CS 189 Introduction to Machine Learning
Summer 2018 HW3

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_6,2_7,2_8,2_9,2_10,3_1,3_2,3_3,3_4,3_5,3_6,3_7,3_8,4_1,4_2,4_3,4_4,5.

This homework is due **Monday, July 9th at 11:59pm.**

2 PCA and Random Projections

In this question, we revisit the task of dimensionality reduction. Dimensionality reduction is useful for several purposes, including but not restricted to, visualization, storage, faster computation etc. While reducing dimension is useful, it is desirable to demand that such reductions preserve some properties of the original data. Often, certain geometric properties like distance and inner products are important to perform certain machine learning tasks. And as a result, we may want to perform dimensionality reduction but ensuring that we approximately maintain the pairwise distances and inner products.

While you have already seen many properties of PCA so far, in this question we investigate if random projections are a good idea for dimensionality reduction. A few advantages of random projections over PCA can be: (1) PCA is expensive when the underlying dimension is high and the number of principal components is also large (however note that there are several very fast algorithms dedicated to doing PCA), (2) PCA requires you to have access to the feature matrix for performing computations. The second requirement of PCA is a bottle neck when you want to take only a low dimensional measurement of a very high dimensional data, e.g., in FMRI and in compressed sensing. In such cases, one needs to design a projection scheme before seeing the data. We now turn to a concrete setting to study a few properties of PCA and random projections.

Suppose you are given n points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $\mathbb{R}^d$. Define the $n \times d$ matrix $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix}$ where each row of the matrix represents one of the given points. In this problem, we will consider a few low-dimensional linear embedding $\psi : \mathbb{R}^d \mapsto \mathbb{R}^k$ that maps vectors from $\mathbb{R}^d$ to $\mathbb{R}^k$.

Let $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ denote the singular value decomposition of the matrix $\mathbf{X}$. Assume that $n \geq d$ and let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_d$ denote the singular values of $\mathbf{X}$.

Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d$ denote the columns of the matrix $\mathbf{V}$. We now consider the following k-dimensional PCA embedding: $\psi_{\text{PCA}}(\mathbf{x}) = (\mathbf{v}_1^\top \mathbf{x}, \ldots, \mathbf{v}_k^\top \mathbf{x})^\top$. Note that this embedding projects a d-dimensional vector on the linear span of the set $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ and that $\mathbf{v}_i^\top \mathbf{x}$ denotes the i-th coordinate of the projected vector in the new space.

We begin with a few matrix algebra relationships, and use them to investigate certain mathematical

properties of PCA and random projections in the first few parts, and then see them in action on a synthetic dataset in the later parts.

Notation: The symbol $[n]$ stands for the set $\{1, \ldots, n\}$.

(a) **What is the ij-th entry of the matrices $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top\mathbf{X}$? Express the matrix $\mathbf{X}\mathbf{X}^\top$ in terms of $\mathbf{U}$ and $\boldsymbol{\Sigma}$, and, express the matrix $\mathbf{X}^\top\mathbf{X}$ in terms of $\boldsymbol{\Sigma}$ and $\mathbf{V}$.**

Solution: Let x_{ik} denote the k-th entry of the vector $\mathbf{x}_i$. Then, we have

$$(\mathbf{X}\mathbf{X}^\top)_{ij} = \left[\begin{bmatrix}\mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top\end{bmatrix}\begin{bmatrix}\mathbf{x}_1 & \ldots & \mathbf{x}_n\end{bmatrix}\right]_{ij} = \mathbf{x}_i^\top\mathbf{x}_j = \sum_{k=1}^{d} x_{ik}x_{jk}$$

and

$$(\mathbf{X}^\top\mathbf{X})_{ij} = \sum_{k=1}^{n}(\mathbf{X}^\top)_{ik}(\mathbf{X})_{kj} = \sum_{k=1}^{n} x_{ki}x_{kj}.$$

Furthermore, we have

$$\mathbf{X}\mathbf{X}^\top = \mathbf{U}\boldsymbol{\Sigma}^2\mathbf{U}^\top \quad \text{and} \quad \mathbf{X}^\top\mathbf{X} = \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}^\top.$$

(b) **Show that**

$$\psi_{\text{PCA}}(\mathbf{x}_i)^\top\psi_{\text{PCA}}(\mathbf{x}_j) = \mathbf{x}_i^\top\mathbf{V}_k\mathbf{V}_k^\top\mathbf{x}_j \quad \text{where} \quad \mathbf{V}_k = \begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \ldots & \mathbf{v}_k\end{bmatrix}.$$

Also **show that** $\mathbf{V}_k\mathbf{V}_k^\top = \mathbf{V}\mathbf{I}^k\mathbf{V}^\top$, where the matrix $\mathbf{I}^k$ denotes a $d \times d$ diagonal matrix with first k diagonal entries as 1 and all other entries as zero.

Solution: Note that

$$\psi_{\text{PCA}}(\mathbf{x}) = \begin{bmatrix}\mathbf{v}_1^\top\mathbf{x} \\ \vdots \\ \mathbf{v}_k^\top\mathbf{x}\end{bmatrix} = \begin{bmatrix}\mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_k^\top\end{bmatrix}\mathbf{x} = \mathbf{V}_k^\top\mathbf{x}$$

and thus we have

$$\psi_{\text{PCA}}(\mathbf{x}_i)^\top\psi_{\text{PCA}}(\mathbf{x}_j) = \mathbf{x}_i^\top\mathbf{V}_k\mathbf{V}_k^\top\mathbf{x}_j$$

as required.

For the second part, we note that for two matrices

$$\mathbf{A} = \begin{bmatrix}\mathbf{a}_1 & \ldots & \mathbf{a}_d\end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix}\mathbf{b}_1^\top \\ \vdots \\ \mathbf{b}_d^\top\end{bmatrix}$$

we have

$$\mathbf{AB} = \sum_{i=1}^{d} \mathbf{a}_i \mathbf{b}_i^\top. \tag{1}$$

Furthermore, given a diagonal matrix $\mathbf{\Gamma}$ of size d-by$d$, with $(\mathbf{\Gamma})_{ii} = \gamma_i$, we have

$$\mathbf{A\Gamma B} = \sum_{i=1}^{d} \gamma_i \mathbf{a}_i \mathbf{b}_i^\top. \tag{2}$$

Using equation (1), we have

$$\mathbf{V}_k \mathbf{V}_k^\top = \sum_{i=1}^{k} \mathbf{v}_i \mathbf{v}_i^\top$$

Furthermore, using equation (2), we obtain that

$$\mathbf{V}\mathbf{I}^k\mathbf{V}^\top = \sum_{i=1}^{d} \mathbf{v}_i (\mathbf{I}^k)_{ii} \mathbf{v}_i^\top = \sum_{i=1}^{k} \mathbf{v}_i \mathbf{v}_i^\top.$$

since $(\mathbf{I}^k)_{ii} = 1$ for $i = 1, \ldots, k$ and 0 for $i > k$. The students are not required to prove so rigorously. The key take away from this part is to be comfortable with different ways to write matrices, and keep in mind the representations (1) and (2).

(c) Suppose that we know the first k singular values are the dominant singular values. In particular, we are given that

$$\frac{\sum_{i=1}^{k} \sigma_i^4}{\sum_{i=1}^{d} \sigma_i^4} \geq 1 - \epsilon,$$

for some $\epsilon \in (0, 1)$. Then **show that the PCA projection to the first k-right singular vectors preserves the *inner products* on average:**

$$\frac{1}{\sum_{i=1}^{n}\sum_{j=1}^{n} (\mathbf{x}_i^\top \mathbf{x}_j)^2} \sum_{i=1}^{n}\sum_{j=1}^{n} |(\mathbf{x}_i^\top \mathbf{x}_j) - (\psi_{\text{PCA}}(\mathbf{x}_i)^\top \psi_{\text{PCA}}(\mathbf{x}_j))|^2 \leq \epsilon. \tag{3}$$

Thus, we find that if there are dominant singular values, PCA projection can preserve the inner products on average.

Hint: Using previous two parts and the definition of Frobenius norm might be useful.

Solution: Using the solution from part (a), we have

$$(\mathbf{X}\mathbf{X}^\top)_{ij} = \mathbf{x}_i^\top \mathbf{x}_j. \tag{4}$$

Using the solution from part (b), we have

$$\Psi = \begin{bmatrix} \psi_{\text{PCA}}(\mathbf{x}_1)^\top \\ \vdots \\ \psi_{\text{PCA}}(\mathbf{x}_n)^\top \end{bmatrix} = \mathbf{X}\mathbf{V}_k. \tag{5}$$

We have

$$\sum_{i=1}^{n}\sum_{j=1}^{n}|(\mathbf{x}_i^\top\mathbf{x}_j)-(\psi_{\text{PCA}}(\mathbf{x}_i)^\top\psi_{\text{PCA}}(\mathbf{x}_j))|^2 \overset{(i)}{=} \|\mathbf{X}\mathbf{X}^\top-\Psi\Psi^\top\|_{\text{F}}^2$$
$$\overset{(ii)}{=} \|\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top-\mathbf{X}\mathbf{V}_k\mathbf{V}_k^\top\mathbf{X}^\top\|_{\text{F}}^2$$
$$\overset{(iii)}{=} \|\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top-\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top\mathbf{V}\mathbf{I}^k\mathbf{V}^\top\mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\|_{\text{F}}^2$$
$$\overset{(iv)}{=} \|\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top-\mathbf{U}\mathbf{\Sigma}\mathbf{I}_k\mathbf{\Sigma}\mathbf{U}^\top\|_{\text{F}}^2,$$

where for step (i) we applied the definition of the Frobenius norm and the equations (4) and (5), for step (ii) we have used the solution from part (a) to replace $\mathbf{X}\mathbf{X}^\top=\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top$ and for step (iii), we have used the singular value decomposition $\mathbf{X}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, and the second result from part (b). For step (iv) we make use of the fact that $\mathbf{V}^\top\mathbf{V}=\mathbf{I}$.

