CS 189 Introduction to Machine Learning
Spring 2018

HW1

Your self-grade URL is http://eecs189.org/self_grade?question_ids=1_1,1_2,2_1,2_2,2_3,2_4,2_5,2_6,3_1,3_2,3_3,3_4,4_1,4_2,4_3,4_4,4_5,4_6,5_1,5_2,5_3,5_4,5_5,6.

This homework is due **Monday, June 25 at 10 p.m.**

2 The accuracy of learning decision boundaries

This problem exercises your basic probability (e.g. from 70) in the context of understanding why lots of training data helps to improve the accuracy of learning things.

For each $\theta \in (1/3, 2/3)$, define $f_\theta : [0,1] \to \{0,1\}$, such that

$$f_\theta(x) = \begin{cases} 1 \text{ if } x > \theta \\ 0 \text{ otherwise.} \end{cases}$$

The function is plotted in Figure 1.

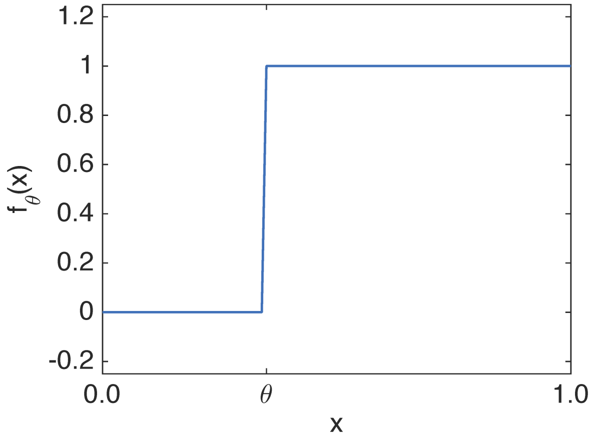


Figure 1: Plot of function $f_\theta(x)$ against x.

We draw samples $X_1, X_2, \ldots, X_n$ uniformly at random and i.i.d. from the interval $[0,1]$. Our goal is to learn an estimate for θ from n random samples $(X_1, f_\theta(X_1)), (X_2, f_\theta(X_2)), \ldots, (X_n, f_\theta(X_n))$.

Let $T_{min} = \max(\{\frac{1}{3}\} \cup \{X_i | f_\theta(X_i) = 0\})$. We know that the true θ must be larger than T_{min}.

Let $T_{max} = \min(\{\frac{2}{3}\} \cup \{X_i | f_\theta(X_i) = 1\})$. We know that the true θ must be smaller than T_{max}.

The gap between T_{min} and T_{max} represents the uncertainty we will have about the true θ given the training data that we have received.

(a) **What is the probability that $T_{max} - \theta > \epsilon$ as a function of ϵ? And what is the probability that $\theta - T_{min} > \epsilon$ as a function of ϵ?**

Solution: First note that when $\theta + \epsilon > \frac{2}{3}$ we have that $\Pr(T_{max} > \theta + \epsilon) \leq \Pr\big(T_{max} > \frac{2}{3}\big) = 0$ by definition of T_{max}.

For the case when $\theta + \epsilon < \frac{2}{3}$, the task is to find the probability of the event of the random variable T_{max} defined by $\mathcal{E} := \{T_{max} > \theta + \epsilon\}$. Note that because $\Pr\big(\{T_{max} < \theta\}\big) = 0$ or equivalently $\{T_{max} < \theta\} = \emptyset$, the probability $\Pr(\mathcal{E})$ can alternatively be expressed by the probability of a different event $\mathcal{E}_0 = \{\theta \leq T_{max} \leq \theta + \epsilon\}$ in terms of

$$\begin{aligned}\Pr(\mathcal{E}) &= \Pr\big(\{T_{max} > \theta + \epsilon\} \cup \{T_{max} < \theta\}\big) \\ &= 1 - \Pr\big(\{\theta \leq T_{max} \leq \theta + \epsilon\}\big) = 1 - \Pr(\mathcal{E}_0).\end{aligned}$$

Now consider the event $\mathcal{E}_1 := \{\text{at least one } X_i \text{ lies in } [\theta, \theta + \epsilon]\}$. You can now show that $\mathcal{E}_0 = \mathcal{E}_1$, i.e.

$$\{\theta \leq T_{max} \leq \theta + \epsilon\} = \{\text{at least one } X_i \text{ lies in } [\theta, \theta + \epsilon]\}.$$

by definition of T_{max}.

Going back to the original event $\mathcal{E}$ we thus find

$$\Pr(\mathcal{E}) = 1 - \Pr(\mathcal{E}_0) = 1 - \Pr(\mathcal{E}_1) = \Pr(\mathcal{E}_1^c)$$

where the complement $\mathcal{E}_1^c = \{\text{no } X_i \text{ lies in } [\theta, \theta + \epsilon]\} = \bigcap_{i=1}^n \{X_i \notin [\theta, \theta + \epsilon]\}$.

