\|\mathbf{v}^l\right\|_2 \left\|\mathbf{u}^{\|}\right\|_2 |\cos\theta| \leq \left\|\mathbf{v}^s\right\|_2 \left\|\mathbf{u}^{\|}\right\|_2 |\cos\psi|\right) \\
&= \Pr\left(\left\|\mathbf{v}^l\right\|_2 |\cos\theta| \leq \left\|\mathbf{v}^s\right\|_2 |\cos\psi|\right) \\
&\leq \Pr\left(\left\|\mathbf{v}^l\right\|_2 |\cos\theta| \leq \left\|\mathbf{v}^s\right\|_2\right) \\
&= \Pr\left(|\cos\theta| \leq \frac{\left\|\mathbf{v}^s\right\|_2}{\left\|\mathbf{v}^l\right\|_2}\right)
\end{aligned}
$$

(e) The algorithm would fail to return the correct 1-nearest neighbor if more than $\tilde{k} - 1$ points appear closer to the query than the 1-nearest neighbor under projection.

The following two statements will be useful:

- For any set of events $\{E_i\}_{i=1}^N$, the probability that at least m of them occur is at most $\frac{1}{m}\sum_{i=1}^N \Pr(E_i)$.[3]
- $\Pr\left(|\cos\theta| \leq \left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right) = 1 - \frac{2}{\pi}\cos^{-1}\left(\left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right)$.

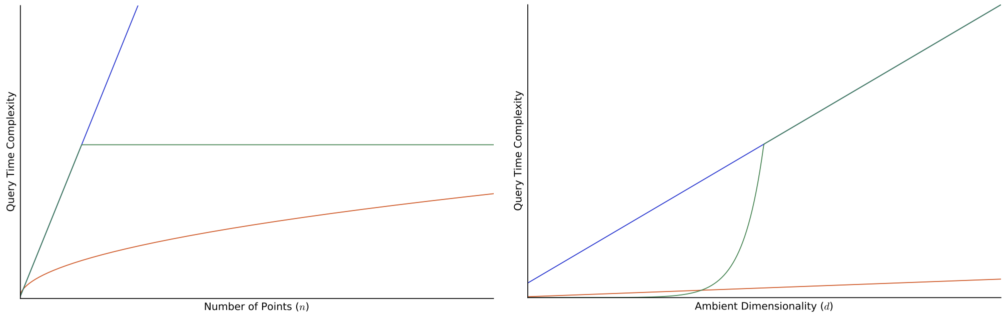
Using the first statement, derive an upper bound on the probability that the algorithm fails. Use $\mathbf{x}^{(i)}$ to denote the ith closest point to the query $\mathbf{z}$. Then use the second statement to simplify the expression.

Solution:

$$
\begin{aligned}
&\Pr(\text{algorithm fails}) \\
&= \Pr\left(\text{at least } \tilde{k} \text{ points are closer to } \mathbf{z} \text{ than } \mathbf{x}^{(1)} \text{ under projection } \mathbf{u}\right) \\
&\leq \frac{1}{\tilde{k}}\sum_{i=2}^{n} \Pr\left(\mathbf{x}^{(i)} \text{ is closer to } \mathbf{z} \text{ than } \mathbf{x}^{(1)} \text{ under projection } \mathbf{u}\right) \\
&= \frac{1}{\tilde{k}}\sum_{i=2}^{n} \Pr\left(\left|\langle \mathbf{x}^{(i)} - \mathbf{z}, \mathbf{u}\rangle\right| \leq \left|\langle \mathbf{x}^{(1)} - \mathbf{z}, \mathbf{u}\rangle\right|\right) \\
&\leq \frac{1}{\tilde{k}}\sum_{i=2}^{n} 1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\left\|\mathbf{x}^{(1)} - \mathbf{z}\right\|_2}{\left\|\mathbf{x}^{(i)} - \mathbf{z}\right\|_2}\right) \leq \frac{1}{\tilde{k}}\sum_{i=2}^{n} \frac{\left\|\mathbf{x}^{(1)} - \mathbf{z}\right\|_2}{\left\|\mathbf{x}^{(i)} - \mathbf{z}\right\|_2}
\end{aligned}
$$

[3] This is a generalization of the union bound; the statement reduces to the union bound when $k' = 1$. (See this paper Ke Li and Jitendra Malik. Fast k-Nearest neighbor Search via Prioritized DCI. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2081–2090, 2017.)