Now we use the fact that Frobenius norm is unitary invariant (Discussion 1, problem 2 (b)). Thus, we have

$$\|\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^\top-\mathbf{U}\mathbf{\Sigma}\mathbf{I}_k\mathbf{\Sigma}\mathbf{U}^\top\|_{\text{F}}^2=\|\mathbf{U}\mathbf{\Sigma}^2-\mathbf{\Sigma}\mathbf{I}_k\mathbf{\Sigma}\mathbf{U}^\top\|_{\text{F}}^2 \quad =\|\mathbf{\Sigma}^2-\mathbf{\Sigma}\mathbf{I}_k\mathbf{\Sigma}\|_{\text{F}}^2=\sum_{i=k+1}^{d}\sigma_i^4.$$

We now use the fact that

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(\mathbf{x}_i^\top\mathbf{x}_j)^2=\|\mathbf{X}\mathbf{X}^\top\|_{\text{F}}^2=\|\Sigma^2\|_{\text{F}}^2=\sum_{i=1}^{d}\sigma_i^4,$$

we obtain that

$$\frac{1}{\sum_{i=1}\sum_{j=1}(\mathbf{x}_i^\top\mathbf{x}_j)^2}\sum_{i=1}^{n}\sum_{j=1}^{n}|(\mathbf{x}_i^\top\mathbf{x}_j)-(\psi_{\text{PCA}}(\mathbf{x}_i)^\top\psi_{\text{PCA}}(\mathbf{x}_j))|^2=\frac{\sum_{i=k+1}^{d}\sigma_i^4}{\sum_{i=1}^{d}\sigma_i^4}$$
$$=1-\frac{\sum_{i=1}^{k}\sigma_i^4}{\sum_{i=1}^{d}\sigma_i^4}$$
$$\leq\epsilon.$$

(d) Now consider a different embedding $\psi:\mathbb{R}^d\mapsto\mathbb{R}^k$ which preserves all pairwise distances and norms up-to a multiplicative factor, that is,

$$(1-\epsilon)\|\mathbf{x}_i\|^2\leq\|\psi(\mathbf{x}_i)\|^2\leq(1+\epsilon)\|\mathbf{x}_i\|^2 \qquad \text{for all } i\in[n], \quad \text{and} \quad (6)$$
$$(1-\epsilon)\|\mathbf{x}_i-\mathbf{x}_j\|^2\leq\|\psi(\mathbf{x}_i)-\psi(\mathbf{x}_j)\|^2\leq(1+\epsilon)\|\mathbf{x}_i-\mathbf{x}_j\|^2 \qquad \text{for all } i,j\in[n], \quad (7)$$

where $0<\epsilon\ll1$ is a small scalar. Further assume that $\|\mathbf{x}_i\|\leq1$ for all $i\in[n]$. **Show that the embedding ψ satisfying equations (7) and (6) preserves *each pairwise inner product*:**

$$|\psi(\mathbf{x}_i)^\top\psi(\mathbf{x}_j)-(\mathbf{x}_i^\top\mathbf{x}_j)|\leq C\epsilon, \qquad \text{for all } i,j\in[n], \quad (8)$$

for some constant C. Thus, we find that if an embedding approximately preserves distances and norms upto a small multiplicative factor, and the points have bounded norms, then inner products are also approximately preserved upto an additive factor.

Hint: You may use the Cauchy-Schwarz inequality.

Solution: We will show that

$$-3\epsilon \leq (\mathbf{x}_i^\top \mathbf{x}_j - \psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)) \leq 3\epsilon.$$

Taking absolute value, we obtain the claimed result for $C = 3$.

Upper bound: Expanding the second inequality from equation (7), we obtain

$$\left\|\psi(\mathbf{x}_i)\right\|^2 + \left\|\psi(\mathbf{x}_j)\right\|^2 - 2\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) \leq (1+\epsilon)\|\mathbf{x}_i\|^2 + (1+\epsilon)\left\|\mathbf{x}_j\right\|^2 - 2(1+\epsilon)\mathbf{x}_i^\top \mathbf{x}_j.$$

We use the lower bounds from equation (6):

$$\begin{aligned}\left\|\psi(\mathbf{x}_i)\right\|^2 &\geq (1-\epsilon)\|\mathbf{x}_i\|^2 \\ \left\|\psi(\mathbf{x}_j)\right\|^2 &\geq (1-\epsilon)\left\|\mathbf{x}_j\right\|^2.\end{aligned}$$

And hence we have

$$(1-\epsilon)\|\mathbf{x}_i\|^2 + (1-\epsilon)\left\|\mathbf{x}_j\right\|^2 - 2\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) \leq (1+\epsilon)\|\mathbf{x}_i\|^2 + (1+\epsilon)\left\|\mathbf{x}_j\right\|^2 - 2(1+\epsilon)\mathbf{x}_i^\top \mathbf{x}_j.$$

Moving terms around we obtain

$$\begin{aligned}2(\mathbf{x}_i^\top \mathbf{x}_j - \psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)) &\leq 2\epsilon(\|\mathbf{x}_i\|^2 + \left\|\mathbf{x}_j\right\|^2 - \mathbf{x}_i^\top \mathbf{x}_j) \\ &\leq 2\epsilon(\|\mathbf{x}_i\|^2 + \left\|\mathbf{x}_j\right\|^2 + \|\mathbf{x}_i\|\left\|\mathbf{x}_j\right\|) \\ &\leq 6\epsilon,\end{aligned}$$

since $\|\mathbf{x}_i\| \leq 1$. Thus we have

$$(\mathbf{x}_i^\top \mathbf{x}_j - \psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)) \leq 3\epsilon.$$

Lower bound: Expanding the first inequality from equation (7), we obtain

$$\left\|\psi(\mathbf{x}_i)\right\|^2 + \left\|\psi(\mathbf{x}_j)\right\|^2 - 2\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) \geq (1-\epsilon)\|\mathbf{x}_i\|^2 + (1-\epsilon)\left\|\mathbf{x}_j\right\|^2 - 2(1-\epsilon)\mathbf{x}_i^\top \mathbf{x}_j.$$

We use the upper bounds from equation (6):

$$\begin{aligned}\left\|\psi(\mathbf{x}_i)\right\|^2 &\leq (1+\epsilon)\|\mathbf{x}_i\|^2 \\ \left\|\psi(\mathbf{x}_j)\right\|^2 &\leq (1+\epsilon)\left\|\mathbf{x}_j\right\|^2.\end{aligned}$$

And hence we have

$$(1+\epsilon)\|\mathbf{x}_i\|^2 + (1+\epsilon)\left\|\mathbf{x}_j\right\|^2 - 2\psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) \geq (1-\epsilon)\|\mathbf{x}_i\|^2 + (1-\epsilon)\left\|\mathbf{x}_j\right\|^2 - 2(1-\epsilon)\mathbf{x}_i^\top \mathbf{x}_j.$$

Moving terms around we obtain

$$\begin{aligned} 2(\mathbf{x}_i^\top \mathbf{x}_j - \psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j)) &\geq -2\epsilon(\|\mathbf{x}_i\|^2 + \left\|\mathbf{x}_j\right\|^2 - \mathbf{x}_i^\top \mathbf{x}_j) \\ &\geq -2\epsilon(\|\mathbf{x}_i\|^2 + \left\|\mathbf{x}_j\right\|^2 + \|\mathbf{x}_i\|\left\|\mathbf{x}_j\right\|) \\ &\geq -6\epsilon, \end{aligned}$$

since $\|\mathbf{x}_i\| \leq 1$. And thus we are done.

(e) Now we consider the *random projection* using a Gaussian matrix as introduced in the section. In next few parts, we work towards proving that if the dimension of projection is moderately big, then with high probability, the random projection preserves norms and pairwise distances approximately as described in equations (7) and (6).

Consider the random matrix $\mathbf{J} \in \mathbb{R}^{k \times d}$ with each of its entry being i.i.d. $\mathcal{N}(0,1)$ and consider the map $\psi_{\mathbf{J}} : \mathbb{R}^d \mapsto \mathbb{R}^k$ such that $\psi_{\mathbf{J}}(\mathbf{x}) = \frac{1}{\sqrt{k}}\mathbf{J}\mathbf{x}$. **Show that for any *fixed non-zero vector* $\mathbf{u}$, the random variable $\dfrac{\|\psi_{\mathbf{J}}(\mathbf{u})\|^2}{\|\mathbf{u}\|^2}$ can be written as**

$$\frac{1}{k}\sum_{i=1}^{k} Z_i^2$$

where Z_i's are i.i.d. $\mathcal{N}(0,1)$ random variables.