Restating the probability of $\mathcal{E}$ in terms of an intersection of events on X_i now allows us to easily find $\Pr(\mathcal{E})$ because of independence and uniform distribution of X_i, which reads

$$\Pr\left(\bigcap_{i=1}^n \{X_i \notin [\theta, \theta + \epsilon]\}\right) = \prod_{i=1}^n \Pr\big(\{X_i \notin [\theta, \theta + \epsilon]\}\big) = (1 - \epsilon)^n.$$

In summary, we obtain

$$\Pr(T_{max} - \theta > \epsilon) = \begin{cases} (1 - \epsilon)^n & \theta + \epsilon < \frac{2}{3} \\ 0 & \text{o.w.} \end{cases}$$

Similar analysis applies to the second part, except our lower bound is $\frac{1}{3}$:

$$\Pr(\theta - T_{min} > \epsilon) = \begin{cases} (1 - \epsilon)^n & \theta - \epsilon > \frac{1}{3} \\ 0 & \text{o.w.} \end{cases}$$

(b) Suppose that you would like the estimator $\hat{\theta} = (T_{max} + T_{min})/2$ for θ that is ϵ-close (defined as $|\hat{\theta} - \theta| < \epsilon$, where $\hat{\theta}$ is the estimation and θ is the true value) with probability at least $1 - \delta$. Both ϵ and δ are some small positive numbers. **Please bound or estimate how big of an n do you need?** You do not need to find the optimal lowest sample complexity n, an approximation using results of question (a) is fine.

Solution: One way to obtain $\hat{\theta}$ within a window of size 2ϵ is to have both T_{max} and T_{min} be within ϵ of θ. To see this, define random variables $L = \theta - T_{min}, U = T_{max} - \theta$. When $L < \epsilon$ and $U < \epsilon$, we have $\theta - T_{min} < \epsilon$ and $0 < T_{max} - \theta$. Adding the two inequalities, we have $\theta - T_{min} < T_{max} - \theta + \epsilon$, thus $\hat{\theta} - \theta > -\epsilon/2 > -\epsilon$. Similarly, with those conditions, we have $\hat{\theta} - \theta < \epsilon$. Thus $L < \epsilon$ and $U < \epsilon$ is a sufficient condition for $\hat{\theta}$ to be ϵ-close, that is

$$\Pr\Big(|\hat{\theta} - \theta| < \epsilon\Big) \geq \Pr\big(\{L < \epsilon\} \cap \{U < \epsilon\}\big).$$

Instead of lower bounding $\Pr\big(\{L < \epsilon\} \cap \{U < \epsilon\}\big)$ we upper bound the probability of the complement of the event, which reads $\{L > \epsilon\} \cup \{U > \epsilon\}$, via union bound as follows:

$$\Pr\big(\{L > \epsilon\} \cup \{U > \epsilon\}\big) \leq \Pr\big(\{L > \epsilon\}\big) + \Pr\big(\{U > \epsilon\}\big) \leq 2(1-\epsilon)^n$$

using the result in problem (a).

We must ensure that this probability is upper bounded by δ, which ensures that we succeed with probability at least $1 - \delta$. Solving for n, we have

$$\begin{aligned} 2(1-\epsilon)^n &< \delta \\ n &> \frac{\ln\big(\frac{2}{\delta}\big)}{\ln\big(1/(1-\epsilon)\big)}. \end{aligned}$$

Again, using the approximation $\ln(1-x) \sim -x$, we have $n > \frac{1}{\epsilon}\ln\big(2/\delta\big)$ for ϵ small.

(c) Let us say that instead of getting random samples $(X_i, f(X_i))$, we were allowed to choose where to sample the function, but you had to choose all the places you were going to sample in advance. **Propose a method to estimate θ. How many samples suffice to achieve an estimate that is ϵ-close as above?** (**Hint:** You need not use a randomized strategy.)

Solution: Pick n points uniformly spaced on the interval $(\frac{1}{3}, \frac{2}{3})$. Then, the ith sample $X_i = \frac{1}{3} + \frac{i}{3n}$. Since we have n points, we create intervals of length $\frac{1}{3n}$. If our intervals are smaller than 2ϵ, we can guarantee that we estimate θ within an interval of 2ϵ. Solving for n, we have $\frac{1}{3n} < 2\epsilon$ and so $n > \frac{1}{6\epsilon}$ samples are sufficient.

Note that using our calculations the sample complexity for this deterministic method is *always lower* than the sample complexity of the probabilistic method in problem (b) $\delta < 1$ since $\ln\left(\frac{2}{\delta}\right) > \frac{1}{6}$ for any $\delta < 1$. Therefore, uniform sampling and allowing for some non-zero probability that we do not obtain an ϵ-close estimator, does not require fewer samples than a deterministic method which always ensures an ϵ-close estimator. In many other settings however, allowing some uncertainty (of finding a good estimator) can help to reduce the sample complexity significantly.

(d) Suppose that you could pick where to sample the function adaptively — choosing where to sample the function in response to what the answers were previously. **Propose a method to estimate θ. How many samples suffice to achieve an estimate that is ϵ-close as above?**

Solution: Use binary search: start with three pointers, $s = 1/3, e = 2/3$ with m as the midpoint. If $f(m) = 0$, set $s = m$ and recompute the midpoint (i.e., search over the second half of the range). Otherwise, $f(m) = 1$ and set $e = m$ (i.e., search over the first half of the range). For each point sampled, we reduce the size of the range by half, so after n points, the interval we consider is $\frac{1}{3 \cdot 2^n}$. We want this to be less than 2ϵ, and so

$$\frac{1}{3 \cdot 2^n} < 2\epsilon \implies n > \log_2(\frac{1}{3\epsilon}) - 1.$$

(e) In the three sampling approaches above: random, deterministic, and adaptive, **compare the scaling of n with ϵ (and δ as well for the random case).**

Solution:

(a) For random, n is logarithmic in $1/\delta$. For ϵ, we use the approximation $\ln\bigl(1/(1-\epsilon)\bigr) \sim \epsilon$ to conclude that n is inversely related to ϵ.

(b) For deterministic, n is inversely related to ϵ. Note that this is the same scaling as choosing random evaluation points.

(c) For adaptive, n is logarithmic in $\frac{1}{\epsilon}$.

