CMPSC 448: Machine Learning and AI
Homework 2

Instruction

This HW includes both theory and implementation problems:

- You cannot look at anyone else's code
- Your homework must work with Python 3.7 (you may install the Anaconda distribution of Python)
- You need to submit a report including all deliverable and figures (in PDF format), also three files Problem4.ipynb, Problem5.py, and Problem5Plot.py
- The only modules your code can import are: math, numpy, matplotlib, random, pandas, sklearn

Theory and problem solving

Problem 1. [20 points] In the lectures, we showed that the MLE estimator for linear regression when the random noise for each data point is identically and independently distributed (i.i.d.) from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$ reduces to the linear regression with squared loss (OLS). In this problem, we would like to change the distribution of the noise model and derive the MLE optimization problem. In particular, let's assume for a fixed unknown linear model $\boldsymbol{w}_* \in \mathbb{R}^d$ the response y_i for each data point $\boldsymbol{x}_i \in \mathbb{R}^d$ in training data $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ is generated by

$$y_i = \boldsymbol{w}_*^\top \boldsymbol{x}_i + \epsilon_i$$

where ϵ_i is generated i.i.d. from a Laplace distribution $\epsilon_i \sim \mathsf{Laplace}(0, \sigma)$. A random variable x has a $\mathsf{Laplace}(\mu, \sigma)$ distribution if its probability density function is

$$f(x|\mu, \sigma) = \frac{1}{2\sigma} \exp(-\frac{|x - \mu|}{\sigma})$$

Under above assumption on the noise model,

1. Show that each $y_i, i = 1, 2, ..., n$ is a random variable that follows $\mathsf{Laplace}(\boldsymbol{w}_*^\top \boldsymbol{x}_i, \sigma)$ distribution.
2. Write down the MLE estimator for training data in $\mathcal{S}$ and derive the final optimization problem. Note that you just need to state the final minimization problem and not its solution.
3. Compare the obtained optimization problem to one obtained under Gaussian noise model and highlight key differences.

Problem 2. [15 points] Suppose we run a ridge regression with regularization parameter λ on a training data with a single variable $\mathcal{S} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, and get coefficient $w_1 \in \mathbb{R}$ (for simplicity, we assumed the data are centered and no bias (intercept) term is needed). We now include an exact copy of first feature to get a new training data as

$$\mathcal{S}' = \{([x_1, x_1]^\top, y_1), ([x_2, x_2]^\top, y_2), \ldots, ([x_n, x_n]^\top, y_n)\}$$

where each training example is a 2 dimensional vector with equal coordinates, refit our ridge regression on $\mathcal{S}'$ and get the solution $[w'_1, w'_2]^\top \in \mathbb{R}^2$.

1. Derive the optimal solution for $[w'_1, w'_2]^\top$ and show that $w'_1 = w'_2$.
2. What is relation between w'_1 and w_1.

Problem 3. [10 points] As we discussed in the lecture, the Perceptron algorithm will converge only if the data is linearly separable. In particular, for linearly separable data with margin γ, if $\|\boldsymbol{x}\|_2 \leq R$ for all data points ,then it will converge in at most $\left(\frac{R}{\gamma}\right)^2$ iterations as stated in the class.

If the training data $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, +1\}$ is not linearly separable, then there is a simple trick to force the data to be linearly separable and then apply the Perceptron algorithm as follows. If you have n data points in d dimensions, map data point $\boldsymbol{x}_i$ to the $(d+n)$-dimensional point $[\boldsymbol{x}_i, \boldsymbol{e}_i]^\top \in \mathbb{R}^{d+n}$, where $\boldsymbol{e}_i \in \{0, 1\}^n$ is a n-dimensional vector of zeros, except for the ith position, which is 1 (e.g., $\boldsymbol{e}_4 = [0, 0, 0, 1, 0, \ldots, 0]^\top$).

Show that if you apply this mapping, the data becomes linearly separable (you may wish to do so by providing a weight vector $\boldsymbol{w}$ in $(d+n)$-dimensional space that successfully separates the data).

Programming and experiment

Problem 4. [25 points] In this problem you will use the Pima Indians Diabetes dataset from the UCI repository to experiment with the k-NN algorithm and find the optimal value for the number of neighbors k. You do not need to implement the algorithm and encouraged to use the implementation in scikit-learn. Below is a simple code showing the steps to use the NN implementation when the number of neighbors is 3:

```
from sklearn.neighbors import KNeighborsClassifier
# Create NN classifier
knn = KNeighborsClassifier(n_neighbors = 3)
# Fit the classifier to the data
knn.fit(X_train,y_train)
# Predict the labels of test data
yhat = knn.predict(X_test)
# Or, directly check accuracy of our model on the test data
knn.score(X_test, y_test)
```

To accomplish this task, please do:

- Download the provided Pima.csv data file and load it using pandas. As a sanity check, make sure there are 768 rows of data (potential diabetes patients) and 9 columns (8 input features including Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, and 1 target output). Note that the data file has no header and you might want to explicitly create the header. The last value in each row contains the target label for that row, and the remaining values are the features. Report the statistics of each feature (min, max, average, standard deviation) and the histogram of the labels (target outputs).