Solution: Let $\mathbf{J} = \begin{bmatrix} J_1^\top \\ \vdots \\ J_k^\top \end{bmatrix}$. Then, we have

$$\frac{\|\psi_{\mathbf{J}}(\mathbf{u})\|^2}{\|\mathbf{u}\|^2} = \frac{1}{k}\sum_{i=1}^{k} \frac{(J_i^\top \mathbf{u})^2}{\|\mathbf{u}\|^2}.$$

Note that

$$J_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \mathbf{I}_d),$$

and hence

$$(J_i^\top \mathbf{u}) \sim \mathcal{N}(0, \|\mathbf{u}\|^2),$$

and consequently, we have

$$Z_i = \frac{J_i^\top \mathbf{u}}{\|\mathbf{u}\|} \overset{i.i.d.}{\sim} \mathcal{N}(0,1).$$

As a result, we conclude that

$$\frac{\|\psi_{\mathbf{J}}(\mathbf{u})\|^2}{\|\mathbf{u}\|^2} = \frac{1}{k}\sum_{i=1}^{k} \frac{(J_i^\top \mathbf{u})^2}{\|\mathbf{u}\|^2}. = \frac{1}{k}\sum_{i=1}^{k} Z_i^2$$

where Z_i's are i.i.d. $\mathcal{N}(0,1)$ random variables.

(f) **(BONUS)** For i.i.d. $Z_i \sim \mathcal{N}(0,1)$, we have the following probability bound

$$\mathbb{P}\left[\left|\frac{1}{k}\sum_{i=1}^{k} Z_i^2\right| \notin (1-t, 1+t)\right] \leq 2e^{-kt^2/8}, \quad \text{for all } t \in (0,1).$$

Note that this bound suggests that $\sum_{i=1}^{k} Z_i^2 \approx \sum_{i=1} E[Z_i^2] = k$ with high probability. In other words, sum of square of Gaussian random variables concentrates around its mean with high probability. Using this bound and the previous part, now **show that if** $k \geq \frac{16}{\epsilon^2}\log\left(\frac{N}{\delta}\right)$**, then**

$$\mathbb{P}\left[\text{for all } i,j \in [n], i \neq j, \frac{\left\|\psi_{\mathbf{J}}(\mathbf{x}_i) - \psi_{\mathbf{J}}(\mathbf{x}_j)\right\|^2}{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2} \in (1-\epsilon, 1+\epsilon)\right] \geq 1-\delta.$$

That is show that for k large enough, with high probability the random projection $\psi_{\mathbf{J}}$ approximately preserves the pairwise distances. Using this result, we can conclude that random projection serves as a good tool for dimensionality reduction if we project to enough number of dimensions. This result is popularly known as the *Johnson-Lindenstrauss Lemma*.

Hint 1: The following (powerful technique cum) bound might be useful: For a set of events A_{ij}, we have

$$\mathbb{P}\left[\cap_{i,j} A_{ij}\right] = 1 - \mathbb{P}\left[(\cap_{i,j} A_{ij})^c\right] = 1 - \mathbb{P}\left[\cup_{i,j} A_{ij}^c\right] \geq 1 - \sum_{ij} \mathbb{P}\left[A_{ij}^c\right].$$

You may define the event $A_{ij} = \left\{\frac{\left\|\psi_{\mathbf{J}}(\mathbf{x}_i) - \psi_{\mathbf{J}}(\mathbf{x}_j)\right\|^2}{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2} \in (1-\epsilon, 1+\epsilon)\right\}$ and use the union bound above.

Solution: We define the event as given in the hint. For any $i, j \in [n]$, $i \neq j$, define

$$A_{ij} = \left\{\frac{\left\|\psi_{\mathbf{J}}(\mathbf{x}_i) - \psi_{\mathbf{J}}(\mathbf{x}_j)\right\|^2}{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2} \in (1-\epsilon, 1+\epsilon)\right\}$$

Note that the event

$$\mathcal{A} = \left\{\text{for all } i,j \in [n], i \neq j, \frac{\left\|\psi_{\mathbf{J}}(\mathbf{x}_i) - \psi_{\mathbf{J}}(\mathbf{x}_j)\right\|^2}{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2} \in (1-\epsilon, 1+\epsilon)\right\}$$

is equivalent to the intersection of all events A_{ij}, since all events A_{ij} must occur for $\mathcal{A}$ to occur. That is

$$\mathcal{A} = \bigcap_{i\in[n], j\in n, i\neq j} A_{ij}.$$

Note that there are $\binom{n}{2}$ events. Now as given in the hint, we have

$$\begin{aligned}
\mathbb{P}\left[\mathcal{A}\right] &= \mathbb{P}\left[\bigcap_{i,j\in[n],i\neq j} A_{ij}\right] \\
&= 1 - \mathbb{P}\left[\left(\bigcap_{i,j\in[n],i\neq j} A_{ij}\right)^c\right] \\
&= 1 - \mathbb{P}\left[\bigcup_{i,j\in[n],i\neq j} A_{ij}^c\right] \\
&\geq 1 - \sum_{i\in[n],j\in[n],i\neq j} \mathbb{P}\left[A_{ij}^c\right].
\end{aligned}$$

Now we have

$$\begin{aligned}
\mathbb{P}\left[A_{ij}^c\right] &= \mathbb{P}\left[\frac{\left\|\psi_{\mathbf{J}}(\mathbf{x}_i) - \psi_{\mathbf{J}}(\mathbf{x}_j)\right\|^2}{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2} \notin (1-\epsilon, 1+\epsilon)\right] \\
&\overset{\text{part (e)}}{=} \mathbb{P}\left[\left|\frac{1}{k}\sum_{i=1}^{k} Z_i^2\right| \notin (1-\epsilon, 1+\epsilon)\right] \\
&\overset{\text{prob. bnd}}{\leq} 2e^{-k\epsilon^2/8}.
\end{aligned}$$

Thus, we have that

$$\begin{aligned}
\mathbb{P}\left[\mathcal{A}\right] &\geq 1 - \sum_{i\in[n],j\in[n],i\neq j} \mathbb{P}\left[A_{ij}^c\right] \\
&\geq 1 - \sum_{i\in[n],j\in[n],i\neq j} 2e^{-k\epsilon^2/8} \\
&\geq 1 - \binom{n}{2} 2e^{-k\epsilon^2/8} \\
&\geq 1 - n(n-1)e^{-k\epsilon^2/8} \\
&\geq 1 - n^2 e^{-k\epsilon^2/8}.
\end{aligned}$$

Now to make this probability greater than δ, we set k such that

$$n^2 e^{-k\epsilon^2/8} \leq \delta \Rightarrow 2\log n - k\frac{\epsilon^2}{8} \leq \log\delta \Rightarrow k \geq \frac{8}{\epsilon^2}\left(2\log n - \log\delta\right).$$

Note that since $\delta < 1$, we have that $k \geq \frac{16}{\epsilon^2}\log\left(\frac{N}{\delta}\right)$ implies that $k \geq \frac{8}{\epsilon^2}\left(2\log n - \log\delta\right)$, and hence we are done.

(g) Suppose there are two clusters of points $S_1 = \{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$ and $S_2 = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ which are far apart, i.e., we have

$$d^2(S_1, S_2) = \min_{u \in S_1, v \in S_2} \|u - v\|^2 \geq \gamma.$$

Then using the previous part, **show that the random projection $\psi_{\mathbf{J}}$ also approximately maintains the distance between the two clusters if k is large enough, that is, with high probability**

$$d^2(\psi_{\mathbf{J}}(S_1), \psi_{\mathbf{J}}(S_2)) = \min_{u \in S_1, v \in S_2} \left\|\psi_{\mathbf{J}}(\mathbf{u}) - \psi_{\mathbf{J}}(\mathbf{v})\right\|^2 \geq (1-\epsilon)\gamma \quad \text{if } k \geq \frac{C}{\epsilon^2}\log(m+n)$$

for some constant C. Note that such a property can help in several machine learning tasks. For example, if the clusters of features for different labels were far in the original dimension, then this problem shows that even after randomly projecting the clusters they will remain far enough and a machine learning model may perform well even with the projected data. We now turn to visualizing some of these conclusions on a synthetic dataset.

Solution: Using previous part, we obtain that pairwise squared distances between $m+n$ points are approximately preserved up to a factor of $(1-\epsilon, 1+\epsilon)$ if $k \geq C\log(m+n)/\epsilon^2$. Now, we consider any pair of points which minimize the objective $\min_{u \in S_1, v \in S_2} \left\|\psi_{\mathbf{J}}(\mathbf{u}) - \psi_{\mathbf{J}}(\mathbf{v})\right\|^2$. Let's call these points $\mathbf{u}_{i^*}$ and $\mathbf{v}_{j^*}$. Then we have

$$\begin{aligned}
\min_{u \in S_1, v \in S_2} \left\|\psi_{\mathbf{J}}(\mathbf{u}) - \psi_{\mathbf{J}}(\mathbf{v})\right\|^2 &= \left\|\psi_{\mathbf{J}}(\mathbf{u}_{i^*}) - \psi_{\mathbf{J}}(\mathbf{v}_{j^*})\right\|^2 \\
&\geq (1-\epsilon)\left\|\mathbf{u}_{i^*} - \mathbf{v}_{j^*}\right\|^2 \\
&\geq (1-\epsilon) \min_{u \in S_1, v \in S_2} \|u - v\|^2 \\
&= (1-\epsilon)\gamma,
\end{aligned}$$

and we are done.

(h) In the next few parts, we visualize the effect of PCA projections and random projections on a classification task. You are given 3 datasets in the data folder and a starter code.

