CS 189 Introduction to Machine Learning HW5
Fall 2018

This homework is due **Wednesday, October 10th at 10pm.**

2 Step Size in Gradient Descent

In this problem, we will look at the convex function $f(\mathbf{x}) = ||\mathbf{x}-\mathbf{b}||_2$. Note that we are using "just" the regular Euclidean ℓ_2 norm, *not* the norm squared! This problem illustrates the importance of understanding how gradient descent works and choosing step sizes strategically. In fact, there is a lot of active research in variations on gradient descent. Throughout the question we will look at different kinds of step sizes. We will also look at the the rate at which the different step sizes decrease and draw some conclusions about the rate of convergence. You have been provided with a tool in `step_size_helper.py` which will help you visualize the problems below.

(a) Let $\mathbf{x}, \mathbf{b} \in \mathbb{R}^d$. **Prove that $f(\mathbf{x}) = \|\mathbf{x}-\mathbf{b}\|_2$ is a convex function of $\mathbf{x}$.**

Solution: Recall that a function is convex if for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$ we have

$$f(\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \le \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)$$

for all $0 \le \lambda \le 1$. For $f(\mathbf{x}) = \|\mathbf{x}-\mathbf{b}\|_2$, the triangle inequality gives us:

$$\begin{aligned}\|\lambda\mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 - \mathbf{b}\|_2 &= \|\lambda(\mathbf{x}_1-\mathbf{b}) + (1-\lambda)(\mathbf{x}_2-\mathbf{b})\|_2 \\ &\le \|\lambda(\mathbf{x}_1-\mathbf{b})\|_2 + \|(1-\lambda)(\mathbf{x}_2-\mathbf{b})\|_2 \\ &= \lambda\|\mathbf{x}_1-\mathbf{b}\|_2 + (1-\lambda)\|\mathbf{x}_2-\mathbf{b}\|_2,\end{aligned}$$

which shows that f is convex.

(b) **For $\nabla_{\mathbf{x}} f(\mathbf{x})$, determine where it is well-defined and compute its value.**

Solution:

$$\begin{aligned}\frac{\partial}{\partial x_i} f(\mathbf{x}) &= \frac{\partial}{\partial x_i}\sqrt{\sum_{i=1}^{d}(x_i-b_i)^2} \\ &= (x_i-b_i)\left(\sum_{i=1}^{d}(x_i-b_i)^2\right)^{-1/2} \\ &= \frac{x_i-b_i}{\|\mathbf{x}-\mathbf{b}\|_2}.\end{aligned}$$

Note that this is only well-defined when $\mathbf{x} \neq \mathbf{b}$. Thus $\nabla_{\mathbf{x}} f(\mathbf{x}) = \frac{\mathbf{x}-\mathbf{b}}{\|\mathbf{x}-\mathbf{b}\|_2}$ when $\mathbf{x} \neq \mathbf{b}$.

(c) We are minimizing $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{b}\|_2$, where $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{b} = [6, 4.5] \in \mathbb{R}^2$, with gradient descent. We use a constant step size of $t_i = 1$. That is,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - t_i \nabla f(\mathbf{x}_i) = \mathbf{x}_i - \nabla f(\mathbf{x}_i).$$

We start at $\mathbf{x}_0 = [0, 0]$. **Will gradient descent find the optimal solution? If so, how many steps will it take to get within 0.01 of the optimal solution (for this and subsequent problems, this is in terms of $\|\mathbf{x}_\mathbf{k} - \mathbf{x}_*\|_2$)? If not, why not?** Prove your answer. (Hint: use the tool to compute the first ten steps.) **What about general $\mathbf{b} \neq \mathbf{0}$?**

Solution: Using the tool provided (or computing gradients by hand), we get

$$\mathbf{x}_6 = \mathbf{x}_8 = [5.6, 4.2]$$

and

$$\mathbf{x}_7 = \mathbf{x}_9 = [6.4, 4.8].$$

Examine the formula

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \nabla f(\mathbf{x}_i)$$

and notice that $\mathbf{x}_{i+1}$ only depends on $\mathbf{x}_i$, regardless of what step we are on (this is only the case because we are using a constant step size). It means that if $\mathbf{x}_i = \mathbf{x}_j$, then $\mathbf{x}_{i+1} = \mathbf{x}_{j+1}$. Thus, this pattern will repeat indefinitely and we do not reach the optimal solution. It is also helpful to notice that

$$\|\mathbf{x}_6\|, \|\mathbf{x}_7\| \in \mathbb{Z}.$$

In general, notice that

$$-\nabla f(\mathbf{x}_i) = \frac{\mathbf{b} - \mathbf{x}_i}{\|\mathbf{x}_i - \mathbf{b}\|}$$

will always have unit length. In fact, we can prove by induction that $\mathbf{x}_k$ is always an integer multiple of the unit vector $\frac{\mathbf{b}}{\|\mathbf{b}\|}$. Thus,

$$\|\mathbf{x}_k\| \in \mathbb{Z}.$$

If we are lucky and $\|\mathbf{b}\|$ happens to be an integer (or very close to integer), then we will reach or get near the optimal solution. Otherwise, our step size is too coarse to hit it.

In fact, for the function $\|\mathbf{x} - \mathbf{b}\|$, a constant step size will rarely work. We are too prone to "jumping over" our solution. A decreasing step size is necessary and we will investigate decreasing stepsizes next.

