EECS 189 Introduction to Machine Learning
Fall 2020 HW2

This homework is due **Tuesday, September 15 at 11:59 p.m.**

1 Getting Started

Read through this page carefully. You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Deliverables:

1. Submit a PDF of your writeup, **with an appendix for your code**, to the appropriate assignment on Gradescope. If there are graphs, include those graphs in the correct sections. Do not simply reference your appendix.
2. If there is code, submit all code needed to reproduce your results.
3. If there is a test set, submit your test set evaluation results.

After you've submitted your homework, watch out for the self-grade form.

(a) Who else did you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

(b) Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadvertently cheats.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions nor have I looked at any online solutions to any of these problems. I have credited all external sources in this write up.

2 Nonlinear Classification Boundaries

Make sure to submit the code you write in this problem to "HW2 Code" on Gradescope.

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

In Problem "A Simple Classification Approach" 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*. The term "linearly separable" in classification 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 their native representation. In such cases we can often find a representation that is linearly separable by augmenting the original data with appropriate features. Sometimes, the polynomial features that we saw in Problem "Learning 1-D functions" of HW1 are good enough to let us do this, because after all, they are from a universal feature family. This "lifting" 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 binary 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 in the x,y plane?**

(b) Classify the points with the technique from Problem 7 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.**

(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

$$\Phi = \begin{pmatrix} x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 & 1 \\ x_2 & x_2 & x_2^2 & x_2y_2 & y_2^2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_n & y_n & x_n^2 & x_ny_n & y_n^2 & 1 \end{pmatrix}$$

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.**

(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?**

3 Blair and their giant peaches

Make sure to submit the code you write in this problem to "HW2 Code" on Gradescope.

Blair is a mage testing how long they can fly a collection of giant peaches. They has n training peaches – with masses given by $x_1, x_2, \ldots x_n$ – and flies these peaches once to collect training data. The experimental flight time of peach i is given by y_i. They believes that the flight time is well approximated by a polynomial function of the mass

$$y_i \approx w_0 + w_1x_i + w_2x_i^2 \cdots + w_Dx_i^D$$

where their goal is to fit a polynomial of degree D to this data. Include all text responses and plots in your write-up.

(a) **Show how Blair's problem can be formulated as a linear regression problem.**

(b) You are given data of the masses $\{x_i\}_{i=1}^n$ and flying times $\{y_i\}_{i=1}^n$ in the "x_train" and "y_train" keys of the file `1D_poly.mat` with the masses centered and normalized to lie in the range $[-1, 1]$. **Fill missing part in part (b) of the iPython notebook to do a least-squares fit (taking care to include a constant term) of a polynomial function of degree D to the data.** Letting f_D denote the fitted polynomial, **plot the average training error $R(D) = \frac{1}{n}\sum_{i=1}^n (y_i - f_D(x_i))^2$ against D in the range $D \in \{0, 1, 2, 3, \ldots, n-1\}$.** You may not use any library other than `numpy` and `numpy.linalg` for computation.

(c) **How does the average training error behave as a function of D, and why? What happens if you try to fit a polynomial of degree n with a standard matrix inversion method?**

(d) Blair has taken CS189 so decides that they needs to run another experiment before deciding that their prediction is true. They run another fresh experiment of flight times using the same peaches, to obtain the data with key "y_fresh" in 1D_POLY.MAT. Denoting the fresh flight time of peach i by $\tilde{y}_i$, **by completing part (d) of the iPython notebook, plot the average error $\tilde{R}(D) = \frac{1}{n}\sum_{i=1}^n (\tilde{y}_i - f_D(x_i))^2$ for the same values of D as in part (b) using the polynomial approximations f_D also from the previous part. How does this plot differ from the plot in (b) and why?**

(e) **How do you propose using the two plots from parts (b) and (d) to "select" the right polynomial model for Blair?**