Use the starter code to load the three datasets one by one. Note that there are two unique values in $\mathbf{y}$. **Visualize the features of $\mathbf{X}$ these datasets using (1) Top-2 PCA components, and (2) 2-dimensional random projections. Use the code to project the features to 2 dimensions and then scatter plot the 2 features with a different color for each class.** Note that you will obtain 2 plots for each dataset (total 6 plots for this part). **Do you observe a difference in PCA vs random projections? Do you see a trend in the three datasets?**

Solution: The code for this part and the following parts is as follows:

```
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib inline
import sklearn.linear_model
from sklearn.model_selection import train_test_split

######## PROJECTION FUNCTIONS ##########
```

```
8
9  ## Random Projections ##
10 def random_matrix(d, k):
11     '''
12     d = original dimension
13     k = projected dimension
14     '''
15     return 1./np.sqrt(k)*np.random.normal(0, 1, (d, k))
16
17 def random_proj(X, k):
18     _, d= X.shape
19     return X.dot(random_matrix(d, k))
20
21 ## PCA and projections ##
22 def my_pca(X, k):
23     '''
24     compute PCA components
25     X = data matrix (each row as a sample)
26     k = #principal components
27     '''
28     n, d = X.shape
29     assert(d>=k)
30     _, _, Vh = np.linalg.svd(X)
31     V = Vh.T
32     return V[:, :k]
33
34 def pca_proj(X, k):
35   '''
36   compute projection of matrix X
37   along its first k principal components
38   '''
39     P = my_pca(X, k)
40     # P = P.dot(P.T)
41     return X.dot(P)
42
43
44 ######### LINEAR MODEL FITTING ############
45
46 def rand_proj_accuracy_split(X, y, k):
47   '''
48   Fitting a k dimensional feature set obtained
49   from random projection of X, versus y
50   for binary classification for y in {-1, 1}
51   '''
52
53     # test train split
54     _, d = X.shape
55     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state
       ↪ =42)
56
57     # random projection
58     J = np.random.normal(0., 1., (d, k))
59     rand_proj_X = X_train.dot(J)
60
61     # fit a linear model
62     line = sklearn.linear_model.LinearRegression(fit_intercept=False)
63     line.fit(rand_proj_X, y_train)
64
65     # predict y
66     y_pred=line.predict(X_test.dot(J))
67
68     # return the test error
69     return 1-np.mean(np.sign(y_pred)!= y_test)
70
71 def pca_proj_accuracy(X, y, k):
72     '''
73   Fitting a k dimensional feature set obtained
74   from PCA projection of X, versus y
```

```
  for binary classification for y in {-1, 1}
  '''

    # test-train split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

    # pca projection
    P = my_pca(X_train, k)
    P = P.dot(P.T)
    pca_proj_X = X_train.dot(P)

    # fit a linear model
    line = sklearn.linear_model.LinearRegression(fit_intercept=False)
    line.fit(pca_proj_X, y_train)

     # predict y
    y_pred=line.predict(X_test.dot(P))


    # return the test error
    return 1-np.mean(np.sign(y_pred)!= y_test)



np.random.seed(7283782)

for i in range(3):

    ######## LOADING THE DATASETS #########

    data = np.load('jl_data/data'+str(i+1)+'.npz')
    X = data['X']
    y = data['y']

    n, d = X.shape
    n_trials = 10  # to average the accuracies for random projection

    ######## CLASSIFICATION USING PROJECTED FEATURES #########

    # linear regression with projected features
    random_accuracy = np.zeros((n_trials, d))
    pca_accuracy = np.zeros(d)

    for k in range(1, d+1):
        pca_accuracy[k-1] = pca_proj_accuracy_split(X, y, k)
        for j in range(n_trials):
            random_accuracy[j, k-1] = rand_proj_accuracy_split(X, y, k)

    # take the mean of random projection performance across trials
    random_accuracy = np.mean(random_accuracy, axis=0)

    ######## PLOTTING RESULTS #########

    plt.figure(figsize=[15, 3])
#     plt.suptitle('Data'+str(i), fontsize=15)
    plt.subplot(141)
    p = pca_proj(X, 2)
    plt.scatter(p[:, 0], p[:, 1], c=y)
    plt.title('First 2 PC')

    plt.subplot(142)
    r = random_proj(X, 2)
    plt.scatter(r[:, 0], r[:, 1], c=y)
    plt.title('Random 2D projection')

    _, S, Vh = np.linalg.svd(X)
    plt.subplot(144)
    plt.plot(np.arange(1, d+1), S)
```

```
    plt.title('Singular value decay')

    plt.subplot(143)
    plt.plot(np.arange(1, d+1), random_accuracy, label="Random")
    plt.plot(np.arange(1, d+1), pca_accuracy, label="PCA")
    plt.legend(loc='best')
    plt.title("Accuracy")
    plt.xlabel("\#Dimensions")

    plt.savefig('jl'+str(i+1)+'.pdf')
   # plt.show()
```

Remark: You may notice a slight difference in the projection matrices used for visualization and for generating the features for regression. For PCA visualization we use only the projection coordinates along the principal components. But for performing regression, we need to use the corresponding directions and hence the two projection matrices are slightly different. Put simply, for visualization the code outputs a k-dimensional vector (coordinates in the principal component basis) but for generating regression features it uses the d-dimensional representation of these k-dimensional vectors. For random projections used in the regression code, there is a slight typo – the scaling is missing. However, the learned model does not change because scaling the entire feature matrix does not change the ordinary least squares solution.

For plots, refer to figures 1, 2 and 3.

For the first and second datasets it is hard to see a difference between the 2D projections with PCA or random projection.

For the third dataset, it is clear that PCA projection clearly separates the two clusters of points, while random projection still has significant overlap of the two sets of points.

Using the PCA projections, we can say that the separation between the two sets of features (corresponding to $y = 1$ and $y = -1$) is largest in data3.npz and probably smallest in data1.npz

(i) For each dataset, we will now fit a linear model on different projections of features to perform classification. The code for fitting a linear model with projected features and predicting a label for a given feature, is given to you. Use these functions and write a code that does prediction in the following way: (1) Use top k-PCA features to obtain one set of results, and (2) Use k-dimensional random projection to obtain the second set of results (take average accuracy over 10 random projections for smooth curves). Use the projection functions given in the starter code to select these features. You should vary k from 1 to d where d is the dimension of each feature $\mathbf{x}_i$. **Plot the accuracy for PCA and Random projection as a function of k. Comment on the observations on these accuracies as a function of k and across different datasets.**

Solution: For plots, refer to figures 1, 2 and 3.

PCA: We see that the accuracy of PCA increase very quickly after adding 2-3 features only and then does not increase too much. In fact, for data1.npz it decays little bit after first 2 features indicating overfitting.

Random projection: We see that the accuracy increases moderately with increase in number of dimension of the projection matrix.

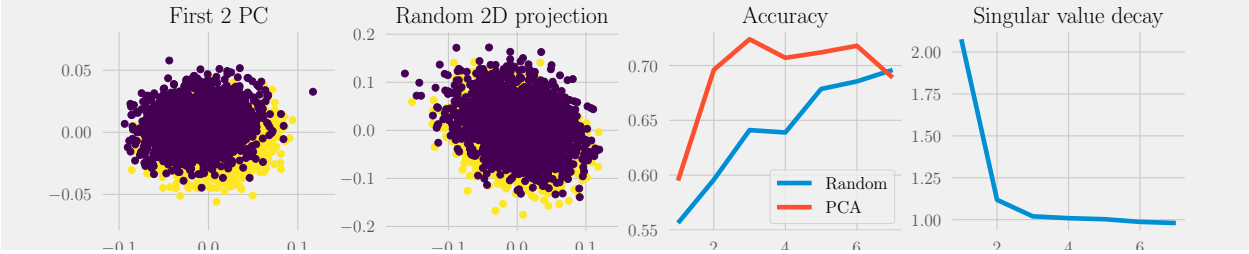


Figure 1: Plots related to data1.npz

The difference in increase in accuracy for PCA and random projection is intuitive as PCA captures effective dimensions – and it appears from the 2D visualization that signal strength is stronger in a few directions. In particular for data3, the data is separable if we just use one principal component. However, random projection is oblivious to the structure in the underlying data if we just use very few directions. Although as the number of dimensions increase, its performance does catch up with PCA.

Across the three datasets, we see that the accuracy for a given set of dimensions increases as we move from data1.npz to data2.npz to data3.npz, again suggesting that the signal strength = the separation between the two clusters is increasing from data1 to data3.

(j) Now plot the singular values for the feature matrices of the three datasets. **Do you observe a pattern across the three datasets? Does it help to explain the performance of PCA observed in the previous parts?**

Solution: For plots, refer to figures 1, 2 and 3.

The significance of the top singular value increases from data1 to data2 with maximum value occurring for data3. For data1, first singular value accounts for approximately $2/8 \approx 25\%$ of the variance, while for data2 it explains $2.5/8.5 \approx 30\%$ variance. For Data3 this percentage is close to $4.5/11 \approx 41\%$.

Intuitively, if the labels are dependent on the direction of maximum variance, we hope to see an increase in accuracy if the ratio of variance explained goes up. PCA 2d visualizations suggest that the labeling is associated with the first principal component.

As a result, we do expect PCA with just one component do very well for data3, and in general the classification accuracy to increase from data1 to data3.

However, note that as the signal to noise ratio increases, the accuracy of random projections also increases.

3 Estimation and approximation in linear regression

In typical applications, we are dealing with data generated by an *unknown* function (with some noise), and our goal is to estimate this function. So far we used linear and polynomial regressions.