(d) We are minimizing $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{b}\|_2$, where $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{b} = [6, 4.5] \in \mathbb{R}^2$, now with a decreasing step size of $t_i = (\frac{6}{7})^i$ at step i. That is,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - t_i \nabla f(\mathbf{x}_i) = \mathbf{x}_i - (\frac{6}{7})^i \nabla f(\mathbf{x}_i).$$

We start at $\mathbf{x}_0 = [0, 0]$. **Will gradient descent find the optimal solution? If so, how many steps will it take to get within 0.01 of the optimal solution? If not, why not?** Prove your answer. (Hint: examine $\|\mathbf{x}_i\|_2$.) **What about general $\mathbf{b} \neq \mathbf{0}$?**

Solution: Notice that we can express $\mathbf{x}_{i+1}$ as the sum of all the steps taken, so we can express its norm as

$$\|\mathbf{x}_{i+1}\| = \|\mathbf{x}_0 - \sum_{j=0}^{i} (\frac{6}{7})^j \nabla f(\mathbf{x}_i)\| \leq \|\mathbf{x}_0\| + \sum_{j=0}^{i} \|(\frac{6}{7})^j \nabla f(\mathbf{x}_i)\|$$

But all of $\|\nabla f(\mathbf{x}_i)\|$ are exactly 1, so we can write this as

$$\|\mathbf{x}_{i+1}\| \leq \|\mathbf{x}_0\| + \sum_{j=0}^{i} (\frac{6}{7})^j \|\nabla f(\mathbf{x}_i)\| = 0 + \sum_{j=0}^{i} (\frac{6}{7})^j \leq 7.$$

In words, we can never "travel" a distance of more than 7 from $\mathbf{0}$, so we will never get to our optimal solution $\mathbf{b}$ if $\|\mathbf{b}\| > 7$. In our case, $\|[6, 4.5]\|_2 = 7.5 > 7$, so we will never reach $\mathbf{b}$. In conclusion, while we need a decreasing step size, we also have to be sure it does not decrease too quickly.

(e) We are minimizing $f(\mathbf{x}) = \|\mathbf{x} - \mathbf{b}\|_2$, where $\mathbf{x} \in \mathbb{R}^2$ and $\mathbf{b} = [6, 4.5] \in \mathbb{R}^2$, now with a decreasing step size of $t_i = \frac{1}{i+1}$ at step i. That is,

$$\mathbf{x}_{i+1} = \mathbf{x}_i - t_i \nabla f(\mathbf{x}_i) = \mathbf{x}_i - \frac{1}{i+1} \nabla f(\mathbf{x}_i).$$

We start at $\mathbf{x}_0 = [0, 0]$. **Will gradient descent find the optimal solution? If so, how many steps will it take to get within** 0.01 **of the optimal solution? If not, why not?** Prove your answer. (Hint: examine $\|\mathbf{x}_i\|_2$, and use $\sum_{i=1}^{n} \frac{1}{i}$ is of the order $\log n$.) **What about general $\mathbf{b} \neq 0$? State (roughly—up to the correct order) the amount of steps required to reach the optimum in the general case, for whichever cases are possible.**

Solution: The key realization here is that all our gradient steps have $\|\nabla f(\mathbf{x}_i)\| = 1$ and move along the same vector towards $\mathbf{b}$ (to be precise, we could prove by induction that our $\mathbf{x}_i$ and $\nabla f(\mathbf{x}_i)$ will always be in the span of $\mathbf{b}$). We can write

$$\|\mathbf{x}_{i+1}\| = \|\mathbf{x}_0 - \sum_{j=0}^{i} (\frac{1}{j+1}) \nabla f(\mathbf{x}_i)\| = \sum_{j=0}^{i} \|(\frac{1}{j+1}) \nabla f(\mathbf{x}_i)\| = \sum_{j=1}^{i} \frac{1}{j+1} \approx \ln i.$$

This sum diverges, so we will reach the optimal solution $\mathbf{b}$ eventually, but it could take quite a while. For $\mathbf{b} = [6, 4.5]$ it takes about 1000 steps (1004 to get within $\|\cdot\| \leq 0.01$ of the optimal $\mathbf{x}$—7.49046, to be exact). In generally, the number of steps it takes will be exponential in $\|\mathbf{b}\|$, since we have

$$\ln i \approx \|\mathbf{b}\| \implies i \approx e^{\|\mathbf{b}\|}.$$

We may not hit $\mathbf{b}$ exactly. Once we exceed $\mathbf{b}$, we start oscillating around it. Let i_0 be the first step where we "cross over" $\mathbf{b}$. After that, we know we are always moving towards $\mathbf{b}$ (whether we are stepping backwards or forwards), so after step i (assuming $i > i_0$), we can be at most $\frac{1}{i+1}$ from the optimal solution (if we are more than $\frac{1}{i+1}$ away, that means our previous step went in the wrong direction).

(f) Now, say we are minimizing $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$. Use the code provided to test several values of $\mathbf{A}$ with the step sizes suggested above. Make plots to visualize what is happening. We suggest trying $\mathbf{A} = [[1, 0], [0, 10]]$ and $\mathbf{A} = [[6, 5], [15, 8]]$. **Will any of the step sizes above work (i.e. eventually be within any presribed distance ϵ) for all choices of A and b?** You do not need to prove your answer, but you should briefly explain your reasoning.

Solution: The first two step sizes did not work for $\mathbf{A} = \mathbf{I}$, so they will not work in general. The last step sizes *does* always work, though it may take a very long time to converge.

