Machine Learning: Homework 2

Instructions

- **Collaboration policy:** Homeworks must be done individually, except where otherwise noted in the assignments. "Individually" means each student must hand in their own answers, and each student must write and use their own code in the programming parts of the assignment. It is acceptable for students to collaborate in figuring out answers and to help each other solve the problems, though you must in the end write up your own solutions individually, and you must list the names of students you discussed this with. We will be assuming that, as participants in an undergraduate course, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration.
- **Online submission:** You must submit your solutions online on NYU Classes. You need to submit (1) a PDF which contains the solutions to all questions (2) `x.py` or `x.ipynb` files for the programming questions. We recommend that you use LaTeX, but we will accept scanned / pictured solutions as well.

Problem 1: Linear Regression

In linear regression model, we assume $y = Xw + \epsilon$, where ϵ represent the random noise and we assume $\epsilon \sim N(0, \sigma^2)$ i.i.d.. The least square estimation for the parameter is $\hat{w} = (X^T X)^{-1} X^T y$. Since the value of $\hat{w}$ depends on the random variable y, $\hat{w}$ is also a random variable. We can derive the expectation of $\hat{w}$ and prove that it is an unbiased estimator for the parameter w

1. **[10 Points]** Given the predictors X, the value of random variable y is dependent on the random noise ϵ. Please derive the expected value for $\hat{w}$ given X: $E_{\epsilon|X}[\hat{w}]$.
2. **[10 Points]** Please derive the variance of $\hat{w}$.
3. **[Bonus (5 Points)]** For ridge regression with shrinkage parameter λ, the parameter estimation is now $\hat{w}_r(\lambda) = (X^T X + \lambda I)^{-1} X^T y$. We can show that $Var[\hat{w}_r(\lambda)] = \sigma^2 (X^T X + \lambda I)^{-1} (X^T X)(X^T X + \lambda I)^{-1}$. Could you prove that $Var[\hat{w}] \geq Var[\hat{w}_r(\lambda)]$?

Problem 2: Feature Correlation

In this problem, we will examine and compare the behavior of the Lasso and ridge regression in the case of an exactly repeated feature. That is, consider the design matrix $X \in R^{m \times d}$, where $X_{\cdot i} = X_{\cdot j}$ for some i and j, where $X_{\cdot i}$ is the i^{th} column of X. We will see that ridge regression divides the weight equally among identical features, while Lasso divides the weight arbitrarily. In an optional part to this problem, we will consider what changes when $X_{\cdot i}$ and $X_{\cdot j}$ are highly correlated (e.g. exactly the same except for some small random noise) rather than exactly the same.

1. **[10 Points]** Without loss of generality, assume the first two colums of X are our repeated features. Partition X and θ as follows:

$$X = \begin{pmatrix} x_1 & x_2 & X_r \end{pmatrix} \qquad \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_r \end{pmatrix}$$

We can write the Lasso objective function as:

$$\begin{aligned} J(\theta) &= \|X\theta - y\|_2^2 + \lambda \|\theta\|_1 \\ &= \|x_1\theta_1 + x_2\theta_2 + X_r\theta_r - y\|_2^2 + \lambda|\theta_1| + \lambda|\theta_2| + \lambda \|\theta_r\|_1 \end{aligned}$$

With repeated features, there will be multiple minimizers of $J(\theta)$. Suppose that

$$\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$$

is a minimizer of $J(\theta)$. Give conditions on c and d such that $\left(c, d, r^T\right)^T$ is also a minimizer of $J(\theta)$. [Hint: First show that a and b must have the same sign, or at least one of them is zero. Then, using this result, rewrite the optimization problem to derive a relation between a and b.]

2. **[10 Points]** Using the same notation as the previous problem, suppose

$$\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$$

minimizes the ridge regression objective function. What is the relationship between a and b, and why?

Problem 3: Linear Regression Using Gradient Descent

Save your code in `standardize.ipynb` and `regression_student.ipynb`
Suppose you are selling your house and you want to know what a good market price would be. One way to do this is to first collect information on recent houses sold and make a model of housing prices. The file housing.txt contains a training set of housing prices in Portland, Oregon. The first column is the size of the house (in square feet), the second column is the number of bedrooms, and the third column is the price of the house.

1. **[5 Points]** By looking at the values in housing.txt, you will note that house sizes are about 1000 times the number of bedrooms. When features differ by orders of magnitude, first performing feature scaling can make gradient descent converge much more quickly.
   Write a program that takes as input housing.txt and creates a file called normalized.txt. To create normalized.txt, subtract the mean value of each feature from the dataset. After subtracting the mean, additionally scale (divide) the feature values by their respective standard deviations. (Starter code is given in 'stardardize.ipynb')
   Implementation Note: When normalizing the features, it is important to store the values used for normalization - the mean value and the standard deviation used for the computations. After learning the parameters from the model, we often want to predict the prices of houses we have not seen before. Given a new x value (living room area and number of bedrooms), we must first normalize x using the mean and standard deviation that we had previously computed from the training set.

2. **[20 Points]** Our hypothesis (also called the model) will take the form $y = f(x_1, x_2) = w_0 + w_1x_1 + w_2x_2$ where x_1 is the normalized size of the house, x_2 is the normalized number of bedrooms, and y is the predicted price of the house. In this problem, your goal is to find the values of w_0, w_1, and w_2 that minimize the mean squared errors (MSE).
   Define the loss function $J(w) = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y^i})^2$, where n is the number of samples, y^i are the observed values, $\hat{y^i}$ are the predicted values. Implement gradient descent to find the values w_0, w_1, and w_2 that minimize $J(w)$. Apply your code to the normalized data set using the learning rates $\alpha = 0.01, 0.1, 0.3$.