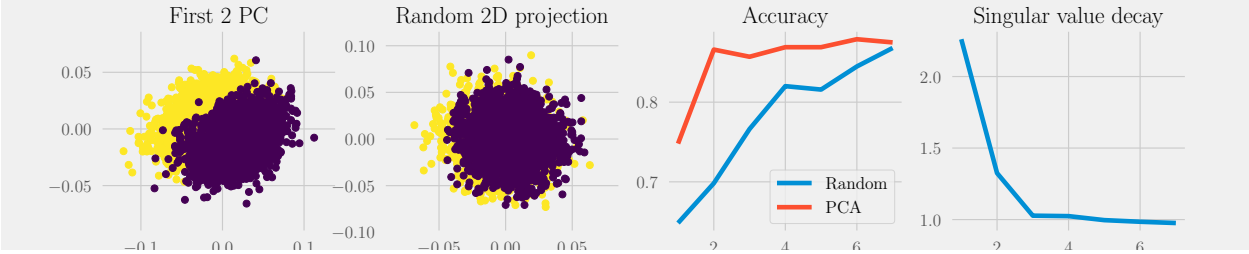


Figure 2: Plots related to data2.npz

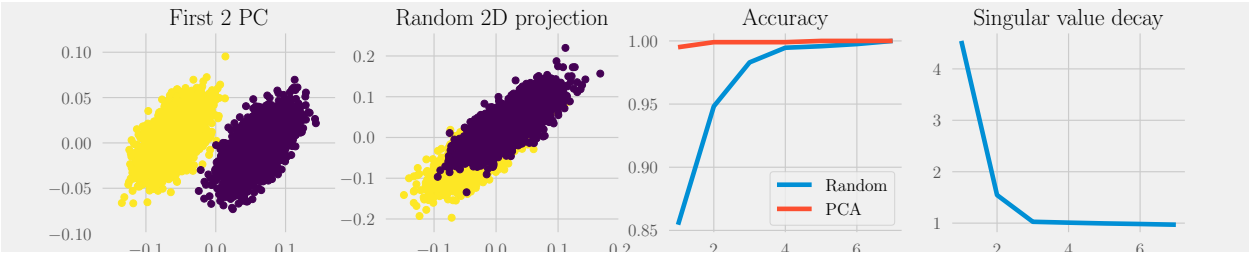


Figure 3: Plots related to data3.npz

In this problem we will explore the quality of polynomial regressions when the true function is not polynomial.

Suppose we are given a full column rank feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ and an observation vector $\mathbf{y} \in \mathbb{R}^n$. Let the vector $\mathbf{y}$ represent the noisy measurement of a true signal $\mathbf{y}^*$:

$$\mathbf{y} = \mathbf{y}^* + \mathbf{z}, \tag{9}$$

with $\mathbf{z} \in \mathbb{R}^n$ representing the random noise in the observation $\mathbf{y}$, where $z_j \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. We define the vectors $\mathbf{w}^*$ and $\hat{\mathbf{w}}$ as follows:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \|\mathbf{y}^* - \mathbf{X}\mathbf{w}\|_2^2 \qquad \text{and} \qquad \hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2.$$

Observe that for a given true signal $\mathbf{y}^*$ the vector $\mathbf{w}^*$ is fixed, but the vector $\hat{\mathbf{w}}$ is a random variable since it is a function of the random noise $\mathbf{z}$. Note that the vector $\mathbf{X}\mathbf{w}^*$ is the best linear fit of the true signal $\mathbf{y}^*$ in the column space of $\mathbf{X}$. Similarly, the vector $\mathbf{X}\hat{\mathbf{w}}$ is the best linear fit of the observed noisy signal $\mathbf{y}$ in the column space of $\mathbf{X}$.

After obtaining $\hat{\mathbf{w}}$, we would like to bound the error $\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}^*\|_2^2$, which is the *prediction error* incurred based on the specific n training samples. In this problem we will see how to get a good estimate of this prediction error. When using polynomial features, we will also learn how to decide the degree of the polynomial when trying to fit a noisy set of observations from a smooth function.

Remark: You can use the closed form solution for OLS and results from Discussion 3 for all parts of this problem. For parts (a)-(c), assume that the feature matrix $\mathbf{X}$ and the true signal vector $\mathbf{y}^*$ are fixed (and not random). Furthermore, in all parts the expectation is taken over the randomness in the noise vector $\mathbf{z}$.

(a) **Show that** $\mathbb{E}[\hat{\mathbf{w}}] = \mathbf{w}^*$ and use this fact to **show that**

$$\mathbb{E}\left[\|\mathbf{y}^* - \mathbf{X}\hat{\mathbf{w}}\|_2^2\right] = \|\mathbf{y}^* - \mathbb{E}[\mathbf{X}\hat{\mathbf{w}}]\|_2^2 + \mathbb{E}\left[\|\mathbf{X}\hat{\mathbf{w}} - \mathbb{E}[\mathbf{X}\hat{\mathbf{w}}]\|_2^2\right].$$

Note that the above decomposition of the squared error corresponds to the sum of bias-squared and the variance of our estimator $\mathbf{X}\hat{\mathbf{w}}$.

Solution: First we will show that $\mathbb{E}[\hat{\mathbf{w}}] = \mathbf{w}^*$. We use the following closed form expressions for the vectors $\hat{\mathbf{w}}$ and $\mathbf{w}^*$:

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}^* \quad \text{and} \quad \hat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}.$$

We have

$$\begin{aligned}\hat{\mathbf{w}} &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top(\mathbf{y}^* + \mathbf{z}) \\ &= (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}^* + (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{z} \\ &= \mathbf{w}^* + (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{z}\end{aligned}$$

Since $\mathbf{w}^*$ is fixed we obtain that $\mathbb{E}[\mathbf{w}^*] = \mathbf{w}^*$. Also notice that $\mathbb{E}[\mathbf{z}] = 0$. Using these two facts, we conclude

$$\mathbb{E}[\hat{\mathbf{w}}] = \mathbb{E}[\mathbf{w}^*] + \mathbb{E}[(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{z}] = \mathbf{w}^* + (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbb{E}[\mathbf{z}] = \mathbf{w}^*.$$

Now we will show the second part of the question. In discussion 3, we defined projection of a vector $\mathbf{v}$ onto column space of a matrix $\mathbf{X}$ as, $P_{\mathbf{X}}(\mathbf{v}) = \mathbf{X}(\arg\min_{\mathbf{w}} \|\mathbf{v} - \mathbf{X}\mathbf{w}\|_2^2)$. As a shorthand we will donte $P_{\mathbf{X}}(\mathbf{v})$ as $\mathbf{v_P}$.
Notice that

$$\begin{aligned}\|\mathbf{y}^* - \mathbf{y_P}\|_2^2 &= \|\mathbf{y}^* - \mathbf{y}_P^* + \mathbf{y}_P^* - \mathbf{y}_P\|_2^2 \\ &= \|\mathbf{y}^* - \mathbf{y}_P^*\|_2^2 + \|\mathbf{y}_P^* - \mathbf{y}_P\|_2^2\end{aligned}$$

Since the vector $\mathbf{y}^* - \mathbf{y}_P^*$ is orthogonal to the vector $\mathbf{y}_P^* - \mathbf{y}_P$ (go over discussion 3 to convince yourself), we can use the Pythagorean theorem for the last step of the equality. And since, $\mathbf{y}_P^* = \mathbf{X}\mathbf{w}^*$ and $\mathbf{y}_P = \mathbf{X}\hat{\mathbf{w}}$. Taking expectations on both sides in the above equality, we find that

$$\begin{aligned}\mathbb{E}\left[\|\mathbf{y}^* - \mathbf{X}\hat{\mathbf{w}}\|_2^2\right] &= \mathbb{E}\|\mathbf{y}^* - \mathbf{X}\mathbf{w}^*\|_2^2 + \mathbb{E}\left[\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*\|_2^2\right] \\ &= \|\mathbf{y}^* - \mathbf{X}\mathbf{w}^*\|_2^2 + \mathbb{E}\left[\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*\|_2^2\right]\end{aligned}$$

The last step makes use of the fact that $\|\mathbf{y}^* - \mathbf{X}\mathbf{w}^*\|$ is a fixed scalar. Using the previous result $\mathbb{E}[\hat{\mathbf{w}}] = \mathbf{w}^*$, we obtain that $\mathbb{E}[\mathbf{X}\hat{\mathbf{w}}] = \mathbf{X}\mathbf{w}^*$ which yields the desired result.

(b) Recall that if $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \Sigma)$, where $\Sigma \in \mathbb{R}^{(d\times d)}$ is the covariance matrix, then for any matrix $A \in \mathbb{R}^{k\times d}$, we have $\mathbf{A}\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{A}\Sigma\mathbf{A}^T)$. Use this fact to **show that** the distribution of the vector $\hat{\mathbf{w}}$ is given by

$$\hat{\mathbf{w}} \sim \mathcal{N}(\mathbf{w}^*, \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}).$$

Solution: In the previous part, we saw that

$$\hat{\mathbf{w}} = \mathbf{w}^* + \underbrace{(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top}_{=:\mathbf{A}}\mathbf{z}. \tag{10}$$

Since $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, we get that $\mathbf{Az} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{AA}^\top)$. Notice that

$$\mathbf{AA}^\top = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\left((\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\right)^\top = (\mathbf{X}^\top\mathbf{X})^{-1}.$$

And thus we have $\mathbf{Az} \sim \mathcal{N}(\mathbf{0}, \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1})$. Finally, recall that if $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \Sigma)$ and $\mathbf{c}$ is any fixed vector, then $\mathbf{v} + \mathbf{c} \sim \mathcal{N}(\mathbf{c}, \Sigma)$. Consequently, the result follows using equation (10) and the fact that $\mathbf{w}^*$ is fixed.