(f) The following plots show the query time complexities of naïve exhaustive search, space partitioning, and DCI as functions of n and d. Curves of the same colour correspond to the same algorithm. (Assume that the failure probability of DCI is small) **Which algorithm does each colour correspond to?**



Solution: The blue curve corresponds to naïve exhaustive search, since its time complexity is linear in both n and d. The green curve corresponds to space partitioning, since its time complexity is exponential in d and constant in n when $3^d < n$. The red curve corresponds to DCI because its time complexity is sublinear in n and linear in the ambient dimension d. Using techniques like DCI can make nearest-neighbor algorithms useful in cases where otherwise, the underlying dimensionalities involved would make the computation prohibitive. Of course, the number of samples required to use nearest-neighbor algorithms effectively is still quite huge because we rely on having a good representation in all localities of the problem.

(g) (Bonus) We will now prove the second statement from (d). **Derive the range of θ such that $|\cos\theta| \leq \left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2$ and show that $\Pr\left(|\cos\theta| \leq \left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right) = 1 - \frac{2}{\pi}\cos^{-1}\left(\left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right)$.**

Hint: Due to rotational invariance of a uniform distribution on the sphere, the angle between $\mathbf{u}^{\|}$ and any vector in P is uniformly distributed.

This part shows that the relative ordering of two data points is more likely to flip if their distances to the query are not very different. Thus, the nearest neighbor search problem is harder if all data points are almost equidistant from the query. The intrinsic hardness of the problem is characterized by the distribution of distances to the query.

Solution:

On $[-\pi, \pi]$, $|\cos\theta| \leq \tau$ if $\theta \in \left[\cos^{-1}(\tau), \pi - \cos^{-1}(\tau)\right] \cup \left[-\pi + \cos^{-1}(\tau), -\cos^{-1}(\tau)\right]$. So, $|\cos\theta| \leq \left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2$ when $\theta \in \left[\cos^{-1}\left(\left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right), \pi - \cos^{-1}\left(\left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right)\right] \cup \left[-\pi + \cos^{-1}\left(\left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right), -\cos^{-1}\left(\left\|\mathbf{v}^s\right\|_2 / \left\|\mathbf{v}^l\right\|_2\right)\right]$. It follows that

$$\Pr\left(|\cos\theta| \leq \frac{\left\|\mathbf{v}^s\right\|_2}{\left\|\mathbf{v}^l\right\|_2}\right) = \Pr\left(\theta \in \left[\cos^{-1}\left(\frac{\left\|\mathbf{v}^s\right\|_2}{\left\|\mathbf{v}^l\right\|_2}\right), \pi - \cos^{-1}\left(\frac{\left\|\mathbf{v}^s\right\|_2}{\left\|\mathbf{v}^l\right\|_2}\right)\right]\right)$$

$$+ \Pr\left(\theta \in \left[-\pi + \cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right), -\cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right)\right]\right)$$

$$= 2\Pr\left(\theta \in \left[\cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right), \pi - \cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right)\right]\right)$$

The length of the interval $\left[\cos^{-1}\left(\|\mathbf{v}^s\|_2 / \|\mathbf{v}^l\|_2\right), \pi - \cos^{-1}\left(\|\mathbf{v}^s\|_2 / \|\mathbf{v}^l\|_2\right)\right]$ is:

$$\left(\pi - \cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right)\right) - \cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right) = \pi - 2\cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right)$$

Since θ is uniformly distributed on $[-\pi, \pi]$,

$$\Pr\left(\theta \in \left[\cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right), \pi - \cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right)\right]\right) = \frac{\pi - 2\cos^{-1}\left(\|\mathbf{v}^s\|_2 / \|\mathbf{v}^l\|_2\right)}{2\pi}$$

$$= \frac{1}{2}\left(1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right)\right)$$

Therefore,

$$\Pr\left(|\cos\theta| \leq \frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right) = 2\Pr\left(\theta \in \left[\cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right), \pi - \cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right)\right]\right)$$

$$= 1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|\mathbf{v}^s\|_2}{\|\mathbf{v}^l\|_2}\right)$$

(h) (Bonus) Notice that the failure probability of the randomized algorithm does not depend on dimensionality at all. It only depends on the distribution of distances from every point to the query, which measures the intrinsic hardness of the problem.