(f) **Why do you think we asked this series of questions? What are the implications of those results in a machine learning application?**

Solution: We ask this question because we want to show how the number of training examples affects the accuracy. Intuitively, more data lead to a more accurate estimator. We quantify this intuition with a simple but concrete example.

The three sampling approaches are some common ways to get the training data. When most of the real world datasets are collected, one doesn't have any control on X_i. That is the random sampling paradigm. The deterministic sampling paradigm refers to the scenario when one could carefully design a set of X_i. One might think that the sample complexity of the deterministic case should be much better than that of the random one, however for this particular model, they are not quite different. There is only a factor of $\log \frac{1}{\delta}$ off. However, when we move to the adaptive paradigm, the sample complexity is exponentially smaller.

For practical machine learning applications, the implication is that you want to have as much control on the samples as you can (such as adaptive sampling) to learn a better model with the same amount of data.

3 Eigenvalue and Eigenvector review

A square matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$ has a (right) eigenvalue $\lambda \in \mathbb{C}$ and (right) eigenvector $\mathbf{x} \in \mathbb{C}^d \setminus 0$ if $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$. Left eigenvalues and eigenvectors are defined analogously — $\mathbf{x}^T\mathbf{A} = \lambda\mathbf{x}^T$. Since the definition is scale invariant (if $\mathbf{x}$ is an eigenvector, then $t\mathbf{x}$ is an eigenvector for any $t \neq 0$), we adopt the convention that each eigenvector has norm 1.

(a) **Compute the right and left eigenvalues and eigenvectors** of the following matrices. You may use a computer for questions v) and vi).

i) $\mathbf{A} = \begin{bmatrix} 2 & -4 \\ -1 & -1 \end{bmatrix}$

Solution:

Left eigenvectors are given by row vectors. We provide their column versions in the following solutions.

Right: $\lambda_1 = -2, \lambda_2 = 3, \mathbf{v}_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 4/\sqrt{17} \\ -1/\sqrt{17} \end{bmatrix}$

Left: $\lambda_1 = -2, \lambda_2 = 3, \mathbf{v}_1 = \begin{bmatrix} 1/\sqrt{17} \\ 4/\sqrt{17} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$

ii) $\mathbf{B} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$

Solution: Since this matrix is symmetric, the right and left eigenvectors are the same.

Right: $\lambda_1 = 2, \lambda_2 = 4, \mathbf{v}_1 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

Left: $\lambda_1 = 4, \lambda_2 = 10, \mathbf{v}_1 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

iii) $\mathbf{A}^2$

Solution: We have $\mathbf{A}^2 = \mathbf{A}\mathbf{A} = \begin{bmatrix} 8 & -4 \\ -1 & 5 \end{bmatrix}$.

Right: $\lambda_1 = 4, \lambda_2 = 9, \mathbf{v}_1 = \begin{bmatrix} \sqrt{1/2} \\ \sqrt{1/2} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 4/\sqrt{17} \\ -1/\sqrt{17} \end{bmatrix}$

Left: $\lambda_1 = 4, \lambda_2 = 9, \mathbf{v}_1 = \begin{bmatrix} 1/\sqrt{17} \\ 4/\sqrt{17} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} \sqrt{1/2} \\ -\sqrt{1/2} \end{bmatrix}$

iv) $\mathbf{B}^2$

Solution: B is a symmetric matrix, so its eigenvalues are given by the square of the eigenvalues of B, with the eigenvectors staying the same. Hence, we have:

Right: $\lambda_1 = 4, \lambda_2 = 16, \mathbf{v}_1 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

Left: $\lambda_1 = 4, \lambda_2 = 16, \mathbf{v}_1 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

v) $\mathbf{AB}$

Solution: We have $\mathbf{AB} = \begin{bmatrix} 2 & -10 \\ -4 & -4 \end{bmatrix}$.

Right: $\lambda_1 = 6, \lambda_2 = -8, \mathbf{v}_1 = \begin{bmatrix} 5/\sqrt{29} \\ -2/\sqrt{29} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

Left: $\lambda_1 = 6, \lambda_2 = -8, \mathbf{v}_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 2/\sqrt{29} \\ 5/\sqrt{29} \end{bmatrix}$

vi) $\mathbf{BA}$

Solution: We have $\mathbf{BA} = \begin{bmatrix} 5 & -13 \\ -1 & -7 \end{bmatrix}$.

Right: $\lambda_1 = 6, \lambda_2 = -8, \mathbf{v}_1 = \begin{bmatrix} 13/\sqrt{170} \\ -1/\sqrt{170} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$

Left: $\lambda_1 = 6, \lambda_2 = -8, \mathbf{v}_1 = \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \mathbf{v}_2 = \begin{bmatrix} 1/\sqrt{170} \\ 13/\sqrt{170} \end{bmatrix}$

(b) **Compute the singular value decompositions** of the matrices above. In addition, please compute the SVD of: $\mathbf{C} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \\ 2 & -4 \\ -1 & -1 \end{bmatrix}$. For $\mathbf{B}$ and $\mathbf{B}^2$ compute by hand. For the rest of the matrices you may use a computer.