(c) Use part (b) to **show that**

$$\frac{1}{n}\mathbb{E}\left[\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*\|_2^2\right] = \sigma^2\frac{d}{n}.$$

Hint: The trace trick: $\mathrm{trace}(AB) = \mathrm{trace}(BA)$, might be useful.

Solution: Lets first expand $\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*\|_2^2$. We have

$$\begin{aligned}\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*\|_2^2 &= \|\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)\|_2^2 \\ &= \left(\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)\right)^\top\left((\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*)\right) \\ &= (\hat{\mathbf{w}} - \mathbf{w}^*)^\top\mathbf{X}^\top\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*).\end{aligned}$$

And thus we have

$$\mathbb{E}\left[\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*\|_2^2\right] = \mathbb{E}\left[(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\mathbf{X}^\top\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)\right].$$

For any scalar α, we can write $\alpha = \mathrm{tr}(\alpha)$. Using the identity $\mathrm{tr}(AB) = \mathrm{tr}(BA)$, we find

$$\mathbb{E}\left[\mathrm{tr}\left((\hat{\mathbf{w}} - \mathbf{w}^*)^\top\mathbf{X}^\top\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)\right)\right] = \mathbb{E}\left[\mathrm{tr}\left(\mathbf{X}^\top\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right)\right].$$

Note that for a random matrix $\mathbf{A} \in \mathbb{R}^{d\times d}$, we have

$$\mathbb{E}[\mathrm{tr}(\mathbf{A})] = \mathbb{E}[\sum_{i=1}^{d} A_{ii}] = \sum_{i=1}^{d}\mathbb{E}[A_{ii}] = \mathrm{tr}(\mathbb{E}[\mathbf{A}]).$$

This identity is a powerful one and worth remembering. Using this identity, we have

$$\mathbb{E}\left[\mathrm{tr}\left(\mathbf{X}^\top\mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right)\right] = \mathrm{tr}\left(\mathbf{X}^\top\mathbf{X}\,\mathbb{E}\left[(\hat{\mathbf{w}} - \mathbf{w}^*)(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right]\right).$$

Recall that $\mathbb{E}[\hat{\mathbf{w}}] = \mathbf{w}^*$. Hence, the matrix $\mathbb{E}\left[(\hat{\mathbf{w}} - \mathbf{w}^*)(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right]$ is precisely the covariance matrix of the random vector $\hat{\mathbf{w}}$ and is equal to $\sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}$. Using this fact, we obtain that

$$\begin{aligned}\mathrm{tr}\left(\mathbf{X}^\top\mathbf{X}\,\mathbb{E}\left[(\hat{\mathbf{w}} - \mathbf{w}^*)(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right]\right) &= \mathrm{tr}\left(\mathbf{X}^\top\mathbf{X}\sigma^2\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\right) \\ &= \sigma^2\,\mathrm{tr}(\mathbb{I}_d) \\ &= \sigma^2 d.\end{aligned}$$

Dividing by n proves the desired result.

To summarize, we have used the following sequence of equalities:

$$\begin{aligned}
\mathbb{E}\left[\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*\|_2^2\right] &= \mathbb{E}\left[(\hat{\mathbf{w}} - \mathbf{w}^*)^\top \mathbf{X}^\top \mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)\right] \\
&= \mathbb{E}\left[\operatorname{tr}\left((\hat{\mathbf{w}} - \mathbf{w}^*)^\top \mathbf{X}^\top \mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)\right)\right] \\
&= \mathbb{E}\left[\operatorname{tr}\left(\mathbf{X}^\top \mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right)\right] \\
&= \operatorname{tr}\left(\mathbb{E}\left[\mathbf{X}^\top \mathbf{X}(\hat{\mathbf{w}} - \mathbf{w}^*)(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right]\right) \\
&= \operatorname{tr}\left(\mathbf{X}^\top \mathbf{X}\mathbb{E}\left[(\hat{\mathbf{w}} - \mathbf{w}^*)(\hat{\mathbf{w}} - \mathbf{w}^*)^\top\right]\right) \\
&= \operatorname{tr}\left(\mathbf{X}^\top \mathbf{X}\sigma^2\left(\mathbf{X}^\top \mathbf{X}\right)^{-1}\right) \\
&= \sigma^2 \operatorname{tr}(\mathbb{I}_d) \\
&= \sigma^2 d.
\end{aligned}$$

(d) Assume the underlying model is a noisy linear model with scalar samples $\{\alpha_i, y_i\}_{i=1}^n$, i.e. $y_i = w_1\alpha_i + w_0 + z_i$. We construct matrix $\mathbf{X}$ by using $D+1$ polynomial features $\mathbf{P}_D(\alpha_i) = [1, \alpha_i, \ldots, \alpha_i^D]^\top$ of the *distinct* sampling points $\{\alpha_i\}_{i=1}^n$. For any $D \geq 1$, compare with model (9) and **compute $\mathbf{w}^*$ for this case. Also compute the bias ($\|\mathbf{y}^* - \mathbf{X}\mathbf{w}^*\|_2$) for this case.** Using the previous parts of this problem, **compute the number of samples n required to ensure that the average expected prediction squared error is bounded by ϵ?** Your answer should be expressed as a function of D, σ^2, and ϵ.

Conclude that as we increase model complexity, we require a proportionally larger number of samples for accurate prediction.

Solution: Notice that we can re-write the underlying model as follows:

$$\begin{aligned}
&\mathbf{y} = \mathbf{y}^* + \mathbf{z} = \mathbf{X}[w_0, w_1]^\top + \mathbf{z} \quad \text{where} \\
&\mathbf{y} = [y_1 \ldots y_n]^\top, \quad \mathbf{z} = [z_1 \ldots z_n]^\top \quad \text{and} \quad \mathbf{X} = [\mathbf{P}_1(\alpha_1) \ldots \mathbf{P}_1(\alpha_n)]^\top
\end{aligned}$$

Now observe that the true signal is indeed linear and hence we only need linear features. Consequently, we have that for $D = 1$: $\mathbf{w}^* = [w_0, w_1]^\top$. Extending for $D \geq 1$, we conclude that $\mathbf{w}^* = [w_0, w_1, \mathbf{0}]^\top$, where $\mathbf{0} \in \mathbb{R}^{D-1}$.

From the above, it is clear that $\|\mathbf{y}^* - \mathbf{X}\mathbf{w}^*\|_2 = 0$, i.e. the bias is 0.

Now, lets compute the number of samples n required to ensure that prediction error is bounded by ϵ. For a polynomial regression model, the matrix $\mathbf{X}$ has full column rank when the points are distinct (refer to HW2), hence $d = D + 1$. From part (a) we know that the error is: $\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}^*\|_2^2 = \|\mathbf{y}^* - \mathbf{X}\mathbf{w}^*\|_2^2 + \|\mathbf{X}\mathbf{w}^* - \mathbf{X}\hat{\mathbf{w}}\|_2^2$, and using part (d) and the fact that the bias is 0, we get that:

$$\frac{1}{n}\mathbb{E}\left[\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}^*\|_2^2\right] = \sigma^2 \frac{D+1}{n}$$

and in order to upper bound the error with ϵ it suffices to have $n > \frac{\sigma^2(D+1)}{\epsilon}$.

Clearly, the number of samples required for error ϵ increases linearly with the degree D and with the variance σ^2.

(e) Simulate the problem from part (d) for yourself. Set $w_1 = 1$, $w_0 = 1$, and sample n points $\{\alpha_i\}_{i=1}^n$ uniformly from the interval $[-1, 1]$. Generate $y_i = w_1\alpha_i + w_0 + z_i$ with z_i representing standard Gaussian noise.

Fit a D degree polynomial to this data and show how the average error $\frac{1}{n}\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}^*\|_2^2$ scales as a function of both D and n.

You may show separate plots for the two scalings. It may also be helpful to average over multiple realizations of the noise (or to plot point clouds) so that you obtain smooth curves.

(For this part, the libraries numpy.random and numpy.polyfit might be useful. You are free to use any and all python libraries.)

Solution: In the Figure 4 below, we plotted the log error vs degree of the polynomial and log of error vs the sample size for better understanding. As we can see the error increases with D and decreases with n.