(f) Blair has a new hypothesis – the flying time is actually a function of the mass, smoothness, size, and sweetness of the peach, and some multivariate polynomial function of all of these parameters. A D-multivariate polynomial function looks like

$$f_D(\mathbf{x}) = \sum_j \alpha_j \prod_i x_i^{p_{ji}},$$

where $\forall j : \sum_i p_{ji} \leq D$. Here α_j is the scale constant for jth term and p_{ji} is the exponent of x_i in jth term. The data in `polynomial_regression_samples.mat` (100000×5) with columns corresponding to the 5 attributes of the peach. **Use 4-fold cross-validation to decide which of $D \in \{0, 1, 2, 3, 4, 5\}$ is the best fit for the data provided**. For this part, compute the polynomial coefficients via ridge regression with penalty $\lambda = 0.1$, instead of ordinary least squares. You are not allowed to use any library other than `numpy` and `numpy.linalg`. Write your implementation by completing the part (g) of the iPython notebook.

(g) Now **redo the previous part, but use 4-fold cross-validation on all combinations of $D \in \{1, 2, 3, 4, 5\}$ and $\lambda \in \{0.05, 0.1, 0.15, 0.2\}$** - this is referred to as a grid search. **Find the best D and λ that best explains the data using ridge regression. Print the average training/validation error per sample for all D and λ.** Again, write your implementation by completing the part (g) of the iPython notebook.

4 Outlier Removal via OMP (Part 2)

In the previous homework (and in 16B), we looked at how the orthogonal matching pursuit (OMP) can be used to remove an effect of outliers in the training data. This problem can be viewed as a continuation of that problem that forms a natural second part. We will look into why the concept of validation is needed here as well as how to use a validation set that may itself be corrupted or contains outliers. We will assume the same setting as the first part. To summarize, we want to perform linear regression on training data that contains an unknown fraction of outliers. Part (a) - (c) involve coding in a Jupyter notebook and some short answers. Please read more detailed instructions in the notebook itself.

Make sure to submit the code you write in this problem to "HW2 Code" on Gradescope.

(a) From the training error plot, we will see that the training error always decreases as the number of non-zeros in the weight or the number of the removed outliers increases. However, this does not reflect the true error in the recovered pattern. For example, if we had a clean or uncorrupted test set, we could get an idea of what that true error is like. **Fill in the missing code to plot the test error using the weights obtained from the OMP outlier-removal method. Then describe in words the trends of the test error you see, and based on the plot alone, tell us how many outliers should be removed by OMP?**

(b) In the real world, we often cannot evaluate our models on a clean test set so we will need to carry out the validation process based on the data we have, i.e. potentially corrupted data. It is no longer safe to compute the mean of the error as before. Computing the mean error on the corrupted validation set will result in an unreliable representation of the errors on uncorrupted data since that mean will be heavily skewed by the outliers.

Luckily, there are multiple ways of measuring the central tendency, other than the *mean* value. In fact, *median* is another option which is known to be robust to outliers. To see how median is more robust compared to mean, **compute the validation errors based on the weights at the specified indices. For each of the weights, plot a histogram of the validation errors and compute their mean and median.** See where the values of mean and median fall in each histogram.

(c) Does the idea of median being a robust estimator give you some idea for how to do validation in the presence of outliers? **Plot mean and median of the validation errors against the number of the non-zero entires in OMP solutions (all the weights we obtained from part (a)). Now determine from the median plot the number of the outliers. Why does the median work better than the mean in this case?**

Now that we have seen how useful the median is against corrupted data and outliers, we will do some theoretical analysis on median. First, we'll look its convergence and second, we'll verify its robustness.