A perspective to notice is that the $\mathbf{x}_i$ are still ultimately moving towards $\mathbf{x}^*$, but not on a straight line but instead on a curved path. The path has finite length, so we will eventually reach $\mathbf{x}^*$. Once we are close to $\mathbf{x}^*$, successive steps will oscillate around $\mathbf{x}^*$.

Formally: This was not asked in the question, but here is the proof: We have

$$\nabla f(\mathbf{x}) = \frac{\mathbf{A}^\top(\mathbf{A}\mathbf{x} - \mathbf{b})}{\|\mathbf{A}\mathbf{x} - \mathbf{b}\|}$$

and therefore $\|\nabla f(\mathbf{x})\|_2^2 \le \sigma_{max}^2(\mathbf{A})$. From convexity, we furthermore have

$$\nabla f(\mathbf{x}_i)^\top(\mathbf{x}^* - \mathbf{x}_i) \le f(\mathbf{x}^*) - f(\mathbf{x}_i)$$

which yields

$$\begin{aligned}
\|\mathbf{x}_{i+1} - \mathbf{x}^*\|_2^2 &= \|\mathbf{x}_i - t_i\nabla f(\mathbf{x}_i) - \mathbf{x}^*\|_2^2 \\
&= \|\mathbf{x}_i - \mathbf{x}^*\|_2^2 - 2t_i\nabla f(\mathbf{x}_i)^\top(\mathbf{x}_i - \mathbf{x}^*) + t_i^2\|\nabla f(\mathbf{x}_i)\|_2^2 \\
&\le \|\mathbf{x}_i - \mathbf{x}^*\|_2^2 - 2t_i(f(\mathbf{x}_i) - f(\mathbf{x}^*)) + t_i^2\sigma_{max}^2(\mathbf{A}) \\
&\le \|\mathbf{x}_0 - \mathbf{x}^*\|_2^2 - 2\sum_{k=0}^{i} t_k(f(\mathbf{x}_k) - f(\mathbf{x}^*)) + \sigma_{max}^2(\mathbf{A})\sum_{k=0}^{i} t_k^2 \\
&\le \|\mathbf{x}_0 - \mathbf{x}^*\|_2^2 - 2(f(\mathbf{x}_{best}) - f(\mathbf{x}^*))\sum_{k=0}^{i} t_k + \sigma_{max}^2(\mathbf{A})\sum_{k=0}^{i} t_k^2
\end{aligned}$$

where $\mathbf{x}_{best} = \arg\min_{0\le k\le i} f(\mathbf{x}_k)$. We denote $\|\mathbf{x}_0 - \mathbf{x}^*\|_2 = d_0$. This yields

$$0 \le d_0^2 - 2(f(\mathbf{x}_{best}) - f(\mathbf{x}^*))\sum_{k=0}^{i} t_k + \sigma_{max}^2(\mathbf{A})\sum_{k=0}^{i} t_k^2$$

Hence,

$$f(\mathbf{x}_{best}) - f(\mathbf{x}^*) \le \frac{d_0^2 + \sigma_{max}^2(\mathbf{A})\sum_k t_k^2}{\sum_k t_k} = \frac{d_0^2 + \sigma_{max}^2(\mathbf{A})\sum_k \frac{1}{(1+k)^2}}{\sum_k \frac{1}{1+k}} \xrightarrow[i\to\infty]{} 0$$

3 Convergence Rate of Gradient Descent

In the previous problem, you examined $||\mathbf{A}\mathbf{x} - \mathbf{b}||_2$ (without the square). You showed that even though it is convex, getting gradient descent to converge requires some care. In this problem, you will examine $\frac{1}{2}||\mathbf{A}\mathbf{x} - \mathbf{b}||_2^2$ (with the square). You will show that now gradient descent converges quickly.

For a matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$ and a vector $\mathbf{b} \in \mathbb{R}^n$, consider the quadratic function $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ such that $\mathbf{A}^\top\mathbf{A}$ is positive definite.

(a) First, consider the case $\mathbf{b} = \mathbf{0}$, and think of each $\mathbf{x} \in \mathbb{R}^d$ as a "state." Performing gradient descent moves us sequentially through the states, which is called a "state evolution." **Write out the state evolution for n iterations of gradient descent using step-size $\gamma > 0$. That is, explicitly express $\mathbf{x}_n$ as a function of $\mathbf{x}_0$ (and not as a function of any other iterates).**

Solution: *Quick note:* First let us clarify why gradient descent on the norm squared converges faster than gradient descent on the norm. The norm is convex but not strictly convex. Therefore gradient descent will only obtain the slow error rate $O(\frac{1}{\sqrt{k}})$ where k is the number of iterations (we did not prove this, but our step size selection in the last part of the last problem led to a worse bound $O(\frac{1}{\log k})$ and with a more refined analysis, we can get $O(\frac{1}{\sqrt{k}})$). On the other hand, the squared norm is strictly convex and therefore gradient descent can obtain the fast linear error rate $O(\rho^k)$ for $\rho < 1$ as we show in the remainder of this problem.