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

deg = 19 #degree of the polynomial
step = 80
n_s = 40 #sample size
w_1 = 1
w_0 = 1
error = np.zeros((deg, step)) # defining error as a function of d and n
alpha = np.random.uniform(-1, 1, n_s) #generating alpha from unifrom dist with sample size
    ↪ n_s
z = np.random.normal(0, 1, n_s) # generating i.i.d noise with sample size n_s
y_s = w_1*alpha+w_0 # this is Y*
y = y_s+z
for N in range(step): # number of steps
    n = N+n_s # sample size at each step
    for d in range(1, deg+1): #degree of polynomial from 1 to deg
        w_hat = np.polyfit(alpha, y, d) #fitting degree d polynomial
        E = 0 #initializing error for this fit with degree=d and sample size=n
        for i in range(n):
            E = E + (np.polyval(w_hat, alpha[i])-y_s[i])**2 # adding error of each sample to
    ↪ total error
        error[d-1][N]=E/n # normalize the error
    alpha_new = np.random.uniform(-1, 1, 1) # generating a new sample for the next step to
    ↪ increase the sample size by 1
    z_new = np.random.normal(0, 1, 1) # generating noise for the new sample for the next step
    ↪  to increase the sample size by 1
    alpha = np.append(alpha, alpha_new) # adding the new sample to the sample array
    z = np.append(z, z_new) # adding the new noise value to the noise array
    y_s = w_1*alpha+w_0 # computing y*
    y = y_s+z # computing y

plt.figure()
plt.subplot(121)
plt.semilogy(np.arange(1, deg+1),error[:,-1])
plt.xlabel('degree of polynomial')
plt.ylabel('log of error')
plt.subplot(122)
plt.semilogy(np.arange(n_s, n_s+step), error[-1,:])
plt.xlabel('number of samples')
plt.ylabel('log of error')
```

```
plt.show()
```

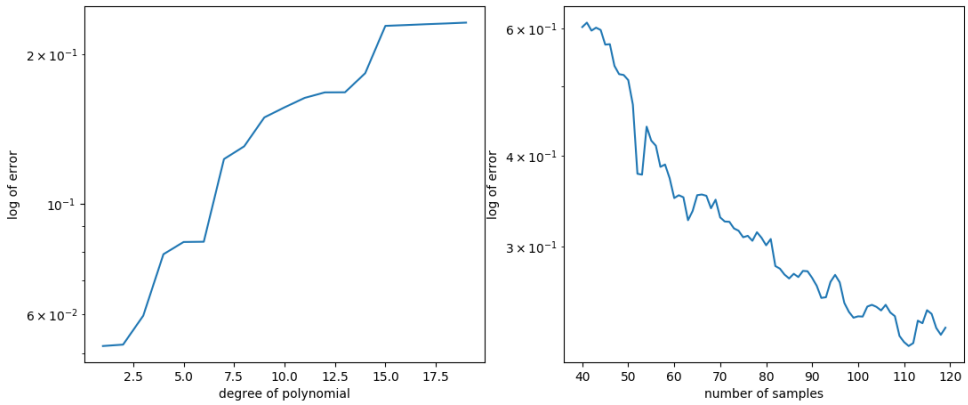


Figure 4: Average error vs D and n

(f) Assume that the underlying model is the noisy exponential function with scalar samples $\{\alpha_i, y_i\}_{i=1}^n$ where $y_i = e^{\alpha_i} + z_i$ with *distinct* sampling points $\{\alpha_i\}_{i=1}^n$, in the interval $[-4, 3]$ and i.i.d. Gaussian noise $z_i \sim \mathcal{N}(0, 1)$. We again construct matrix $\mathbf{X}$ by using $D + 1$ polynomial features $[1, \alpha_i, \ldots, \alpha_i^D]^\top$ and use linear regression to fit the observations. Recall, the definitions of the bias and variance of the OLS estimator from part (a) and **show that for a fixed n, as the degree D of the polynomial increases: (1) the bias decreases and (2) the variance increases**. Use the values you derived for bias and variance and **show that for the prediction error, the optimal choice of D is given by $\mathcal{O}(\log n / \log\log n)$.**

Hint: You can directly use previous parts of this problem, Discussion 3 and Problem 4 of HW 2. You may assume that n and D are large for approximation purposes.

Solution: Using part (a), we split the prediction error in two terms: bias and variance. We first compute the bias (approximation error).

We use HW 2 Problem 4 now. For the approximation error, we use the Dth order Taylor series approximation of the function $f : \alpha \mapsto e^\alpha$ about the point 0, which is given (for some $\alpha' \in [0, \alpha]$) by

$$e^\alpha = \underbrace{\sum_{i=1,3,..}^{D} (-1)^{(i-1)/2} \frac{\alpha^i}{i!}}_{\phi_D(\alpha)} + f^{D+1}(\zeta) \frac{\alpha'^{D+1}}{(D+1)!}.$$

We derived the following bound in the last homework:

$$||f - \phi_D||_\infty = \sup_{\alpha \in [-3,4]} |e^\alpha - \phi_D(\alpha)| \leq \frac{e^3 4^{D+1}}{(D+1)!}.$$

Consequently, the approximation error for the sample $\{\alpha_i, y_i^*\}$ is bounded by $\frac{e^3 4^{D+1}}{(D+1)!}$. Summing and dividing through by n yields that

$$\text{Bias}^2 = \text{Approximation error} = \frac{1}{n}\|\mathbf{y}^* - \mathbf{X}\mathbf{w}^*\|^2 \leq \left(\frac{e^3 4^{D+1}}{(D+1)!}.\right)^2$$

Now using part (c) and the facts that the matrix $\mathbf{X}$ is of size $n \times (D+1)$ and $\sigma = 1$, we obtain that

$$\text{Variance} = \text{Estimation error} := \frac{1}{n}\mathbb{E}\left[\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{X}\mathbf{w}^*\|_2^2\right] = \frac{D+1}{n}.$$

To optimally trade-off bias and variance, we set them to be equal. This provides the right scaling (upto constants). Think about why? Hint: Approximation $x + y \approx \max\{x, y\}$ for $x, y > 0$ will be useful. Using Stirling's approximation, ignoring constants and making a few approximations we find that

$$\begin{aligned}
\frac{(4e)^{D+1}}{(D+1)^{D+1}} &\approx \sqrt{\frac{D}{n}} \\
\frac{(4e)^{D+1}}{D^D} &\approx \sqrt{\frac{D}{n}} \\
\sqrt{n} &\approx \left(\frac{D}{4e}\right)^{D+1} \\
\sqrt{n} &\approx \left(\frac{D}{4e}\right)^{D} \\
\log n &\approx D \log\left(\frac{D}{4e}\right).
\end{aligned}$$

Now we verify if the approximation is valid for $D = \log n / \log\log n$:

$$\begin{aligned}
D \log\left(\frac{D}{4e}\right) &\approx D \log D \\
&= \frac{\log n}{\log\log n} \cdot \log\left(\frac{\log n}{\log\log n}\right) \\
&= \frac{\log n}{\log\log n} \cdot (\log\log n - \log\log\log n) \\
&\approx \log n,
\end{aligned}$$

and hence we are done.

(g) Simulate the problem in part (f) yourself. Sample n points $\{\alpha_i\}_{i=1}^n$ uniformly from the interval $[-4, 3]$. Generate $y_i = e^{\alpha_i} + z_i$ with z_i representing standard Gaussian noise. **Fit a D degree polynomial to this data and show how the average error $\frac{1}{n}\|\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}^*\|_2^2$ scales as a function of both D and n.** You may show separate plots for the two scalings. For scaling with D,

choose $n = 120$. It may also be helpful to average over multiple realizations of the noise (or to plot point clouds) so that you obtain smooth curves. (For this part, the libraries numpy.random and numpy.polyfit might be useful and you are free to use any and all python libraries.)

Solution: In the Figure 5 below, we plotted the log of error vs degree of the polynomial and log of error vs the sample size for better understanding.

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

deg = 12 #degree of the polynomial
step = 100
n_s = 20 #intial sample size
error = np.zeros((deg, step)) # defining error as a function of d and n
alpha = np.random.uniform(-4, 3, n_s) #generating alpha from unifrom dist with sample size
    ↪ n_s
z = np.random.normal(0, 1, n_s) # generating i.i.d noise with sample size n_s
y_s = np.exp(alpha) # this is y*
y = y_s+z
for N in range(step): # number of steps
    n = N+n_s # sample size at each step
    for d in range(1, deg+1): #degree of polynomial from 1 to deg
        w_hat = np.polyfit(alpha, y, d) #fitting degree d polynomial
        E = 0 #initializing error for this fit with degree=d and sample size=n
        for i in range(n):
            E = E + (np.polyval(w_hat, alpha[i])-y_s[i])**2 # adding error of each sample to
    ↪ total error
        error[d-1][N]=E/n # normalize the error
    alpha_new = np.random.uniform(-1, 1, 1) # generating a new sample for the next step to
    ↪ increase the sample size by 1
    z_new = np.random.normal(0, 1, 1) # generating noise for the new sample for the next step
    ↪  to increase the sample size by 1
    alpha = np.append(alpha, alpha_new) # adding the new sample to the sample array
    z = np.append(z, z_new) # adding the new noise value to the noise array
    y_s = np.exp(alpha) # computing y*
    y = y_s+z # computing y

plt.figure()
plt.subplot(121)
plt.semilogy(np.arange(1, deg+1),error[:,-1])
plt.xlabel('degree of polynomial')
plt.ylabel('log of error')
plt.subplot(122)
plt.semilogy(np.arange(n_s, n_s+step), error[5,:])
plt.xlabel('number of samples')
plt.ylabel('log of error')
plt.show()
```

(h) Comment on the differences in plots obtained in part (e) and part (g).

Solution:

In part (e), the true model was a linear model and therefore polynomial feature based OLS estimate was unbiased, i.e., we had no bias or approximation error. The variance or estimation error scales linearly with the degree of the polynomial and hence we saw in the graph that the prediction error increases monotonically with D.

For part (g), the true signal was an exponential function which can not be represented exactly by polynomials. As a result, we have an approximation error which however decreases with increase in the degree of the polynomial (as we had explored in the previous homework).