(d) From the law of large numbers, we have seen that with a large number of samples, the sample mean converges to the population mean or expected value. More rigorously, the weak law of large numbers states the following: For any positive number ε,

$$\lim_{n\to\infty} \mathbb{P}\left(\left|\overline{X}_n - \mu\right| > \varepsilon\right) = 0$$

where μ is the expectation, and $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ is the sample mean. Here, we would like to make a similar statement for sample median and population median. Given a sequence of n random variables i.i.d. drawn from the same distribution, $\{X_1, X_2, ..., X_n\}$, let's denote the population median as $med(X)$ and the sample median as $\tilde{X}_n$. We want to make no other assumption on the distribution of X_i's. The goal is to give a proof of the following statement:

$$\lim_{n\to\infty} \mathbb{P}\left(\left|\tilde{X}_n - med(X)\right| > \epsilon\right) = 0$$

But to make the proof easier to follow and to understand things in terms of their natural dependencies, we will modify the above statement to involve *quantiles* of X. For every $\varepsilon > 0$ for which the $(\frac{1}{2} - \varepsilon)$ quantile is different from the median and the $(\frac{1}{2} + \varepsilon)$ quantile is also different from the median, we have:

$$\lim_{n\to\infty} \mathbb{P}\left(\tilde{X}_n < (1/2 - \varepsilon)\text{-quantile or } \tilde{X}_n > (1/2 + \varepsilon)\text{-quantile}\right) = 0.$$

Here, (for simplicity) a p-quantile of a random variable is a value x for which the CDF $\mathbb{P}(X \leq x) = p$. [To be precise, a p-quantile is an x for which $\mathbb{P}(X < x) \leq p$ and $\mathbb{P}(X \leq x) \geq p$. This allows the distribution of X to have atoms in it and for quantiles to still be defined in a reasonable manner.]

Notice that by choosing an appropriate value of ε, we can recover the desired ϵ, and hence, the two statements are equivalent.

(First hint: Consider a binomial random variable: $Y_i = \mathbb{1}\{X_i > (1/2 + \varepsilon)\text{-quantile}\}$)

(Second hint: Think about a relevant Chernoff bound and use it. You don't have to use a Chernoff bound to prove it, but it helps in understanding the speed of this convergence.)

(e) For this part, we want to also show that median has a robustness property against a strong adversary who can completely change portions of the data. We will use a similar setting as in the previous part but asssume that the samples are fixed. In other words, we will compute the median of n samples $\{x_1, x_2, ..., x_n\}$, but there is now an adversary who can completely change γn samples or an γ portion where $0 < \gamma < 1/2$. The goal is to **find the largest deviation from the true median of the n samples that the adversary can cause** as a function of γ. Your answer should involve quantiles.

This should be more straightforward than the previous part.

(Hint: Suppose that $n = 100$ and the samples were $1, 2, 3, \ldots, 99, 100$. Suppose that the adversary wanted to make your median be as small as possible and was allowed to change ten values. Which ten would the adversary change and what would they change them to? What would the resulting median be of the modified data? Working out this example should help you understand the general case.)

5 Simple Regularization Tradeoff, with a Final Twist

In an earlier HW, you saw how ridge-regression can be reconceptualized as adding fake data points at zero and just doing ordinary least-squares on the augmented data set. Because ordinary least-squares can be viewed as a generalization of the simple act of taking an average, it is worth stepping back and exploring regularization in this extremely simple context.

The advantage of the extremely simple context is that it lets us easily use probability-based models to sharpen our understanding of what is going on, and to understand what the in-principle best amount of regularization should be.

Consider a random variable X, which has unknown mean μ and unknown variance σ^2. Given n iid realizations of training samples $X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n$ from the random variable, we wish to have the best constant predictor for new realizations from this distribution. If we had access to the distribution of X, the mean μ would be the best constant predictor in the squared error sense since the mean μ minimizes $E[(X-\mu)^2]$.

However, we don't have access to the distribution of X and only have data. We will call our predictor for X the random variable $\hat{X}$, which will have its own mean $\hat{\mu}$. There are a few ways we can estimate μ given the realizations of the n samples:

1. Average the n samples: $\frac{x_1+x_2+\cdots+x_n}{n}$.
2. Average the n samples and one sample of 0: $\frac{x_1+x_2+\cdots+x_n}{n+1}$.
3. Average the n samples and n_0 samples of 0: $\frac{x_1+x_2+\cdots+x_n}{n+n_0}$.
4. Ignore the samples: just return 0.