Solution: We begin with the reduced SVD of $\mathbf{C}$:

$$\mathbf{C} = \begin{bmatrix} -0.14 & 0.79 \\ -0.56 & 0.32 \\ 0.80 & 0.43 \\ 0.18 & -0.28 \end{bmatrix} \begin{bmatrix} 5.20 & 0 \\ 0 & 3.86 \end{bmatrix} \begin{bmatrix} 0.08 & 0.99 \\ -0.99 & 0.08 \end{bmatrix}^\top$$

(a) $\mathbf{A} = \begin{bmatrix} 0.99 & 0.11 \\ 0.11 & -0.99 \end{bmatrix} \begin{bmatrix} 4.50 & 0 \\ 0 & 1.33 \end{bmatrix} \begin{bmatrix} 0.42 & 0.91 \\ -0.91 & 0.42 \end{bmatrix}^\top$

(b) Since $\mathbf{B}$ is symmetric and has unique and positive eigenvalues. $\mathbf{U} = \mathbf{V}$ are the stacked columns of the eigenvectors and Σ is diagonal matrix of the corresponding eigenvalues of $\mathbf{B}$.

$\mathbf{B} = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^\top$

(c) $\mathbf{A}^2 = \begin{bmatrix} -0.93 & 0.39 \\ 0.39 & 0.92 \end{bmatrix} \begin{bmatrix} 9.59 & 0 \\ 0 & 3.75 \end{bmatrix} \begin{bmatrix} -0.81 & 0.59 \\ 0.59 & 0.81 \end{bmatrix}^\top$

(d) Same reasoning as for $\mathbf{B}$ we get: $\mathbf{B}^2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 16 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^\top$

(e) $\mathbf{AB} = \begin{bmatrix} 0.93 & 0.35 \\ 0.35 & -0.93 \end{bmatrix} \begin{bmatrix} 10.78 & 0 \\ 0 & 4.45 \end{bmatrix} \begin{bmatrix} 0.04 & 0.99 \\ -0.99 & 0.04 \end{bmatrix}^\top$

(f) $\mathbf{BA} = \begin{bmatrix} 0.91 & 0.42 \\ 0.42 & -0.91 \end{bmatrix} \begin{bmatrix} 15.3 & 0 \\ 0 & 3.14 \end{bmatrix} \begin{bmatrix} 0.27 & 0.96 \\ -0.96 & 0.27 \end{bmatrix}^\top$

(c) **Show** from the definition of a right eigenvalue that the quantity λ is an eigenvalue with associated eigenvector $\mathbf{x}$ iff for all $1 \leq i \leq d$, we have

$$(\lambda - A_{ii})x_i = \sum_{j \neq i} A_{ij}x_j.$$

Solution: Distribute, and we have $\lambda x_i - A_{ii}x_i = \sum_{j \neq i} A_{ij}x_j$. So,

$$\lambda x_i = \sum_j A_{ij}x_j = A_i x_i$$

If this is true for all i, then $\lambda \mathbf{x} = \mathbf{Ax}$ and λ is therefore an eigenvalue.

Next we'll show the other direction. λ is eigenvalue with associated eigenvector $\mathbf{x}$, $\mathbf{Ax} = \lambda \mathbf{x}$, hence, $\sum_j A_{ij}x_j = \lambda x_i$ for all $1 \leq i \leq n$. $\sum_j A_{ij}x_j = \sum_{j \neq i} A_{ij}x_j + A_{ii}x_i = \lambda x_i$. And we get exactly that, $\sum_{j \neq i} A_{ij}x_j = (\lambda - A_{ii})x_i$.

(d) Now for an arbitrary eigenvalue λ of $\mathbf{A}$ and its associated eigenvector $\mathbf{x}$, choose index i such that $|x_i| \geq |x_j|$ for all $j \neq i$. For such an index i, **show** that

$$|\lambda - A_{ii}| \leq \sum_{j \neq i} |A_{ij}|.$$

You have just proved Gershgorin's circle theorem, which states that all the eigenvalues of a $d \times d$ matrix lie within the union of d disks in the complex plane, where disk i has center A_{ii}, and radius $\sum_{j \neq i} |A_{ij}|$.

Solution: From the previous part, we see that

$$|\lambda - A_{ii}| = \frac{1}{|x_i|} |\sum_{j \neq i} A_{ij}x_j| \leq \sum_{j \neq i} |A_{ij}| \frac{|x_j|}{|x_i|} \leq \sum_{j \neq i} |A_{ij}|. \tag{1}$$

The last inequality holds because by assumption we have $\frac{|x_j|}{|x_i|} \leq 1$ for all $j \neq i$.

4 Fun with least squares

In ordinary least squares we learn to predict a *target* scalar $y \in \Re$ given a *feature* vector $\mathbf{x} \in \mathbb{R}^d$. Each element of $\mathbf{x}$ is called a feature, which could correspond to a scientific *measurement*. For example, the i-th element of $\mathbf{x}$, denoted by $(\mathbf{x})_i$, could correspond to the velocity of a car at time i. y could represent the final location (say just in one direction) of the car.

For the purpose of predicting y from $\mathbf{x}$ we are given n samples $(\mathbf{x}_i, y_i)$ with $i = 1, \ldots, n$ (where feature vectors and target scalars are observed in pairs), which we also call the training set. In this problem we want to predict the unobserved target y corresponding to a new $\mathbf{x}$ (not in the training set) by some linear prediction $\hat{y} = \mathbf{x}^\top \widehat{\mathbf{w}}$ where the *weight* $\widehat{\mathbf{w}} \in \Re^d$ minimizes the least-squares training cost

$$\sum_{i=1}^{n} (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

where in the matrix $\mathbf{X} \in \Re^{n \times d}$, the transposed sample feature vectors $\mathbf{x}_i^\top$ constitute the d-dimensional row vectors, and the n-dimensional vectors of training measurements $\mathbf{x}^j = ((\mathbf{x_1})_j, \ldots, (\mathbf{x_n})_j)^\top$ for $j = 1, \ldots, d$ correspond to the column vectors (see Figure 2). and $\mathbf{y} = (y_1, \ldots, y_n)^\top$.