What's a typical distribution of distances? Natural data usually lies on a manifold, which is a generalization of Euclidean subspace that can be "curved" (more concretely, there is a neighborhood around every point on the manifold that resembles a low-dimensional Euclidean space). For simplicity, we'll consider the case when the data is uniformly distributed on a d'-dimensional subspace, where d' is much less than the ambient dimensionality d. Often, d' is known as the intrinsic dimensionality. Then the number of points inside a ball of radius r

is roughly $cr^{d'}$ for some constant c. So, the number of points inside a ball of constant radius grows exponentially in d'.

Assume for all r such that $cr^{d'}$ is an integer, the number of points inside a ball centerd at $\mathbf{z}$ of radius r is exactly $cr^{d'}$. This is equivalent to saying $\|\mathbf{x}^{(cr^{d'})} - \mathbf{z}\|_2 = r$ for any such r. (If we recall from the previous part, $\mathbf{x}^{(i)}$ denotes the ith closest point to $\mathbf{z}$.) **Show the quantity $\sum_{i=2}^{n} \|\mathbf{x}^{(1)} - \mathbf{z}\|_2 / \|\mathbf{x}^{(i)} - \mathbf{z}\|_2$ in this case is like $\sum_{i=2}^{n} (1/i)^{1/d'}$.**

Hint: to derive an expression for $\|\mathbf{x}^{(i)} - \mathbf{z}\|_2$ in terms of i, substitute i for $cr^{d'}$ in the equality $\|\mathbf{x}^{(cr^{d'})} - \mathbf{z}\|_2 = r$.

Solution:

Since $\left\|\mathbf{x}^{(cr^{d'})} - \mathbf{z}\right\|_2 = r$, if we use i to denote $cr^{d'}$, $\left\|\mathbf{x}^{(i)} - \mathbf{z}\right\|_2 = (i/c)^{1/d'}$. Then,

$$\sum_{i=2}^{n} \frac{\left\|\mathbf{x}^{(1)} - \mathbf{z}\right\|_2}{\left\|\mathbf{x}^{(i)} - \mathbf{z}\right\|_2} = \sum_{i=2}^{n} \frac{(1/c)^{1/d'}}{(i/c)^{1/d'}} = \sum_{i=2}^{n} \left(\frac{1}{i}\right)^{1/d'}.$$

Notice that here, we leaned very heavily on our assumption of how distances are going to be distributed with intrinsic dimension d'. It is possible to use bounding-type arguments to make things more precise while preserving this flavor of result. Practically speaking, whenever you make such assumptions, you might want to back up your assumptions of scaling with a numerical simulation.

(i) (Bonus) **Show the quantity $\sum_{i=2}^{n} (1/i)^{1/d'}$ is less than $(n^{1-1/d'} - 1)/\left(1 - 1/d'\right)$.**

Hint: use the fact that $\sum_{i=a}^{b} \phi(i) = \int_a^{b+1} \phi(\lfloor t \rfloor) dt$ for any function ϕ and $t - 1 < \lfloor t \rfloor$ for any t, where $\lfloor \cdot \rfloor$ denotes the floor operator.