In the parts of this question, we will measure the *bias* and *variance* of each of our estimators. The *bias* is defined as $E[\hat{X}-\mu]$ and the *variance* is defined as $\text{Var}[\hat{X}]$. Notice that both of these do not depend on the actual realizations of the n samples.

(a) **What is the bias of each of the four estimators above?**

(b) **What is the variance of each of the four estimators above?**

(c) Suppose we have constructed an estimator $\hat{X}$ from some samples of X. We now want to know how well $\hat{X}$ estimates a fresh (new) sample of X. Denote this fresh sample by X'. Note that X' is an i.i.d. copy of the random variable X. **Derive a general expression for the expected squared error $E[(\hat{X}-X')^2]$ in terms of σ^2 and the bias and variance of the estimator $\hat{X}$. Similarly, derive an expression for the expected squared error $E[(\hat{X}-\mu)^2]$. Compare the two expressions and comment on the differences between them, if any.**

(d) For the following parts, we will refer to expected total error as $E[(\hat{X}-\mu)^2]$. It is a common mistake to assume that an unbiased estimator is always "best." Let's explore this a bit further. **Compute the expected squared error for each of the estimators above.**

(e) **Demonstrate that the four estimators are each just special cases of the third estimator, but with different instantiations of the hyperparameter n_0.**

(f) **What happens to bias as n_0 increases? What happens to variance as n_0 increases?**

(g) Say that $n_0 = \alpha n$. **Find the setting for α that would minimize the expected total error, assuming you secretly knew μ and σ.** Your answer will depend on σ, μ, and n.

In other words, how many fake 0s should we add to our data set?

(h) **What happens to the optimal α in the previous part as you get more samples and n gets bigger?** What does this mean in terms of the number n_0 of fake data points we should add?

(i) For this part, let's assume that we had some reason to believe that μ *should be small* (close to 0) and σ *should be large*. In this case, **what happens to the expression for the optimal α that you computed in a previous part?**

(j) Draw a connection between α in this problem and the regularization parameter λ in the ridge-regression version of least-squares.

What is the optimal value for λ as a function of n, σ, μ?

(k) Now, we can use this setup to take a more skeptical look at whether a simple validation approach — taking some extra n_v validation points and using them to tune the hyperparameter of interest — can work reasonably in this particular setting.

Suppose we had n_v validation points $x_{n+1}, \dots, x_{n+n_v}$ that were all drawn i.i.d. according to the distribution for X. **Describe what would happen if you used these validation points to pick the hyperparameter α (you can also answer in terms of λ if you wish) that minimizes the validation error and comment on whether the resulting estimator is good or not.**

(HINT: Try this out and see what happens. If it helps, suppose that some adversary just made your validation points come from a different distribution. How much can the adversary impact you?)

6 Hyperparameter Selection for Ridge Regression

In the previous question, we understood the case of learning a constant predictor. In this problem, we will increase the complexity of the model that we are learning so that we can better understand the role of hyperparameter selection.

(a) For the first part of this problem, we will focus on the special case when $d = 1$, working with the observation model:

$$\mathbf{y} = \mathbf{x}w + \boldsymbol{\epsilon}$$

where w is the true parameter.

Assume that we have n observations, each with $\mathbf{x}_i$ of dimension $d = 1$, so $\mathbf{x}$ is an $n \times d$ matrix, just a column. $\boldsymbol{\epsilon}$ is some random source of error in our observations. For this problem, we will model each component of the error as an independent Gaussian random variable, such that

$$\epsilon_i \sim N(0, \sigma^2).$$

Let $\hat{w}(\lambda)$ be the ridge-regression estimate. If w is the true parameter value that generated the data, **give an expression for $\mathbb{E}[\hat{w}(\lambda) - w)^2]$ and an expression for the optimal choice of λ as a possible function of $n, \mathbf{x}, w, \sigma^2$.**

Of course, in many machine-learning contexts, you don't already know w or σ^2. This part is just asking you to understand how the in-principle optimal value of λ depends on these parameters.