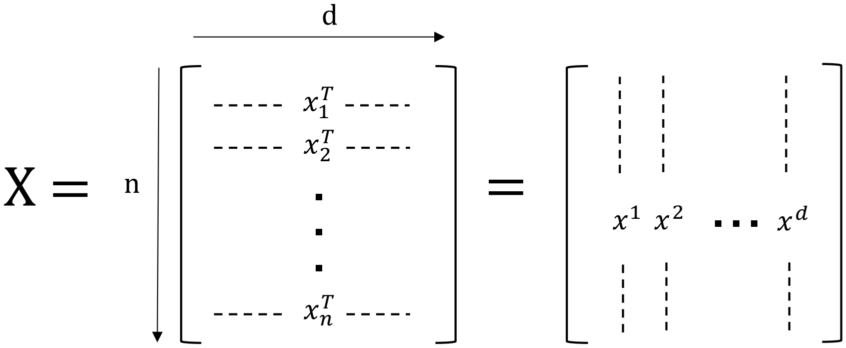


Figure 2: Dimensions, column and row vector notation for matrix $\mathbf{X}$

Let us actually build on the example mentioned above and view the measurements $(\mathbf{x_i})_j$ of each sample $\mathbf{x}_i$ as a sequence of measurements, e.g. velocities of car i, over time $j = 1, \ldots, d$.

(a) Is this problem in a supervised or unsupervised learning setting? **Please explain.**

Solution: Machine learning in general is fundamentally using collected training data to learn something useful about the data. In this problem, the goal is to predict $y \in \Re$ by *learning* a weight $\widehat{\mathbf{w}}$ from our observed training samples. This weight is then used to yield a prediction $\hat{y} = \mathbf{x}^\top \widehat{\mathbf{w}}$ for the target variable y.

Because our training samples includes observed target variables y, we are working in a supervised learning setting. In our concrete examples with velocities of cars, we can imagine collecting velocity and final location data from n different cars which we accumulate in the matrix $\mathbf{X}$ and vector $\mathbf{y}$ respectively.

(b) Suppose that we want to learn (from our training set) to predict the final location y from only the first t measurements. Denoting the prediction of y from the first t measurements by $\hat{y}^t$, we thus want to use $(\mathbf{x})_j, j = 1, \ldots, t$ to predict y. If we now learn how to obtain $\hat{y}^t$ for each $t = 1, \ldots, d$, we end up with a sequence of estimators $\hat{y}^1, \ldots, \hat{y}^d$ for each car.

Provide a method to obtain $\hat{y}^t$ for each t. Note that we will obtain a different model for each t.

Solution:

For the clarification of the problem: Note that there are no "true intermediate locations" y^t. We only have predictors $\hat{y}^t$ which give us the best prediction we can have of the *final* location y at time t.

Let us define the matrix $\mathbf{X}^t$ as the submatrix consisting of the first t columns of $\mathbf{X}$. In order to solve for the t-th linear predictor, we solve the least squares problem $\widehat{\mathbf{w}}^t = \arg\min_{\mathbf{w}\in\Re^t} \|\mathbf{X}^t\mathbf{w} - \mathbf{y}\|_2^2$. The best linear prediction for y given $\mathbf{X}^t$ is thus given by $\hat{y}^t = \mathbf{X}^t\widehat{\mathbf{w}}^t$.

(c) Someone suggests that maybe the measurements themselves are partially predictable from the previous measurements, which suggests employing a two stage strategy to solve the original prediction problem: First we predict the t-th measurement $(\mathbf{x})_t$ based on the previous measurements $(\mathbf{x})_1, \ldots, (\mathbf{x})_{t-1}$. Then we look at the differences (sometimes deemed the "innovation") between the actual t-th measurement we obtained and our prediction for it, i.e. $(\Delta\mathbf{x})_t := (\mathbf{x})_t - (\widehat{\mathbf{x}})_t$ for $t > 1$ and assume $(\Delta\mathbf{x})_1 = (\mathbf{x})_1$. Finally, we use $(\Delta\mathbf{x})_1, \ldots, (\Delta\mathbf{x})_t$ to obtain a prediction $\tilde{y}^t$.

In order to learn the maps which allow us to (1) take $(\mathbf{x})_1, \ldots, (\mathbf{x})_{t-1}$ to obtain $(\Delta\mathbf{x})_1, \ldots, (\Delta\mathbf{x})_t$ and (2) take $(\Delta\mathbf{x})_1, \ldots, (\Delta\mathbf{x})_t$ to predict $\tilde{y}^t$, we again use our training set. Specifically for each t, in stage (1), we fit the vectors of training measurements $\mathbf{x}^1, \ldots, \mathbf{x}^{t-1}$ linearly to $\mathbf{x}^t$ using least squares for each t. In stage (2), we use the innovation vectors $(\Delta\mathbf{x}^1, \ldots, \Delta\mathbf{x}^t)$ to predict $\mathbf{y}^t$ again using least squares. Let's define the matrix $\tilde{\mathbf{X}}^t := (\Delta\mathbf{x}^1, \ldots, \Delta\mathbf{x}^t)$ and $\tilde{\mathbf{X}} = \tilde{\mathbf{X}}^d$.

Show how we can learn the best linear predictions $\widehat{\mathbf{x}}^t$ from $\mathbf{x}^1, \ldots, \mathbf{x}^{t-1}$. Then **provide an expression** for $\tilde{\mathbf{y}}^t$ depending on the innovations $\Delta\mathbf{x}^1, \ldots, \Delta\mathbf{x}^t$.