Solution:

Using the hint, we first rewrite the summation as an integral:

$$\sum_{i=2}^{n} \left(\frac{1}{i}\right)^{1/d'} = \int_2^{n+1} \left(\frac{1}{\lfloor t \rfloor}\right)^{1/d'} dt$$

Since $t - 1 < \lfloor t \rfloor$ for any t, $1/(t-1) > 1/\lfloor t \rfloor$ and so $\left(1/(t-1)\right)^{1/d'} > \left(1/\lfloor t \rfloor\right)^{1/d'}$. Hence,

$$\begin{aligned}
\int_2^{n+1} \left(\frac{1}{\lfloor t \rfloor}\right)^{1/d'} dt &< \int_2^{n+1} \left(\frac{1}{t-1}\right)^{1/d'} dt \\
&= \int_2^{n+1} (t-1)^{-1/d'} dt \\
&= \int_1^{n} s^{-1/d'} ds \\
&= \left.\frac{s^{1-1/d'}}{1 - 1/d'}\right|_1^n
\end{aligned}$$

$$= \left(\frac{n^{1-1/d'}}{1-1/d'} - \frac{1}{1-1/d'} \right)$$
$$= \frac{n^{1-1/d'} - 1}{1-1/d'}$$

(j) (Bonus) **Show the failure probability is at most $O(n^{1-1/d'}/\tilde{k})$ for $d' \geq 2$.**

Solution: By the parts above, $\sum_{i=2}^{n} \left(\left\| \mathbf{x}^{(1)} - \mathbf{z} \right\|_2 \right) / \left(\left\| \mathbf{x}^{(i)} - \mathbf{z} \right\|_2 \right) = \sum_{i=2}^{n} (1/i)^{1/d'} < \left(n^{1-1/d'} - 1 \right) / \left(1 - 1/d' \right)$.

Since $d' \geq 2$,

$$\begin{aligned} \frac{n^{1-1/d'} - 1}{1-1/d'} &\leq 2\left(n^{1-1/d'} - 1 \right) \\ &< 2n^{1-1/d'} \\ &\in O\left(n^{1-1/d'} \right) \end{aligned}$$

Therefore, we conclude that:

$$\sum_{i=2}^{n} \frac{\left\| \mathbf{x}^{(1)} - \mathbf{z} \right\|_2}{\left\| \mathbf{x}^{(i)} - \mathbf{z} \right\|_2} = \sum_{i=2}^{n} \left(\frac{1}{i} \right)^{1/d'} \in O\left(n^{1-1/d'} \right)$$

Since the failure probability is upper bounded by $\left(1/\tilde{k} \right) \sum_{i=2}^{n} \left(\left\| \mathbf{x}^{(1)} - \mathbf{z} \right\|_2 / \left\| \mathbf{x}^{(i)} - \mathbf{z} \right\|_2 \right)$, it follows that the failure probability is at most $O\left(n^{1-1/d'}/\tilde{k} \right)$.

3 Regularization and Risk Minimization

(a) Let $\mathbf{A}$ be a $d \times n$ matrix. For any $\mu > 0$, show that $(\mathbf{A}\mathbf{A}^\top + \mu\mathbf{I})^{-1}\mathbf{A} = \mathbf{A}(\mathbf{A}^\top\mathbf{A} + \mu\mathbf{I})^{-1}$.
Solution:

$$\begin{aligned} (AA^\top + \mu I)^{-1}A &= A(A^\top A + \mu I)^{-1} \\ A &= (AA^\top + \mu I)A(A^\top A + \mu I)^{-1} \\ A(A^\top A + \mu I) &= (AA^\top + \mu I)A \\ AA^\top A + \mu A &= AA^\top A + \mu A \end{aligned}$$

(b) Let $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$ be a sequence of data points. Each y_i is a scalar and each $\mathbf{x}_i$ is a vector in $\mathbb{R}^d$. Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^\top$ and $\mathbf{y} = [y_1, \ldots, y_n]^\top$. Consider the *regularized* least squares problem.

$$\min_{\mathbf{w}\in\mathbb{R}^d} \|\mathbf{X}\mathbf{w}-\mathbf{y}\|_2^2 + \mu\|\mathbf{w}\|_2^2$$

Show that the optimum $\mathbf{w}_*$ is unique and can be written as the linear combination $\mathbf{w}_* = \sum_{i=1}^n \alpha_i \mathbf{x}_i$ for some scalars $\alpha_1, ..., \alpha_n$. What are the coefficients α_i?