The general formula for the gradient update is

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \gamma\nabla f(\mathbf{x}_{n-1})$$

which in this case is

$$\mathbf{x}_n = \mathbf{x}_{n-1} - \gamma(\mathbf{A}^\top\mathbf{A}\mathbf{x}_{n-1} - \mathbf{A}^\top\mathbf{b}) = (\mathbf{I} - \gamma\mathbf{A}^\top\mathbf{A})\mathbf{x}_{n-1} - \gamma\mathbf{A}^\top\mathbf{b}.$$

Since we assumed $\mathbf{b} = \mathbf{0}$, this works out to

$$\mathbf{x}_n = (\mathbf{I} - \gamma\mathbf{A}^\top\mathbf{A})\mathbf{x}_{n-1},$$

so by induction we can show

$$\mathbf{x}_n = (\mathbf{I} - \gamma\mathbf{A}^\top\mathbf{A})^n\mathbf{x}_0.$$

(b) A state evolution is said to be *stable* if it does not blow up arbitrarily over time. Specifically, if state n is

$$\mathbf{x}_n = \mathbf{B}^n\mathbf{x}_0$$

then we need *all* the eigenvalues of $\mathbf{B}$ to be less than or equal to 1 in absolute value, otherwise $\mathbf{B}^n$ might blow up $\mathbf{x}_0$ for large enough n. To see this, consider the case when $\mathbf{x}_0$ is an eigenvector of $\mathbf{B}$ corresponding to an eigenvalue with magnitude greater than 1.

When is the state evolution of the iterations you calculated above stable? Express this condition in terms of eigenvalues of $\mathbf{A}^\top\mathbf{A}$.

Solution: An important fact in linear algebra is that if we take a matrix $\mathbf{B}$ with eigenvalues λ_i to the power of k, then the resulting matrix $\mathbf{B}^k$ has eigenvalues λ_i^k with the same eigenvectors. The proof of this fact (inductively) is

$$\mathbf{B}^k\mathbf{v} = \mathbf{B}^{k-1}\mathbf{B}\mathbf{v} = \mathbf{B}^{k-1}\lambda\mathbf{v} = \lambda\mathbf{B}^{k-1}\mathbf{v} = \lambda\lambda^{k-1}\mathbf{v} = \lambda^k\mathbf{v}.$$

Let $\mathbf{B} = (\mathbf{I} - \gamma\mathbf{A}^\top\mathbf{A})$. In order for the state evolution to be stable, we need *all* the eigenvalues of $\mathbf{B}$ to be less than or equal to 1 in absolute value. More formally, this connects to an identity we have previously reviewed, which is

$$\|\mathbf{B}^k\mathbf{v}\|_2 \le |(\lambda_{\max}(\mathbf{B}))^k|\|\mathbf{v}\|_2 = (|\lambda_{\max}(\mathbf{B})|)^k\|\mathbf{v}\|_2.$$

So if the largest eigenvalue of $(\mathbf{I}-\gamma\mathbf{A}^\top\mathbf{A})$ (in absolute value) is bounded by 1, then the system is stable. Thus, we require $|1-\gamma\lambda_{\max}(\mathbf{A}^\top\mathbf{A})| < 1$, or $\lambda_{\max}(\mathbf{A}^\top\mathbf{A}) < 2/\gamma$.

(c) We want to bound the progress of gradient descent in the general case, when $\mathbf{b}$ is arbitrary. To do this, we first show a slightly more general bound, which relates how much the spacing between two points changes if they *both* take a gradient step. If this spacing shrinks, this is called a contraction. Define map taking an iterate to its next step as $\varphi(\mathbf{x}) = \mathbf{x} - \gamma\nabla f(\mathbf{x})$, for some constant step size $\gamma > 0$. **Show that for any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$,**

$$\|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\|_2 \le \rho\|\mathbf{x} - \mathbf{x}'\|_2$$

where $\rho = \max\left\{|1-\gamma\lambda_{\max}(\mathbf{A}^\top\mathbf{A})|, |1-\gamma\lambda_{\min}(\mathbf{A}^\top\mathbf{A})|\right\}$. Note that $\lambda_{\min}(\mathbf{A}^\top\mathbf{A})$ denotes the smallest eigenvalue of the matrix $\mathbf{A}^\top\mathbf{A}$; similarly, $\lambda_{\max}(\mathbf{A}^\top\mathbf{A})$ denotes the largest eigenvalue of the matrix $\mathbf{A}^\top\mathbf{A}$. (Hint: You may use the fact regarding the spectral norm (the "induced 2-norm"): $\|Qz\|_2 \le \|Q\|_2\|z\|_2$, where $\|Q\|_2 := \sigma_1(Q)$.)

Solution: We start off by expanding the left side of the equation we are interested in:

$$\begin{aligned}\|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\|_2 &= \|(\mathbf{x} - \gamma\mathbf{A}^\top(\mathbf{A}\mathbf{x}-\mathbf{b})) - (\mathbf{x}' - \gamma\mathbf{A}^\top(\mathbf{A}\mathbf{x}'-\mathbf{b}))\|_2 \\ &= \|(\mathbf{x}-\mathbf{x}') - \gamma\mathbf{A}^\top\mathbf{A}(\mathbf{x}-\mathbf{x}')\|_2 \\ &= \|(\mathbf{I}-\gamma\mathbf{A}^\top\mathbf{A})(\mathbf{x}-\mathbf{x}')\|_2\end{aligned}$$

Let $\mathbf{B} = \mathbf{I} - \gamma\mathbf{A}^\top\mathbf{A}$. We can now square both sides to obtain

$$\|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\|_2^2 \le (\mathbf{x}-\mathbf{x}')^\top\mathbf{B}^\top\mathbf{B}(\mathbf{x}-\mathbf{x}').$$

Recall the definition of the Rayleigh quotient, by which we know that for a symmetric matrix $\mathbf{X}$, we have

$$R(\mathbf{v}) = \frac{\mathbf{v}^\top\mathbf{X}\mathbf{v}}{\mathbf{v}^\top\mathbf{v}} \le R(\mathbf{v}_1) = \lambda_1(\mathbf{X}),$$

when $\mathbf{v}_1$ is the eigenvector corresponding to the maximum eigenvalue $\lambda_1(\mathbf{X})$ of the matrix $\mathbf{X}$.