When presented with a new feature vector $\mathbf{x}$, are the sequence of final predictions of the one-stage training $\hat{y}^t$ in (b) and two-stage training $\tilde{y}^t$ in (c) the same? **Explain your reasoning.**

Solution: (1) For each vector of training measurements $\mathbf{x}^t$, there is a component of that measurement in the subspace spanned by the previous measurements $\mathbf{x}^1, \ldots, \mathbf{x}^{t-1}$, and another component that is orthogonal to this subspace. From the geometric intuition of least squares, we can find the projection of $\mathbf{x}^t$ onto the subspace formed by $\mathbf{x}^1, \ldots, \mathbf{x}^{t-1}$ by solving $\tilde{\mathbf{v}}^t = \arg\min_{\mathbf{v}\in\Re^{t-1}} \|\mathbf{x}^t - \mathbf{X}^{t-1}\mathbf{v}\|_2^2$. This gives us the best linear predictor $\widehat{\mathbf{x}^t} = \mathbf{X}^{t-1}\tilde{\mathbf{v}}^t$ of $\mathbf{x}^t$.

Analogous to problem (b), we solve the least squares problem $\min_{\mathbf{w}\in\Re^t} \|\tilde{\mathbf{X}}^t\mathbf{w} - \mathbf{y}\|_2^2$ to obtain $\tilde{\mathbf{y}}^t$. Note that our predictions $\tilde{\mathbf{y}}^t$ are exactly the same as $\widehat{\mathbf{y}}^t$ derived in part (b). This equality is due to the fact that the column space of the matrices $\mathbf{X}^t$ and $\tilde{\mathbf{X}}^t$ are the same for each t. However, the regression coefficients estimated from measurements (part (b)) are different from those estimated from the innovations (part (c)).

(d) **Which well-known procedure do the steps to obtain $\tilde{\mathbf{X}}$ from $\mathbf{X}$ remind you of?** (**HINT:** Think about how the column vectors in $\tilde{\mathbf{X}}$ are geometrically related.)

Is there an efficient way to update the weight vector $\widehat{\mathbf{w}}^t$ from $\widehat{\mathbf{w}}^{t-1}$ when computing the sequence of predictions $\tilde{y}^t$? Here, $\widehat{\mathbf{w}}^t$ are the weights for the second stage in (c).

Solution: Note that by the definition of the projection operations noted above, the resulting columns of the matrix $\tilde{\mathbf{X}}$ are orthogonal. This is a form of feature preprocessing, that allows us to make the features in our problem orthogonal. With an additional normalization step, this is equivalent to performing Gram-Schmidt orthogonalization of the matrix $\mathbf{X}$.

As explained in the previous parts, our predictions $\widehat{\mathbf{y}}^t$ in (b) and $\tilde{\mathbf{y}}^t$ in (c) the same. Because by design the innovation matrices $\tilde{\mathbf{X}}^t$ have orthogonal columns however, we do not have to solve a t-dimensional linear regression problem from scratch at every time point t. Instead, by exploiting orthogonality, the full least-squares minimization at time d decomposes into finding the projection onto each vector individually, that is

$$\min_{\mathbf{w}\in\Re^d} \|\mathbf{y} - \tilde{\mathbf{X}}\mathbf{w}\|_2^2 = \sum_{j=1}^{d} \min_{w_j\in\Re}[w_j^2\|\Delta\mathbf{x}^j\|^2 - 2\sum_{j=1}^{d} w_j(\Delta\mathbf{x}^j)^\top\mathbf{y}].$$

so that the individual minimizer of the one-dimensional problems constitute the elements of the minimizer of the d-dimensional problem. As a consequence, at each time t, the j-th coefficient (with $j = 1, \ldots, t$) of the weight vector $\hat{\mathbf{w}}^t$ can be obtained by $(\hat{\mathbf{w}}^t)_j = \frac{(\Delta\mathbf{x}^j)^\top\mathbf{y}}{\|\Delta\mathbf{x}^j\|^2}$. The update to $\widehat{\mathbf{w}}^t$ from $\widehat{\mathbf{w}}^{t-1}$ thus consists of merely attaching the new coefficient $(\hat{\mathbf{w}}^t)_t$ to the previous weight vector $\widehat{\mathbf{w}}^{t-1}$.

(e) Now let's consider the more general setting where we now want to predict a target vector $\mathbf{y} \in \Re^k$ from a feature vector $\mathbf{x} \in \Re^d$, thus having a training set consisting of observations $(\mathbf{x}_i, \mathbf{y}_i)$ for $i = 1, \ldots, n$.

Instead of learning a weight vector $\mathbf{w} \in \Re^d$, we now want a linear estimate $\widehat{\mathbf{y}} = \hat{\mathbf{W}}\mathbf{x}$ with a weight matrix $\hat{\mathbf{W}} \in \Re^{k\times d}$ instead. From our samples, we obtain wide matrices $\mathbf{Y} \in \Re^{k\times n}$ with columns $\mathbf{y}_1, \ldots, \mathbf{y}_n$ and $\mathbf{X} \in \Re^{d\times n}$ with columns $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (note that this is the transpose of $\mathbf{X}$ in Figure 2!). In order to learn $\hat{\mathbf{W}}$ we now want to minimize $\|\mathbf{Y} - \mathbf{W}\mathbf{X}\|_F^2$ where $\|\cdot\|_F$ denotes the Frobenius norm of matrices, i.e. $\|\mathbf{L}\|_F^2 = \mathrm{tr}(\mathbf{L}^\top\mathbf{L})$.