Solution: Start by taking the gradient of the loss function as follows:

$$\begin{aligned}\nabla_W(\|Xw-y\|_2^2 + \mu\|w\|_2^2) &= (Xw-y)^\top X + \mu w^\top \\ &= W^\top X^\top X - Y^\top X + \mu w^\top \\ &= X^\top Xw - X^\top y + \mu w = 0\end{aligned}$$

$$\begin{aligned}w &= (X^\top X + \mu I)^{-1}X^\top y \\ w &= X^\top(XX^\top + \mu I)^{-1}y\end{aligned}$$

Recall that $(XX^\top + \mu I)$ is positive definite and has real, positive eigenvalues when $\mu > 0$. The invertibility of this matrix implies a unique solution for w_*. $(X^\top X + \mu I)$ can thus be diagonalized into the form $U\Lambda U^\top$ by the Spectral Theorem, with $U\Lambda^{-1}U^\top$ as its inverse. This allows us to write

$$w_* = \sum_{i=1}^n \alpha_i x_i$$

where

$$\alpha_i = \sum_{j=1}^d y_j * u_i\Lambda^{-1}u_j^\top$$

and u_i is the i^{th} row of U.

(c) More generally, consider the general regularized empirical risk minimization problem

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^n \text{loss}(\mathbf{w}^\top\mathbf{x}_i, y_i) + \mu\|\mathbf{w}\|_2^2$$

where the loss function is convex in the first argument. Prove that the optimal solution has the form $\mathbf{w}_* = \sum_{i=1}^n \alpha_i\mathbf{x}_i$. If the loss function is not convex, does the optimal solution have the form $\mathbf{w}_* = \sum_{i=1}^n \alpha_i\mathbf{x}_i$? Justify your answer.

Solution: The expression $w_* = \sum_{i=1}^n \alpha_i x_i$ writes w_* as a linear combination of the columns of $X^\top$. Suppose the optimal weight vector w_* does not lie in the subspace spanned by the columns of $X^\top$ and so can be written as $w_*' = \sum_{i=1}^n \alpha_i x_i + v = w_* + v$, where $v^\top x_i = 0$ for x_i's. Using w_*' as our predictor, our loss function becomes

$$\frac{1}{n}\sum_{i=1}^n \text{loss}((w_*^\top + v^\top)x_i, y_i) + \mu\|w+v\|_2^2$$

$$\frac{1}{n}\sum_{i=1}^{n}\text{loss}((w_*^\top)x_i, y_i) + \mu(\|w\|_2^2 + \|v\|_2^2)$$

v, being orthogonal to w and the x_i's, should be set to 0 to minimize the objective function. Hence, the optimal solution has the form $w_* = \sum_{i=1}^{n}\alpha_i x_i$, which is true in the cases of both convex and non-convex loss functions.

4 Convergence Rate of Gradient Descent

In homework 5 we showed that gradient descent converges for *quadratic* functions, in this problem we will show that for any strongly convex function, gradient descent converges quickly.

First, let us go through some definitions. It is possible to define a convex function f as one where, for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$,

$$f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) \leq f(\mathbf{y}).$$

Geometrically, this means that if we start at $(\mathbf{x}, f(\mathbf{x}))$ and move along the gradient towards $(\mathbf{y}, f(\mathbf{y}))$, we end up "below" the actual value of $f(\mathbf{y})$. We can extend this definition to a stronger one, saying that a function f is α-strongly convex if it satisfies

$$f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{\alpha}{2}\|\mathbf{x} - \mathbf{y}\|^2 \leq f(\mathbf{y}).$$

Finally, a function f is β-smooth if for any $\mathbf{x}, \mathbf{y} \in R^n$, one has

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq \beta\|\mathbf{x} - \mathbf{y}\|.$$

(a) Let φ be a β-smooth convex function. Show that

$$(\nabla\varphi(\mathbf{x}) - \nabla\varphi(\mathbf{y}))^\top(\mathbf{x} - \mathbf{y}) \geq \frac{1}{\beta}\|\nabla\varphi(\mathbf{x}) - \nabla\varphi(\mathbf{y})\|^2$$

Solution: We need to show that lemma 3.5 can be used. By β smoothness we know that the upper bound holds. It suffices to show that the lower bound is 0. This will follow from convexity:

$$\varphi((x)) + \nabla\varphi((x))^\top(y - x) \leq f(y)$$
$$0 \leq \nabla\varphi((x))^\top(y - x) + f(y) - (fx)$$