- Split the data into training and test sets with 80% training and 20% test data sizes. You can easily do this in scikit-learn, e.g.,

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

  Use 5-fold cross-validation on training data to decide the best number of neighbours k. To this end, you can use the built in functionality in scikit-learn such as cross_val_score (note that this function returns the accuracy for each fold in an array and you have to average them to get the accuracy for all splits). For $k = 1, 2, 3, \ldots, 15$ compute the 5-fold cross validation error and plot the results (with values of k on the x-axis and accuracy on the y-axis). Include the plot in your report and justify your decision for picking a particular number of neighbors k.

- Evaluate the k-NN algorithm on test data with the optimal number of neighbours you obtained in previous step and report the test error.

- Process the input data by subtracting the mean (a.k.a. centralization) and dividing by the standard deviation (a.k.a. standardization) over each dimension (feature), repeat the previous part and report the accuracy. Do centralization and standardization affect the accuracy? Why?

Running and Deliverable

You are provided with a Problem4.ipynb file to put the implementation code for all parts above. Make sure your notebook is runnable on Anaconda with Python 3.7. The results and discussions should be included in the PDF file.

Problem 5. [30 points] In this problem, we consider a simple linear regression model with a modified loss function and try to solve it with Gradient Descant (GD) and Stochastic Gradient Descant (SGD).

In general setting, the data has the form $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ where $\boldsymbol{x}_i$ is the d-dimensional feature vector and y_i is a real-valued target. For this regression problem, we will be using linear prediction $\boldsymbol{w}^\top \boldsymbol{x}_i$ with the objective function:

$$f(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} g_\delta(\boldsymbol{w}; \boldsymbol{x}_i, y_i) + \lambda \sum_{j=1}^{d} w_j^2$$

where $g_\delta(\boldsymbol{w}; \boldsymbol{x}, y)$ is the error of linear model with parameter vector $\boldsymbol{w} \in \mathbb{R}^d$ on a single training pair $(\boldsymbol{x}, y)$ defined by:

$$g_\delta(\boldsymbol{w}; \boldsymbol{x}, y) = \begin{cases} (y - \boldsymbol{w}^\top \boldsymbol{x} - \delta)^2 & \text{if } y \geq \boldsymbol{w}^\top \boldsymbol{x} + \delta \\ 0 & \text{if } |y - \boldsymbol{w}^\top \boldsymbol{x}| < \delta \\ (y - \boldsymbol{w}^\top \boldsymbol{x} + \delta)^2 & \text{if } y \leq \boldsymbol{w}^\top \boldsymbol{x} - \delta \end{cases}$$

Please note that we simply dropped the intercept term by the simple trick we discussed in the lecture, i,e., adding a constant feature, which is always equal to one to simplify estimation of the "intercept" term.

Gradient Descent

In this part, you are asked to optimize the above objective function using gradient descent and plot the function values over different iterations, which can be done using the python library matplotlib.

To this end, in Problem5.py, fill in the function bgd_l2(data, y, w, eta, delta, lam, num_iter) where data is a two dimensional numpy array with each row being a feature vector, y is a one-dimensional numpy array of target values, w is a one-dimensional numpy array corresponding to the weight vector, eta is the

learning rate, delta and lam are parameters of the objective function. This function should return new weight vector, history of the value of objective function after each iteration (python list).

Run this function for the following settings and plot the history of objective function (you should expect a monotonically decreasing function over iteration number):

- $\eta = 0.05, \delta = 0.1, \lambda = 0.001, \text{num\_iter} = 50$
- $\eta = 0.1, \delta = 0.01, \lambda = 0.001, \text{num\_iter} = 50$
- $\eta = 0.1, \delta = 0, \lambda = 0.001, \text{num\_iter} = 100$
- $\eta = 0.1, \delta = 0, \lambda = 0, \text{num\_iter} = 100$

Stochastic Gradient Descent

In Problem5.py fill in the function sgd_l2(data, y, w, eta, delta, lam, num_iter, i) where data, y, w, lam, delta, num_iter are same as previous part. In this part, you should use $\frac{n}{\sqrt{t}}$ as a learning rate, where t is the iteration number, starting from 1. The variable i is for testing the correctness of your function. If i set to -1 then you just need to apply the normal SGD (randomly select the data point), which runs for num_iter, but if i set to something else (other than -1), your code only needs to compute **SGD** for that specific data point (in this case, the num_iter will be 1!).

Run this function for the settings below and plot the history of objective function:

- $\eta = 1, \delta = 0.1, \lambda = 0.5, \text{num\_iter} = 800$
- $\eta = 1, \delta = 0.01, \lambda = 0.1, \text{num\_iter} = 800$
- $\eta = 1, \delta = 0, \lambda = 0, \text{num\_iter} = 40$
- $\eta = 1, \delta = 0, \lambda = 0, \text{num\_iter} = 800$

Running and Deliverable

You are provided with a sample data sets (data.npy). You can load these files into numpy array using this syntax: np.load('data.npy'). The sample data for this homework has 100 data points (x_i, y_i) in a 100×2 numpy array. Please note that you need to add a column of all ones to data to handle the intercept term, making your data a 100×3 numpy array. For the plotting part, make sure that your plots have appropriate title, x and y-axis labels. You need to submit two python files Problem5.py and Problem5Plot.py and a PDF file including all the plots. In Problem5.py everything except the imports should be inside the functions definition we mentioned above. The file Problem5Plot.py is where you are generating the plots.