Show how to find $\hat{\mathbf{W}} = \arg\min_{\mathbf{W}\in\Re^{k\times d}} \|\mathbf{Y} - \mathbf{W}\mathbf{X}\|_F^2$ using vector calculus as reviewed in Discussion 0 and 1.

Solution: Recall the following trace identity for two matrices $\mathbf{P} \in \mathbb{R}^{p\times q}$ and $\mathbf{Q} \in \mathbb{R}^{q\times p}$.

$$\mathrm{trace}(\mathbf{PQ}) = \mathrm{trace}(\mathbf{QP}).$$

(You can show this by writing out the trace of both matrices and verifying that they are equal.)

Now expand the objective as

$$\begin{aligned}\|\mathbf{Y} - \mathbf{W}\mathbf{X}\|_F^2 &= \mathrm{trace}((\mathbf{Y} - \mathbf{W}\mathbf{X})^\top(\mathbf{Y} - \mathbf{W}\mathbf{X})) \\ &= \mathrm{trace}(\mathbf{Y}^\top\mathbf{Y}) - 2\mathrm{trace}(\mathbf{Y}^\top\mathbf{W}\mathbf{X}) + \mathrm{trace}(\mathbf{X}^\top\mathbf{W}^\top\mathbf{W}\mathbf{X})\end{aligned}$$

$$= \text{trace}(\mathbf{Y}^\top\mathbf{Y}) - 2\text{trace}(\mathbf{X}\mathbf{Y}^\top\mathbf{W}) + \text{trace}(\mathbf{W}\mathbf{X}\mathbf{X}^\top\mathbf{W}^\top).$$

Let us now use two useful identities from vector calculus (see discussion 0 and discussion 1).

$$\frac{\partial}{\partial \mathbf{P}}\text{trace}(\mathbf{Q}^\top\mathbf{P}) = \mathbf{Q}^\top.$$
$$\frac{\partial}{\partial \mathbf{P}}\text{trace}(\mathbf{P}\mathbf{Q}\mathbf{P}^\top) = (\mathbf{Q}+\mathbf{Q}^\top)\mathbf{P}^\top.$$

Let us now differentiate the objective and apply these identities to obtain

$$\frac{\partial}{\partial \mathbf{W}}\|\mathbf{Y}-\mathbf{W}\mathbf{X}\|_F^2 = \frac{\partial}{\partial \mathbf{W}}\left[\text{trace}(\mathbf{Y}^\top\mathbf{Y}) - 2\text{trace}(\mathbf{X}\mathbf{Y}^\top\mathbf{W}) + \text{trace}(\mathbf{W}\mathbf{X}\mathbf{X}^\top\mathbf{W}^\top)\right]$$
$$= -2\mathbf{X}\mathbf{Y}^\top + 2\mathbf{X}\mathbf{X}^\top\mathbf{W}^\top$$

Setting the derivative to zero and moving terms around yields

$$\mathbf{W} = \mathbf{Y}\mathbf{X}^\top(\mathbf{X}\mathbf{X}^\top)^{-1}.$$

(f) In the setting of problem (e), **argue why** the computation of the best linear prediction $\widehat{\mathbf{y}}$ of a target vector $\mathbf{y}$ using a feature vector $\mathbf{x}$ can be solved by separately finding the best linear prediction for each measurement $(\mathbf{y})_j$ of the target vector $\mathbf{y}$.

Solution: Let us answer the more general question, of why the above problem in part (d) decomposes into n independent problems. We have by definition of the Frobenius norm

$$\|\mathbf{Y}-\mathbf{W}\mathbf{X}\|_F^2 = \sum_{\ell=1}^{k}\|\mathbf{y}^\ell - \mathbf{w}_\ell^\top\mathbf{X}\|_2^2,$$

where $\mathbf{y}^\ell$ is the ℓth row of $\mathbf{Y}$, and $\mathbf{w}_\ell^\top$ is the ℓth row of matrix $\mathbf{W}$. Note the slight change in notation for different matrices.

Since the matrices $\mathbf{X}$ and $\mathbf{Y}$ are fixed, the minimization of the above with respect to $\mathbf{W}$ decomposes as

$$\min_{\mathbf{W}\in\Re^{k\times d}}\|\mathbf{Y}-\mathbf{W}\mathbf{X}\|_F^2 = \min_{\mathbf{w}_1,\ldots,\mathbf{w}_k\in\Re^d}\sum_{\ell=1}^{k}\|\mathbf{y}^\ell - \mathbf{w}_\ell^\top\mathbf{X}\|_2^2$$
$$= \sum_{\ell=1}^{k}\min_{\mathbf{w}_\ell\in\Re^d}\|\mathbf{y}^\ell - \mathbf{w}_\ell^\top\mathbf{X}\|_2^2$$

where the last equation holds because there are no terms linking $\mathbf{w}_i$ and $\mathbf{w}_j$ for $i \neq j$, and so each minimization can be performed independently.

5 A Simple Classification Approach

Make sure to submit the code you write in this problem to "HW1 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.

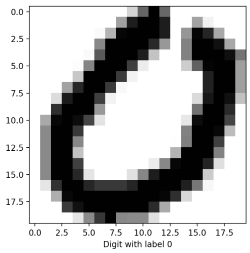
The goal of this problem is to show that even a regression technique like linear regression can be used to solve a classification problem. This can be achieved by regressing the data points in the training set against -1 or 1 depending on their class and then using the level set of 0 of the regression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

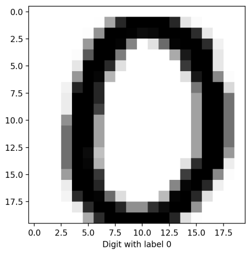
Later in lecture we will learn why linear regression is not the optimal approach for classification and we will study better approaches like logistic regression, SVMs and neural networks.