Using this fact substituting this in our result, we see that

$$\|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\|_2^2 \le \lambda_1(\mathbf{B}^\top\mathbf{B})\|\mathbf{x}-\mathbf{x}'\|_2^2,$$

and consequently, we have

$$\|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\|_2 \leq \sqrt{\lambda_1(\mathbf{B}^\top\mathbf{B})}\|\mathbf{x} - \mathbf{x}'\|_2.$$

We must now compute the maximum eigenvalue of $\mathbf{B}^\top\mathbf{B}$. We know that the eigenvalues of $\mathbf{B}^\top\mathbf{B}$ are the squared eigenvalues of $\mathbf{B}$. In particular, that means that the maximum eigenvalue of $\mathbf{B}^\top\mathbf{B}$ is either the square of the maximum eigenvalue of $\mathbf{B}$ (when that is large and positive), or the minimum eigenvalue of $\mathbf{B}$ (when that is large and negative). Since the maximum and minimum eigenvalues of $\mathbf{I} - \gamma\mathbf{A}^\top\mathbf{A}$ are given by $1 - \gamma\lambda_{\text{min}}(\mathbf{A}^\top\mathbf{A})$, and $1 - \gamma\lambda_{\text{max}}(\mathbf{A}^\top\mathbf{A})$, we have

$$\lambda_1(\mathbf{B}^\top\mathbf{B}) = \max\left\{(1 - \gamma\lambda_{\text{min}}(\mathbf{A}^\top\mathbf{A}))^2, (1 - \gamma\lambda_{\text{max}}(\mathbf{A}^\top\mathbf{A}))^2\right\}.$$

Taking the square root and combining the pieces, we have

$$\|\varphi(\mathbf{x}) - \varphi(\mathbf{x}')\|_2 \leq \rho\|\mathbf{x} - \mathbf{x}'\|_2.$$

(d) Now we give a bound for progress after k steps of gradient descent. Define

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}).$$

Show that

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_2 = \|\varphi(\mathbf{x}_k) - \varphi(\mathbf{x}^*)\|_2$$

and conclude that

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_2 \leq \rho^{k+1}\|\mathbf{x}_0 - \mathbf{x}^*\|_2.$$

Solution: Note that $\varphi(\mathbf{x}^*) = \mathbf{x}^*$ at optimality and $\varphi(\mathbf{x}_k) = \mathbf{x}_k - \gamma\nabla f(\mathbf{x}_k) = \mathbf{x}_{k+1}$. Then,

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_2 = \|\varphi(\mathbf{x}_k) - \varphi(\mathbf{x}^*)\|_2.$$

Applying the previous part, we have

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_2 \leq \rho\|\mathbf{x}_k - \mathbf{x}^*\|_2.$$

Applying the same bound k times, we have

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\|_2 \leq \rho^{k+1}\|\mathbf{x}_0 - \mathbf{x}^*\|_2.$$

(e) Now, denote $D = \|\mathbf{x}_0 - \mathbf{x}^*\|_2$, and assume $\rho < 1$. Assume we stop gradient descent after k iterations. **Give a sufficient condition on k to ensure we stopped within ϵ of the optimal $\mathbf{x}_*$. Write your answer in terms of positive quantites.**

Solution: From the previous part, it suffices to take $\rho^k D \leq \epsilon$, i.e.

$$k \geq \frac{\log\frac{D}{\epsilon}}{\log\rho^{-1}}.$$

(f) However, what we often care about is progress in the objective value $f(\mathbf{x})$. That is, we want to show how quickly $f(\mathbf{x}_k)$ is converging to $f(\mathbf{x}^*)$. We can do this by relating $f(\mathbf{x}_k) - f(\mathbf{x}^*)$ to $\|\mathbf{x}_k - \mathbf{x}^*\|_2$, or even better, relating $f(\mathbf{x}_k) - f(\mathbf{x}^*)$ to $\|\mathbf{x}_0 - \mathbf{x}^*\|_2$, for some starting point $\mathbf{x}_0$. First, **show that**

$$f(\mathbf{x}) - f(\mathbf{x}^*) = \frac{1}{2}\|\mathbf{A}(\mathbf{x} - \mathbf{x}^*)\|_2^2.$$

Solution: Note that at $\nabla f(\mathbf{x}^*) = \mathbf{A}^\top(\mathbf{A}\mathbf{x}^* - \mathbf{b}) = \mathbf{0}$ (the gradient evaluated at $\mathbf{x}^*$), implying that $\mathbf{A}^\top\mathbf{A}\mathbf{x}^* = \mathbf{A}^\top\mathbf{b}$. From here we do a bit of algebra to reach our answer, canceling terms and then substituting for $\mathbf{A}^\top\mathbf{b}$ and rearranging:

$$\begin{aligned}
f(\mathbf{x}) - f(\mathbf{x}^*) &= \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 - \frac{1}{2}\|\mathbf{A}\mathbf{x}^* - \mathbf{b}\|_2^2 \\
&= \frac{1}{2}\left((\mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x} - 2\mathbf{b}^\top\mathbf{A}\mathbf{x} + \mathbf{b}^\top\mathbf{b}) - (\mathbf{x}^{*\top}\mathbf{A}^\top\mathbf{A}\mathbf{x}^* - 2\mathbf{b}^\top\mathbf{A}\mathbf{x}^* + \mathbf{b}^\top\mathbf{b})\right) \\
&= \frac{1}{2}\left((\mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x} - 2\mathbf{b}^\top\mathbf{A}\mathbf{x}) - (\mathbf{x}^{*\top}\mathbf{A}^\top\mathbf{A}\mathbf{x}^* - 2\mathbf{b}^\top\mathbf{A}\mathbf{x}^*)\right) \\
&= \frac{1}{2}\left((\mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x} - 2\mathbf{x}^{*\top}\mathbf{A}^\top\mathbf{A}\mathbf{x}) - (\mathbf{x}^{*\top}\mathbf{A}^\top\mathbf{A}\mathbf{x}^* - 2\mathbf{x}^{*\top}\mathbf{A}^\top\mathbf{A}\mathbf{x}^*)\right) \\
&= \frac{1}{2}\left(\mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x} - 2\mathbf{x}^{*\top}\mathbf{A}^\top\mathbf{A}\mathbf{x} + \mathbf{x}^{*\top}\mathbf{A}^\top\mathbf{A}\mathbf{x}^*\right) \\
&= \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{A}\mathbf{x}^*\|_2^2 \\
&= \frac{1}{2}\|\mathbf{A}(\mathbf{x} - \mathbf{x}^*)\|_2^2
\end{aligned}$$

(g) **Now, show that**

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \le \frac{\lambda_1}{2}\|\mathbf{x}_k - \mathbf{x}^*\|_2^2,$$

for $\lambda_1 = \lambda_{\max}(\mathbf{A}^\top\mathbf{A})$, and show that

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \le \frac{\lambda_1}{2}\rho^{2k}\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2.$$

(Hint: Compare with parts (c) and (d)).

Solution: Starting from the previous part, we have

$$\begin{aligned}
f(\mathbf{x}_k) - f(\mathbf{x}^*) &= \frac{1}{2}\|\mathbf{A}(\mathbf{x}_k - \mathbf{x}^*)\|_2^2 \\
&= (\mathbf{x}_k - \mathbf{x}^*)^\top\mathbf{A}^\top\mathbf{A}(\mathbf{x}_k - \mathbf{x}^*) \\
&\le \frac{1}{2}\lambda_{\max}(\mathbf{A}^\top\mathbf{A})\|\mathbf{x}_k - \mathbf{x}^*\|_2^2 \\
&= \frac{\lambda_1}{2}\|\mathbf{x}_k - \mathbf{x}^*\|_2^2
\end{aligned}$$

Combining this with part (d), we get the desired geometric convergence rate for least squares (quadratic functions)

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \le \frac{\lambda_1}{2}\rho^{2k}\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2$$

(h) As before, denote $D = \|\mathbf{x}_0 - \mathbf{x}^*\|_2$, and assume $\rho < 1$. Assume we stop gradient descent after k iterations. **Give a sufficient condition on k to ensure our function value when we stop, $f(\mathbf{x_k})$, is within η of the optimal function value $f(\mathbf{x}_*)$. Write your answer in terms of positive quantites (assume D is large). How does the scaling in η in this bound differ from the scaling in ϵ in the bound derived in part (e)?**

Solution: From the previous part, it suffices to take $\frac{\lambda_1}{2}\rho^{2k}D^2 \le \epsilon$, i.e.

$$k \ge \frac{\log \sqrt{\frac{\lambda_1}{2}}\frac{D}{\sqrt{\epsilon}}}{\log \rho^{-1}}.$$

We see that this scales as $\log \frac{1}{\sqrt{\epsilon}}$ rather than $\log \frac{1}{\epsilon}$.

(i) Finally, the convergence rate is a function of ρ, so it's desirable for ρ to be as small as possible. Recall that ρ is a function of γ, so we want to pick γ such that ρ is as small as possible, as a function of $\lambda_{\min}(\mathbf{A}^\top\mathbf{A}), \lambda_{\max}(\mathbf{A}^\top\mathbf{A})$. **Write the resulting convergence rate ρ as a function of $\kappa = \frac{\lambda_{\max}(\mathbf{A}^\top\mathbf{A})}{\lambda_{\min}(\mathbf{A}^\top\mathbf{A})}$.** This quantity is known as the *condition number* of $\mathbf{A}^\top\mathbf{A}$.

Solution: We would like to solve

$$\min_\gamma \rho(\gamma) = \min_\gamma \max\{|1-\gamma\lambda_{\max}(\mathbf{A}^\top\mathbf{A})|, |1-\gamma\lambda_{\min}(\mathbf{A}^\top\mathbf{A})|\}$$

Recall that $0 < \lambda_{\min}(\mathbf{A}^\top\mathbf{A}) \le \lambda_{\max}(\mathbf{A}^\top\mathbf{A})$ and that $\gamma > 0$. Then, the optimal solution is when the two values are equal in absolute value but opposite in value.

$$-(1-\gamma\lambda_{\max}(\mathbf{A}^\top\mathbf{A})) = 1-\gamma\lambda_{\min}(\mathbf{A}^\top\mathbf{A})$$

This gives $\gamma = \frac{2}{\lambda_{\max}(\mathbf{A}^\top\mathbf{A})+\lambda_{\min}(\mathbf{A}^\top\mathbf{A})}$.