Next we apply lemma 2 twice.

$$(\nabla\varphi(x) - \nabla\varphi(y))^\top(x - y)$$
$$(\nabla\varphi(x))^\top(x - y) + (\nabla\varphi(y))^\top(y - x)$$

Applying lemma 3.5 to each of the individual terms in the sum above gives us our desired result

$$\leq \varphi(x) - \varphi(y) + \frac{1}{2\beta}\|\nabla\varphi(x) - \nabla\varphi(y)\|^2 + \varphi(y) - \varphi(x) + \frac{1}{2\beta}\|\nabla\varphi(x) - \nabla\varphi(y)\|^2$$

$$\leq \frac{1}{\beta}\|\nabla\varphi(x) - \nabla\varphi(y)\|^2$$

(b) Let f be β-smooth and α-strongly convex. Show that $\varphi(\mathbf{x}) = f(\mathbf{x}) - \frac{\alpha}{2}\|\mathbf{x}\|^2$ is convex and $(\beta - \alpha)$-smooth.

Solution: Due to strong convexity, f satisfies the following

$$f(\mathbf{x}) - f(\mathbf{y}) \leq \nabla f(\mathbf{x})^\top(\mathbf{x} - \mathbf{y}) - \frac{\alpha}{2}\|\mathbf{x} - \mathbf{y}\|^2$$

Then we have

$$\begin{aligned}
f(x) - f(y) &\leq \nabla f(x)^\top(x - y) - \frac{\alpha}{2}\|x\|^2 - \frac{\alpha}{2}\|y\|^2 + \alpha(x^\top y)\\
(f(x) - \frac{\alpha}{2}\|x\|^2) - (f(y) - \frac{\alpha}{2}\|y\|^2) &\leq \nabla f(x)^\top(x - y) - \alpha\|x\|^2 + \alpha(x^\top y)\\
&= \nabla f(x)^\top(x - y) - \alpha x^\top(x - y)\\
&= (\nabla f(x) - \alpha x)^\top(x - y)\\
\varphi(x) - \varphi(y) &\leq \nabla\varphi(x)^\top(x - y)
\end{aligned}$$

And thus we conclude that φ is convex.

$$\begin{aligned}
\|\nabla\varphi(x) - \nabla\varphi(y)\|^2 &= \|(\nabla f(x) - \alpha x) - (\nabla f(y) - \alpha y)\|^2\\
&= \|\nabla f(x) - \nabla f(y)\|^2 - 2\alpha(\nabla f(x) - \nabla f(y))^\top(x - y) + \alpha^2\|x - y\|^2\\
&\leq \|\nabla f(x) - \nabla f(y)\|^2 - 2\alpha/\beta\|\nabla f(x) - \nabla f(y)\|^2 + \alpha^2\|x - y\|^2\\
&\leq ((1 - 2\frac{\alpha}{\beta})\beta^2 + \alpha^2)\|x - y\|^2\\
&= (\beta - \alpha)^2\|x - y\|^2\\
\|\nabla\varphi(x) - \nabla\varphi(y)\| &\leq (\beta - \alpha)\|x - y\|
\end{aligned}$$

And thus we conclude that φ is $(\beta - \alpha)$-smooth.

(c) Show that for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, one has

$$(\nabla f(\mathbf{x}) - \nabla f(\mathbf{y}))^\top(\mathbf{x} - \mathbf{y}) \geq \frac{\alpha\beta}{\beta + \alpha}\|\mathbf{x} - \mathbf{y}\|^2 + \frac{1}{\beta + \alpha}\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|^2$$