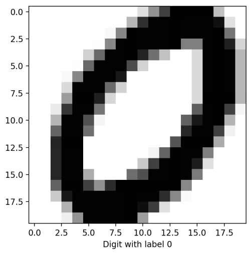
(a) The dataset used in this exercise is a subset of the MNIST dataset. The MNIST dataset assigns each image of a handwritten digit their value from 0 to 9 as a class. For this problem we only keep digits that are assigned a 0 or 1, so we simplify the problem to a two-class classification problem.

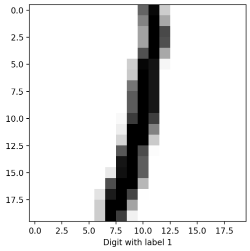
Download and visualize the dataset (example code included). Include three images that are labeled as 0 and three images that are labeled as 1 in your submission.

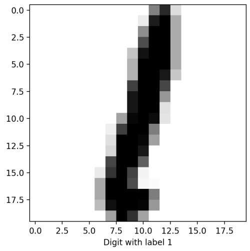
Solution: We have digits labeled as zero at indices 2, 3, 5 in the dataset. We have digits labeled as one at indices 0, 1, 4 in the dataset.

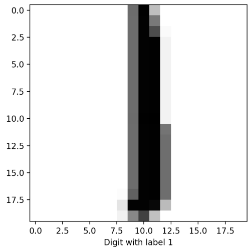












This can be visualized with the following script:

```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:

```
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) Somebody suggests to use 0 (for class 0) and 1 (for class 1) as the entries for the target vector $\mathbf{b}$. Try out how well this is doing (make sure to adapt the classification rule, i.e. the threshold set for the outputs). **Report what percentage of digits are correctly classified using this approach on the training set and test set.** How are the performances of the two approaches if you add a bias column to the feature matrix $\mathbf{X}$? A bias column is a column of all ones, i.e. the new feature matrix $\mathbf{X}'$ is

$$\mathbf{X}' = \begin{bmatrix} \mathbf{x}_0^\top & 1 \\ \mathbf{x}_1^\top & 1 \\ \vdots & \vdots \\ \mathbf{x}_n^\top & 1 \end{bmatrix}$$

Report what percentage of digits are correctly classified using regression targets $0/1$ and $-1/1$ with bias on the training set and test set. Try to explain the results!

Solution: First using 0 and 1 labels, without bias:

```
y2 = 0.5*(y + 1)
w2 = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, y2))

def classify2(w, features):
    return 1*(np.dot(features, w) >= 0.5)

train_error2 = 1.0 * sum(classify2(w2, train_features) != train_labels) / n_train
test_error2 = 1.0 * sum(classify2(w2, test_features) != test_labels) / n_test

print("classification error on the train dataset is {}".format(train_error2))
print("classification error on the test dataset is {}".format(test_error2))
```

The output is:

```
classification error on the train dataset is 0.0103243568786
classification error on the test dataset is 0.00851063829787
```

Now using 0 and 1 labels, with bias:

```
Xbias = np.hstack([np.ones((X.shape[0], 1)), X])
w3 = np.linalg.solve(np.dot(Xbias.T, Xbias), np.dot(Xbias.T, y2))
train_error3 = 1.0 * sum(classify2(w3, Xbias) != train_labels) / n_train
Xbias_test = np.hstack([np.ones((test_features.shape[0], 1)), test_features])
```

```
test_error3 = 1.0 * sum(classify2(w3, Xbias_test) != test_labels) / n_test

print("classification error on the train dataset is {}".format(train_error3))
print("classification error on the test dataset is {}".format(test_error3))
```

The output is:

```
classification error on the train dataset is 0.00585046889787
classification error on the test dataset is 0.00378250591017
```

Using -1 and 1 labels, with bias:

```
w4 = np.linalg.solve(np.dot(Xbias.T, Xbias), np.dot(Xbias.T, y))
train_error4 = 1.0 * sum(classify(w4, Xbias) != train_labels) / n_train
test_error4 = 1.0 * sum(classify(w4, Xbias_test) != test_labels) / n_test

residuals = np.dot(Xbias, w4) - y
epsilon = np.linalg.norm(residuals)

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

print("classification error on the train dataset is {}".format(train_error4))
print("classification error on the test dataset is {}".format(test_error4))
```

The output is:

```
regression residual error is 19.1194429218
classification error on the train dataset is 0.00585046889787
classification error on the test dataset is 0.00378250591017
```

So the results are as follows:

	train set	test set
$0/1$ without bias	98.97	99.14
$0/1$ with bias	99.42	99.62
$-1/1$ without bias	99.76	99.81
$-1/1$ with bias	99.42	99.62

So as you see the bias can absorb the differences in choice of the target vector, i.e. if b is the bias, we have

$$\begin{bmatrix} \mathbf{x}_0^\top & 1 \\ \mathbf{x}_1^\top & 1 \\ \vdots & \vdots \\ \mathbf{x}_n^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} - \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} = \mathbf{X}\mathbf{w} - \begin{bmatrix} y_0 - b \\ y_1 - b \\ \vdots \\ y_n - b \end{bmatrix}$$

and therefore with $b = 1/2$ and $\mathbf{w} \to w/2$ we can go from $0/1$ target to $-1/1$ target.

Also note that introducing the bias vector makes the $-1/1$ regression loss go down, but a little bit surprisingly not the the classification loss.

6 Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn the material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.