Then, the convergence result from the previous part can be written as

$$\begin{aligned} f(\mathbf{x}_k) - f(\mathbf{x}^*) &= \frac{\lambda_1}{2}\left(\frac{\lambda_{\max}(\mathbf{A}^\top\mathbf{A}) - \lambda_{\min}(\mathbf{A}^\top\mathbf{A})}{\lambda_{\max}(\mathbf{A}^\top\mathbf{A}) + \lambda_{\min}(\mathbf{A}^\top\mathbf{A})}\right)^{2k}\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2 \\ &= \frac{\lambda_1}{2}\left(\frac{\kappa-1}{\kappa+1}\right)^{2k}\|\mathbf{x}_0 - \mathbf{x}^*\|_2^2 \end{aligned}$$

4 A Simple Classification Approach

Make sure to submit the code you write in this problem to "HW5 Code" on Gradescope.

Classification is an important problem in applied machine learning and is used in many different applications like image classification, object detection, speech recognition, machine translation and others.

In *classification*, we assign each datapoint a class from a finite set (for example the image of a digit could be assigned the value $0, 1, \ldots, 9$ of that digit). This is different from *regression*, where each datapoint is assigned a value from a continuous domain like $\mathbb{R}$ (for example features of a house like location, number of bedrooms, age of the house, etc. could be assigned the price of the house).

In this problem we consider the simplified setting of classification where we want to classify data points from $\mathbb{R}^d$ into *two* classes. For a linear classifier, the space $\mathbb{R}^d$ is split into two parts by a hyperplane: All points on one side of the hyperplane are classified as one class and all points on the other side of the hyperplane are classified as the other class.

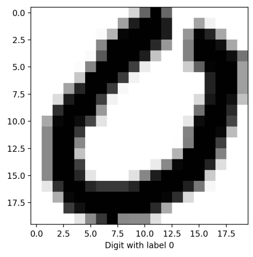
The goal of this problem is to show that even a regression technique like linear regression can be used to solve a classification problem. This can be achieved by regressing the data points in the training set against -1 or 1 depending on their class and then using the level set of 0 of the regression function as the classification hyperplane (i.e. we use 0 as a threshold on the output to decide between the classes).

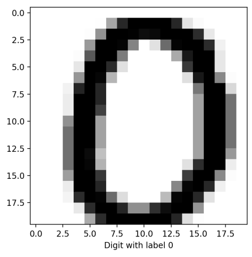
Later in lecture we will learn why linear regression is not the optimal approach for classification and we will study better approaches like logistic regression, SVMs and neural networks.

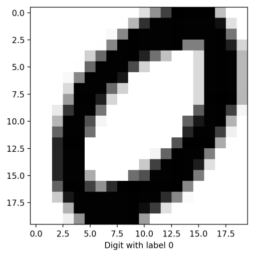
(a) The dataset used in this exercise is a subset of the MNIST dataset. The MNIST dataset assigns each image of a handwritten digit their value from 0 to 9 as a class. For this problem we only keep digits that are assigned a 0 or 1, so we simplify the problem to a two-class classification problem.

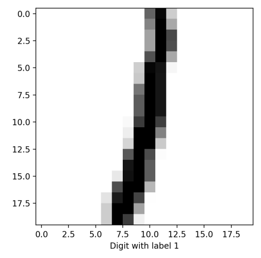
Download and visualize the dataset (example code included). Include three images that are labeled as 0 and three images that are labeled as 1 in your submission.

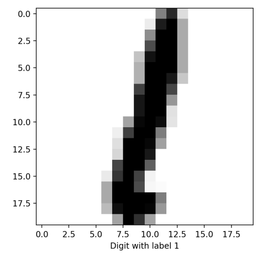
Solution: We have digits labeled as zero at indices 2, 3, 5 in the dataset. We have digits labeled as one at indices 0, 1, 4 in the dataset.

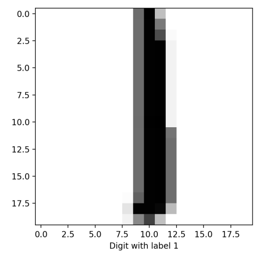












This can be visualized with the following script:

```python
import numpy as np
import matplotlib.pyplot as plt

# Load the training dataset
train_features = np.load("train_features.npy")
train_labels = np.load("train_labels.npy").astype("int8")

n_train = train_labels.shape[0]

def visualize_digit(features, label):
    # Digits are stored as a vector of 400 pixel values. Here we
    # reshape it to a 20x20 image so we can display it.
    plt.imshow(features.reshape(20, 20), cmap="binary")
    plt.xlabel("Digit with label " + str(label))
    plt.show()

# zeros
visualize_digit(train_features[2,:], train_labels[2])
visualize_digit(train_features[3,:], train_labels[3])
visualize_digit(train_features[5,:], train_labels[5])

# ones
visualize_digit(train_features[0,:], train_labels[0])
visualize_digit(train_features[1,:], train_labels[1])
visualize_digit(train_features[4,:], train_labels[4])
```

(b) We now want to use linear regression for the problem, treating class labels as real values $y = -1$ for class "zero" and $y = 1$ for class "one". In the dataset we provide, the images have already been flattened into one dimensional vectors (by concatenating all pixel values of the two dimensional image into a vector) and stacked as rows into a feature matrix $\mathbf{X}$. We want to set up the regression problem $\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$ where the entry y_i is the value of the class (-1 or 1) corresponding to the image in row $\mathbf{x}_i^\top$ of the feature matrix. **Solve this regression problem for the training set and report the value of $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$ as well as the weights $\mathbf{w}$.** For this problem you may only use pure Python and numpy (no machine learning libraries!).