Solution: Plugging in $\varphi(x) = f(x) - \frac{\alpha}{2}\|x\|^2$ to the part (a), we have

$$\begin{aligned}
((\nabla f(x) - \alpha x) - (\nabla f(y) - \alpha y))^\top(x - y) &\geq \frac{1}{\beta - \alpha}\|(\nabla f(x) - \alpha x) - (\nabla f(y) - \alpha y)\|^2\\
&= \frac{1}{\beta - \alpha}\|\nabla f(x) - \nabla f(y)\|^2\\
&\quad - 2\frac{\alpha}{\beta - \alpha}(\nabla f(x) - \nabla f(y))^\top(x - y)
\end{aligned}$$

$$
\begin{aligned}
&+\frac{\alpha^2}{\beta-\alpha}\|x-y\|^2 \\
(1+2\frac{\alpha}{\beta-\alpha})(\nabla f(x)-\nabla f(y))^\top(x-y) &\geq \frac{1}{\beta-\alpha}\|\nabla f(x)-\nabla f(y)\|^2+(\frac{\alpha^2}{\beta-\alpha}+\alpha)\|x-y\|^2 \\
(\beta+\alpha)(\nabla f(x)-\nabla f(y))^\top(x-y) &\geq \|\nabla f(x)-\nabla f(y)\|^2+\alpha\beta\|x-y\|^2 \\
(\nabla f(x)-\nabla f(y))^\top(x-y) &\geq \frac{1}{\beta+\alpha}\|\nabla f(x)-\nabla f(y)\|^2+\frac{\alpha\beta}{\beta+\alpha}\|x-y\|^2
\end{aligned}
$$

(d) Let f be β-smooth and α-strongly convex, and denote $Q=\frac{\beta}{\alpha}$. Then show that gradient descent with fixed step size $\eta=\frac{2}{\alpha+\beta}$ satisfies

$$f(\mathbf{x}_t)-f(\mathbf{x}^*)\leq\frac{\beta}{2}\left(\frac{Q-1}{Q+1}\right)^{2(t-1)}\|\mathbf{x}_1-\mathbf{x}^*\|^2$$

Hint: By β-smoothness, you have

$$f(\mathbf{x}_t)-f(\mathbf{x}^*)\leq\frac{\beta}{2}\|\mathbf{x}_t-\mathbf{x}^*\|^2$$

Solution:

$$
\begin{aligned}
\|x_t-x^*\|^2 &= \|x_{t-1}-\eta\nabla f(x_{t-1})-x^*\|^2 \quad \text{(gradient descent step)} \\
&= \|x_{t-1}-x^*\|^2-2\eta\nabla f(x_{t-1})^\top(x_{t-1}-x^*)+\eta^2\|\nabla f(x_{t-1})\|^2 \quad \text{(expand)}
\end{aligned}
$$

Using part (c), we have that

$$
\begin{aligned}
(\nabla f(x_{t-1})-\nabla f(x^*))^\top(x_{t-1}-x^*) &\geq \frac{\alpha\beta}{\beta+\alpha}\|x_{t-1}-x^*\|^2+\frac{1}{\beta+\alpha}\|\nabla f(x_{t-1})-\nabla f(x^*)\|^2 \\
\nabla f(x_{t-1})^\top(x_{t-1}-x^*) &\geq \frac{\alpha\beta}{\beta+\alpha}\|x_{t-1}-x^*\|^2+\frac{1}{\beta+\alpha}\|\nabla f(x_{t-1})\|^2
\end{aligned}
$$

since $\nabla f(x^*)=0$. Then, we have that

$$
\begin{aligned}
\|x_t-x^*\|^2 &\leq (1-2\eta\frac{\alpha\beta}{\beta+\alpha})\|x_{t-1}-x^*\|^2+(\frac{2\eta}{\beta+\alpha}+\eta^2)\|\nabla f(x_{t-1})\|^2 \\
&= \left(\frac{Q-1}{Q+1}\right)^2\|x_{t-1}-x^*\|^2
\end{aligned}
$$

Then we conclude that for strongly convex functions, gradient descent gives a fast geometric convergence rate.

$$
\begin{aligned}
f(x_t)-f(x^*) &\leq \frac{\beta}{2}\|x_t-x^*\|^2 \\
&\leq \frac{\beta}{2}\left(\frac{Q-1}{Q+1}\right)^2\|x_{t-1}-x^*\|^2 \\
&\leq \frac{\beta}{2}\left(\frac{Q-1}{Q+1}\right)^{2(t-1)}\|x_1-x^*\|^2
\end{aligned}
$$