(b) Now, we will consider the case when $d > 1$, so each of our $\mathbf{x}_i$ have d components, not scalars. Similarly, the true weight vector $\mathbf{w}$ has d components. The observation noise model is the same as the previous part.

To make the calculations more tractable, we will consider the special case when all the columns of our training data matrix $\mathbf{X}$ are orthonormal, so $\mathbf{X}^T\mathbf{X} = \mathbf{I}$. (For example, from the previous homework, you should see how this situation can arise if the features are Fourier features and the training samples are taken in a regularly spaced manner.)

We are going to keep to a single scalar hyperparameter λ and must find a single λ which does a good job of balancing the needs of the different components of $\mathbf{w}$.

Solve for the scalar value of λ that minimizes $\mathbb{E}[(\hat{\mathbf{w}}(\lambda) - \mathbf{w})^2]$ exactly, as a function of n, $\mathbf{w}$, λ, and σ^2.

(c) In the cases considered in the previous parts, we were able to analytically compute the optimal value of λ. Can we use that closed form expression in practice? Unfortunately not, as it depends on $\mathbf{w}$, which is of course exactly the quantity that we are trying to estimate. Instead, in the real-world, we have to conduct *hyperparameter tuning* to pick a reasonable λ in a data driven way. Essentially, we split our dataset into two groups: training and validation. For various values of λ, we train our model on the training set, and observe its performance on the validation set. Then, we choose the value of λ that minimizes the error on the validation set, and use that λ. How do we use that λ? For example, we can use it to train our model on the entire input dataset.

Using this method, we'd expect the optimized value of λ to approximately match the predictions from the previous parts.

Fill in the blanks in the Jupyter notebook to compute the optimal value λ_{opt} for a toy model, then fill in the blanks to implement hyperparameter tuning, and compute the tuned value of λ_{tuned}. Report both values here for some σ and validation fraction, and compare them. Also report the σ and validation fraction you chose.

[*Hint: You should expect $\lambda_{opt} \approx \lambda_{tuned}$. If they differ, you have done something wrong, so you can use this part to verify your answers to the previous parts.*]

7 Ridge Regression Through Feature Augmentation

In Homework 1, you showed that ridge regression can be accomplished through augmenting the feature space of a least-squares problem with an appropriately scaled identity. In this problem, you will see this equivalence on a real dataset, and practice predicting with a model trained using such augmented features.

Before beginning the body of this problem, you should **copy over helper functions from Problem 7.**

(a) First, you will perform hyperparameter tuning and explicit ridge regression to establish a baseline for later parts. **Report the tuned λ you obtain through validation here, and your final MSE on the training data**.

(b) Next, augment the feature space of the training data with an appropriate identity and use the minimum-norm least-squares solution to find $\hat{\mathbf{w}}$. You should find that you reach an identical parameter vector, but a different training MSE. **Explain why the training MSE is different when solving with this method of regularization.**

(c) Now that we have shown the equivalence of our two regression methods, practice predicting on the test set by extracting $\hat{\mathbf{w}}$ from $\boldsymbol{\eta}$ and by appropriately augmenting the test data. **Report what you augmented the test set with.** You should see different training MSEs but identical test MSEs with the two methods.

One of the important points of this question is to help you understand what the concept of training MSE is, and its dependence on context.

(d) Finally, try augmenting the test set with various scaled random variables on the diagonal. Try different distributions and scalings **describe how the augmented features impact the test performance as you vary the scale and distribution.**

This part of this problem is somewhat open ended, but your explorations here are setting the stage for a problem that will come in a later homework that connects to a very important issue in modern machine learning.

8 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.

Contributors:

- Alexander Tsigler
- Anant Sahai
- Ashwin Pananjady
- Chawin Sitawarin
- Josh Sanz
- Lydia T. Liu
- Peter Wang
- Philipp Moritz
- Rahul Arya
- Yichao Zhou