Solution: With the same loading code as above:

```python
# Linear regression

X = train_features
y = 2*train_labels.astype("int8") - 1
w = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, y))
residuals = np.dot(X, w) - y
epsilon = np.linalg.norm(residuals)**2

print("regression residual squared error is {}".format(epsilon))

print("weights are {}".format(w))
```

The outputs are:

```
regression residual squared error is 422.750833193
weights are [ -3.30879480e-01   3.91613483e-01   1.48013666e-01  -1.60475999e-01
   1.03201739e-01  -1.96623709e-02  -1.27722740e-01   9.46968887e-03
  -1.71516072e-02  -5.67266904e-03  -4.68674535e-03  -1.12593472e-02
  -5.70893241e-03   4.59235208e-03   1.78789273e-02  -3.00950985e-02
   1.00359349e-02  -6.45869076e-02  -2.30248440e-02  -2.63121855e-02
  -1.63464829e-01   4.66997147e-01  -2.83771055e-03  -1.05607457e-01
  -1.80681601e-01   1.34385273e-01  -5.08473255e-03  -3.07385344e-02
  -6.59416839e-02   1.75730512e-02  -3.06562148e-02  -6.23832922e-03
   1.54059939e-02  -3.13759260e-02  -2.54694535e-03  -6.17963215e-03
  -1.02767663e-03   5.19377999e-02   3.95462736e-02   6.29393160e-02
...
   3.72376144e-02  -2.26117996e-03   5.63378446e-02   9.63310339e-03
   1.01846397e-01   8.83652642e-02  -4.26671766e-02   2.76091605e-01
   5.16173951e-02   5.59021682e-02   1.50067955e-02  -7.88934063e-03
  -1.78755876e-02   5.81122423e-03  -1.39549663e-02  -2.94532962e-02
  -7.18745068e-02  -6.40835539e-02   2.70997570e-03  -2.76509989e-02
  -3.69484797e-02   7.89522007e-03   7.59321675e-02  -8.43895786e-03
   1.20719289e-02   1.71253383e-01  -3.02490622e-01   2.85772234e-03
  -2.39322335e-03  -3.16695720e-02   3.74489347e-03  -3.21877114e-02
   1.20041789e-02  -4.41990234e-03   1.04306452e-02   3.75087769e-03
   1.62195284e-02  -3.37275560e-03  -4.06924039e-02  -1.61125399e-02
  -6.15769811e-03  -3.09251025e-02  -5.01930043e-02   1.58581156e-02
  -1.27891421e-01   1.68097302e-01  -2.77848482e-01   4.18617725e-02]
```

(c) Given a new flattened image $\mathbf{x}$, one natural rule to classify it is the following one: It is a zero if $\mathbf{x}^\top \mathbf{w} \leq 0$ and a one if $\mathbf{x}^\top \mathbf{w} > 0$. **Report what percentage of the digits in the training set are correctly classified by this rule. Report what percentage of the digits in the test set are correctly classified by this rule.**

Solution: Continuing the code from above:

```python
def classify(w, features):
    return 1*(np.dot(features, w) >= 0.0)

train_error = 1.0 * sum(classify(w, train_features) != train_labels) / n_train

print("classification error on the train dataset is {}".format(train_error))

# Load the test dataset
# It is good practice to do this after the training has been
# completed to make sure that no training happens on the test
# set!
test_features = np.load("test_features.npy")
test_labels = np.load("test_labels.npy").astype("int8")

n_test = test_labels.shape[0]

test_error = 1.0 * sum(classify(w, test_features) != test_labels) / n_test

print("classification error on the test dataset is {}".format(test_error))
```

The output is:

```
classification error on the train dataset is 0.00240901660501
classification error on the test dataset is 0.00189125295508
```

So 99.76 percent of digits on the train set are correctly classified and 99.81 percent of digits on the test set are correctly classified.

(d) **Why is the performance typically evaluated on a separate test set (instead of the training set) and why is the performance on the training and test set similar in our case?** We will cover these questions in more detail later in the class.

Solution: If the model that is fit on the training set is very flexible (for example a linear function with very high dimension or a so called neural network with many parameters), fitting the model often reduces the training error close to zero. This is called overfitting and can for example happen if the dimension of the model is similar to the number of observation. In this case, the classification on the training set is perfect and not indicative of the classification performance on inputs that are not part of the training set. Therefore, evaluating the performance on the (unseen) test set is a better indication of performance on future data.

In the setting of this problem, we are pretty safe against overfitting, since the dimension of the model $d = 400$ is much smaller than the number of observations $n = 11623$.

(e) Unsatisfied with the current performance of your classifier, you call your mother for advice, and she suggests to use random features instead of raw pixel features. Specifically, she suggests to use the feature map

$$\phi(\mathbf{x}) = \cos\Big(\mathbf{G}^\top \mathbf{x} + \mathbf{b}\Big),$$

where each entry of $G \in \mathbb{R}^{d \times p}$ is drawn i.i.d. as $\mathcal{N}(0, 0.01)$ and each entry in the vector $b \in \mathbb{R}^p$ is drawn i.i.d from the uniform distribution on $[0, 2\pi]$. Note that $\cos$ should be taken point wise, i.e. $\phi(x)$ should output a vector in $\mathbb{R}^p$. **With $p = 5000$, report what percentage of digits are correctly classified using this approach on the training set and test set. Make sure to adapt the classification rule, i.e. the threshold set for the outputs.**

Solution: TODO