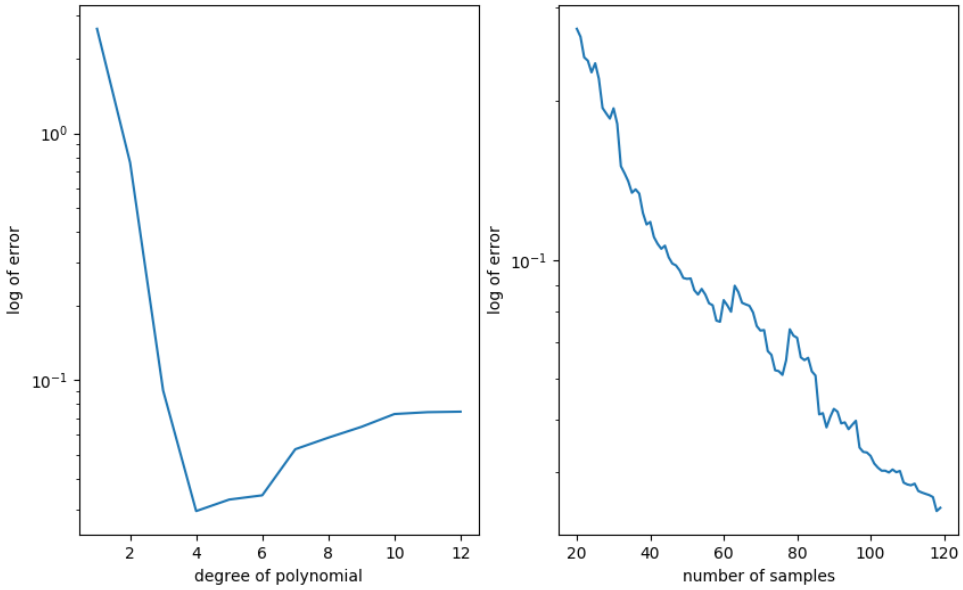


Figure 5: Average error vs D and n

However, as the model complexity (which in our case is equal to the degree of the polynomial) increases, the variance of the estimator also increases. As a result we see a trade-off between the bias and the variance. For small D, the bias is large and for large D, the variance is too big. Consequently, we see a sweet spot D^* at which the two terms are of the same order.

More generally, the approximation error measures how well our regression model fits the true data. Since we can better approximate the unknown function with higher order polynomials, this term decreases with D. However, it becomes harder to estimate the correct polynomial, since we now give ourselves extra freedom. This phenomenon is the bias variance trade-off in action!

As far as the scaling with respect to the number of samples n goes, we see that bias is unaffected by the number of samples. The variance of the estimator should only decrease with the number of samples and hence we see a decrease in prediction error with increase in number of samples.

Note that in the limit of infinite samples, the variance vanishes away and the prediction error will be equal to the bias term!

4 Nonlinear Classification Boundaries

Make sure to submit the code you write in this problem to "HW3 Code" on Gradescope.

In this problem we will learn how to use polynomial features to learn nonlinear classification boundaries.

In Problem 5 on HW1, we found that linear regression can be quite effective for classification. We

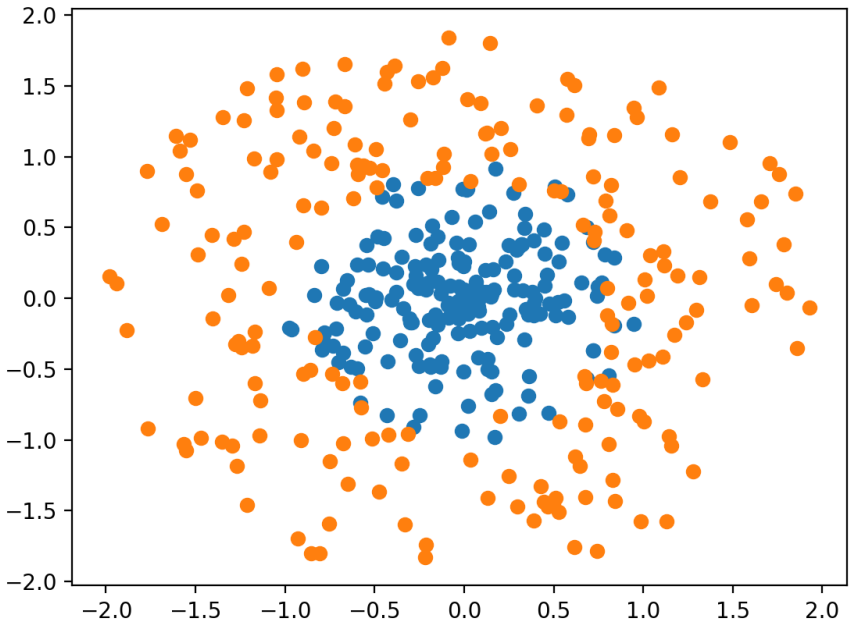
applied it in the setting where the training data points were *approximately linearly separable*. This means there exists a hyperplane such that most of the training data points in the first class are on one side of the hyperplane and most training data points in the second class are on the other side of the hyperplane.

However, often times in practice classification datasets are not linearly separable. In this case we can create features that are linearly separable by augmenting the original data with polynomial features as seen in Problem 5. This embeds the data points into a higher dimensional space where they are more likely to be linearly separable.

In this problem we consider a simple dataset of points $(x_i, y_i) \in \mathbb{R}^2$, each associated with a label b_i which is -1 or $+1$. The dataset was generated by sampling data points with label -1 from a disk of radius 1.0 and data points with label $+1$ from a ring with inner radius 0.8 and outer radius 2.0.

(a) **Run the starter code to load and visualize the dataset and submit a scatterplot of the points with your homework. Why can't these points be classified with a linear classification boundary?**

Solution:



If we want to find a hyperplane that separates these points approximately, one of the half-spaces must contain most of the points from the outer ring (because they form a class). However due to the shape of the dataset, this half space will also contain most of the points in the inner circle and therefore the points cannot be classified with a linear classification boundary.

(b) Classify the points with the technique from Problem 5 on HW1 ("A Simple classification approach"). Use the feature matrix $\mathbf{X}$ whose first column consists of the x-coordinates of

the training points and whose second column consists of the y-coordinates of the training points. The target vector $\mathbf{b}$ consists of the class label -1 or $+1$. Perform the linear regression $\mathbf{w}_1 = \arg\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{b}\|_2^2$. **Report the classification accuracy on the test set.**

Solution: The accuracy can be computed with the following code:

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

n = 400

X = np.load("Xtrain.npy")
y = np.load("ytrain.npy")

def visualize_dataset(X, y):
    plt.scatter(X[y < 0.0, 0], X[y < 0.0, 1])
    plt.scatter(X[y > 0.0, 0], X[y > 0.0, 1])
    plt.show()

# TODO: visualize the dataset
visualize_dataset(X, y)

w1 = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, y))

Xtest = np.load("Xtest.npy")
ytest = np.load("ytest.npy")

ypred = -1.0*(Xtest.dot(w1) < 0.0) + 1.0*(Xtest.dot(w1) > 0.0)

print("prediction accuracy on the test set is ", 1.0*sum(ypred == ytest) / n)
```

The accuracy on the test set is 0.5075, so the classifier only gets about half of the predictions right. This is because the data is not linearly separable as seen in (a).

(c) Now augment the data matrix $\mathbf{X}$ with polynomial features $1, x^2, xy, y^2$ and classify the points again, i.e. create a new feature matrix

$$\Phi = \begin{pmatrix} x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & 1 \\ x_2 & x_2 & x_2^2 & x_2 y_2 & y_2^2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_n & y_n & x_n^2 & x_n y_n & y_n^2 & 1 \end{pmatrix}$$

and perform the linear regression $\mathbf{w}_2 = \arg\min_{\mathbf{w}} \|\Phi\mathbf{w} - \mathbf{b}\|_2^2$. **Report the classification accuracy on the test set.**

Solution: Continuing from the code in the last part, the accuracy for the augmented data can be computed in the following way:

```
Phi = np.vstack([X[:,0], X[:,1], X[:,0]**2, X[:,0] * X[:,1], X[:,1]**2, 1.0 + 0.0*X[:,1]]).T

w2 = np.linalg.solve(np.dot(Phi.T, Phi), np.dot(Phi.T, y))

Phitest = np.vstack([Xtest[:,0], Xtest[:,1], Xtest[:,0]**2, Xtest[:,0] * Xtest[:,1], Xtest
    ↪ [:,1]**2, 1.0 + 0.0*Xtest[:,1]]).T

ypred2 = -1.0*(Phitest.dot(w2) < 0.0) + 1.0*(Phitest.dot(w2) > 0.0)

print("prediction accuracy on the test set is ", 1.0*sum(ypred2 == ytest) / n)

print("w2 = ", w2)
```

The accuracy on the test set is now 0.89 which shows that polynomial features are able to classify most of the points in the test set correctly.

(d) **Report the weight vector that was found in the feature space with the polynomial features. Show that up to small error the classification rule has the form $\alpha x^2 + \alpha y^2 \leq \beta$. What is the interpretation of β/α here? Why did the classification in the augmented space work?**

Solution: The weight vector is

```
w2 =  array([ 0.08278755,  0.04680803,  0.65422045,
             -0.03610703,  0.71621139, -0.81494877])
```

so we have $\alpha \approx 0.7$ and $\beta \approx 0.8$. The interpretation of β/α is that $r^2 = \beta/\alpha$ is about the radius of the inner disk squared, so $r \approx 1$ which is the approximate radius of the inner disk. The classification in the polynomial feature space worked, because the polynomial features are rich enough to encode the exact classification rule ($x^2 + y^2 \leq r^2$).

Later on in the course, when we have access to nonlinear optimization techniques, we will see how the same ideas of using features can be combined with better loss functions for classification. But this problem should show you the power of features in classification problems as well as in regression ones.

5 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.