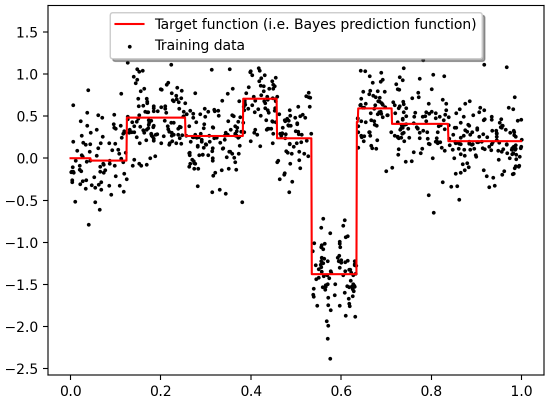

Figure 1: Training data and target function we will be considering in this assignment.

A good way to verify that gradient descent is working correctly is to look at the value of $J(w)$ and check that it is decreasing with each step. Assuming you have implemented gradient descent correctly and your learning rate is not too big, your value of $J(w)$ should never increase, and should converge to a steady value by the end of the algorithm. Plot $J(w)$ for 10, 20,30,40,50, 60,70, 80 iterations for each of your α values.

3. **[5 Points]** You will now use the w you obtained in Part 2 to predict the housing prices. Predict the price of a house with 3150 square feet and 4 bedrooms. Don't forget to normalize your features when you make this prediction!

Problem 4: Ridge Regression

For the problem, we are generating some artificial data using code in the file `setup_problem.py`. We are considering the regression setting with the 1-dimensional input space R. An image of the training data, along with the target function (i.e. the Bayes prediction function for the square loss function) is shown in Figure 1 below.

You can examine how the target function and the data were generated by looking at `setup_problem.py`. The figure can be reproduced by running the `LOAD_PROBLEM` branch of the main function.

As you can see, the target function is a highly nonlinear function of the input. To handle this sort of problem with linear hypothesis spaces, we will need to create a set of features that perform nonlinear transforms of the input. A detailed description of the technique we will use can be found in the Jupyter notebook `basis-fns.ipynb`.

In this assignment, we are providing you with a function that takes care of the featurization. This is the "featurize" function, returned by the `generate_problem` function in `setup_problem.py`. The `generate_problem` function also gives the true target function, which has been constructed to be a sparse linear combination of our features. The coefficients of this linear combination are also provided by `generate_problem`, so you can compare the coefficients of the linear functions you find to the target function coefficients. The `generate_problem` function also gives you the train and validation sets that you should use.

To get familiar with using the data, and perhaps to learn some techniques, it's recommended that you work through the main() function of the include file ridge_regression.py. You'll go through the following steps (on your own - no need to submit):

(a) Load the problem from disk into memory with load_problem.

(b) Use the featurize function to map from a one-dimensional input space to a d-dimensional feature space.

(c) Visualize the design matrix of the featurized data. (All entries are binary, so we will not do any data normalization or standardization in this problem, though you may experiment with that on your own.)

(d) Take a look at the class `RidgeRegression`. Here we've implemented our own `RidgeRegression` using the general purpose optimizer provided by scipy.optimize. This is primarily to introduce you to the sklearn framework, if you are not already familiar with it. It can help with hyperparameter tuning, as we will see shortly.

(e) Take a look at compare_our_ridge_with_sklearn. In this function, we want to get some evidence that our implementation is correct, so we compare to sklearn's ridge regression. Comparing the outputs of two implementations is not always trivial – often the objective functions are slightly different, so you may need to think a bit about how to compare the results. In this case, sklearn has total square loss rather than average square loss, so we needed to account for that. In this case, we get an almost exact match with sklearn. This is because ridge regression is a rather easy objective function to optimize. You may not get as exact a match for other objective functions, even if both methods are "correct."

(f) Next take a look at do_grid_search, in which we demonstrate how to take advantage of the fact that we've wrapped our ridge regression in an sklearn "Estimator" to do hyperparameter tuning. It's a little tricky to get GridSearchCV to use the train/test split that you want, but an approach is demonstrated in this function. In the line assigning the param_grid variable, you can see my attempts at doing hyperparameter search on a different problem. Below you will be modifying this (or using some other method, if you prefer) to find the optimal L2 regularization parameter for the data provided.

Ridge Regression

Save your code in `ridge_student.ipynb`.
In the problems below, you do not need to implement ridge regression. You may use any of the code provided in the assignment, or you may use other packages. However, your results must correspond to the ridge regression objective function that we use, namely

$$J(w;\lambda) = \frac{1}{n}\sum_{i=1}^{n}\left(w^T x_i - y_i\right)^2 + \lambda\|w\|^2.$$

1. **[15 Points]** Run ridge regression on the provided training dataset. Choose the λ that minimizes the empirical risk (i.e. the average square loss) on the validation set. Include a table of the parameter values you tried and the validation performance for each. Also include a plot of the results.

2. **[15 Points]** Now we want to visualize the prediction functions. On the same axes, plot the following: the training data, the target function, an unregularized least squares fit (still using the featurized data), and the prediction function chosen in the previous problem.
   Next, along the lines of the bar charts produced by the code in compare_parameter_vectors, visualize the coefficients for each of the prediction functions plotted, including the target function. Describe the patterns, including the scale of the coefficients, as well as which coefficients have the most weight.