ECE 595: Machine Learning I
Spring 2019
Instructor: Prof. Stanley H. Chan

Homework 2 Introduction to Optimization Tools

Spring 2019
(Due: Friday, Feb 1, 2019)

Homework is due at 4:30pm. Please put your report in the dropbox located at MSEE 330. No late homework will be accepted.

Objective

The objective of this homework is threefold:

(a) Familiarizing yourself with basic tools in Python that can perform the following tasks: reading and processing of data files (csv format), and numerically solving specified optimization problems;

(b) Review of important concepts from linear algebra and optimization theory, with emphasis on linear least square estimation;

(c) Implement a linear classification algorithm, visualize its decision boundary, and test its accuracy.

Installation and Get Started

In this project, you will be asked to numerically solve several convex optimization problems in Python. Of course, you can use whichever optimization toolbox you prefer, but for those new to optimization tools in Python, we introduce a popular toolbox here, CVXPY. CVXPY is a Python-embedded modeling language with a user-friendly API for convex optimization problems.

To install the software package, detailed instructions are given on the website `https://www.cvxpy.org/install/index.html`. A word of advice: if after installing the package in Anaconda, you cannot open Spyder, try upgrading Anaconda to the newest version.

After installation, let us look at a toy example. The following program solves a linear least-squares problem with box constraint, taken directly from the software's website:

```
import cvxpy as cp
import numpy as np

# Problem data.
m = 30
n = 20
np.random.seed(1)
A = np.random.randn(m, n)
b = np.random.randn(m)

# Construct the problem.
x = cp.Variable(n)
objective = cp.Minimize(cp.sum_squares(A*x - b))
constraints = [0 <= x, x <= 1]
prob = cp.Problem(objective, constraints)
```

```
# The optimal objective value is returned by ‘prob.solve()‘.
result = prob.solve()
# The optimal value for x is stored in ‘x.value‘.
print(x.value)
# The optimal Lagrange multiplier for a constraint is stored in
# ‘constraint.dual_value‘.
print(constraints[0].dual_value)
```

For more examples, head over to `https://www.cvxpy.org/examples/index.html`.

Important Concepts

Classifiers and Training

Consider a dataset $\mathcal{D}$ consisting of training samples $\{(\boldsymbol{x}_j, y_j)\}_{j=1}^N$. The vector $\boldsymbol{x}_j \in \mathbb{R}^d$ is called the i-th input vector, and the scalar $y_j \in \{1, \ldots, K\}$ is called the class label. We assume that there are K labels, and we denote the number of samples that has label $y_j = k$ to be N_k.

In the simplest sense, a K-class classifier is a mapping $h : \mathbb{R}^d \to \{1, 2, ..., K\}$ which takes a measurement $\boldsymbol{x} \in \mathbb{R}^d$ and predicts its class. The goal of **training** the classifier is to use existing data points (the $\boldsymbol{x}_j$'s) and their respective class labels (the y_j's) to find $(d-1)$-dimensional spaces (e.g. hyperplanes) that divide the feature space $\mathbb{R}^d$ into a finite number of regions, each region corresponding to a class; we call these regions **decision regions**, and the $(d-1)$-dimensional spaces separating them **decision boundaries**. Therefore, the problem of finding the classifier reduces to finding the decision regions, or decision boundaries. Some parametric methods (the category where linear classifiers belong to) find a decision boundary between two adjacent clusters of data points with different labels by solving a minimization problem with the objective function being the error generated by the inaccuracy of the classifier, such as the least square problem:

$$g^* = \underset{g}{\text{argmin}} \ \sum_{j=1}^N (g(\boldsymbol{x}_j) - y_j)^2 \tag{1}$$

where, for instance, we take the decision boundary to be $\{\boldsymbol{x} \in \mathbb{R}^d | \, g(\boldsymbol{x}) = r\}$ for some constant $r \in \mathbb{R}$. Notice that, if there is no restriction on g, the above optimization will land on a trivial solution where g is a point estimator so that every $\boldsymbol{x}_j$ can be mapped to y_j perfectly. This will give zero training error, but the resulting classifier g will have very poor generalizing capability, i.e., it is over-fitted. To alleviate the problem, it is necessary to pose structure on the classifier. In this project, we will study a type of linear classifier, where the function g in equation (1) takes the special form

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 \tag{2}$$

and the classification function h bases its predictions on the continuous outputs from g.

Exercise 1: Loading Data via Python

The National Health and Nutrition Examination Survey (NHANES) is a program to assess the health and nutritional status of adults and children in the United States [1]. The complete survey result contains over 4,000 samples of health-related data of individuals who participated in the survey between 2011 and 2014. In this project, we will focus on two categories of the data for each individual: the height (in mm) and body mass index (BMI). The data is divided into two classes based on gender. Table 1 contains snippets of the data.

[1] https://www.cdc.gov/nchs/nhanes/index.htm

index	female_bmi	female_stature_mm
0	28.2	1563
1	22.2	1716
2	27.1	1484
3	28.1	1651

index	male_bmi	male_stature_mm
0	30	1679
1	25.6	1586
2	24.2	1773
3	27.4	1816

Table 1: Male and Female Data Snippets

✓(a) Use `csv.reader` to read the training data files for the two classes, `male_train_data.csv` and `female_train_data.csv`. Feel free to use a data structure that you like to store the read values. If you prefer, you can complete the following partial code to read the csv files:

```
# Reading csv file for male data
with open("male_train_data.csv", "r") as csv_file:
    reader = csv.reader(csv_file, delimiter=’,’)
    # Add your code here to process the data into usable form
csv_file.close()

# Reading csv file for female data
with open("female_train_data.csv", "r") as csv_file:
    reader = csv.reader(csv_file, delimiter=’,’)
    # Add your code here to process the data into usable form
csv_file.close()
```

✓(b) Print a few entries of the data read in part (a) to verify that the read was successful.

Please submit your code.

Exercise 2: Building a Linear Classifier via Optimization

The goal of this exercise is to build a simple two-class linear classifier via optimization tools. Although this is a relatively simple exercise, it is not difficult to generalize the idea to higher dimensions and more complicated data distributions.

Consider a dataset with only two classes of sample, $\mathcal{D} = \{(\boldsymbol{x}_j, y_j)\}_{j=1}^N$, where $y_j \in \{+1, -1\}$. We define the decision boundary of our **linear classifier** as the set $\{\boldsymbol{x} \in \mathbb{R}^d | \, g(\boldsymbol{x}) = 0\}$ where:

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0, \tag{3}$$

Note that $g : \mathbb{R}^d \to \mathbb{R}$ is a linear function, hence, the decision boundary is a (affine) hyperplane; moreover, $h(\boldsymbol{x}) = \text{sign}(g(\boldsymbol{x}))$ for this classifier. We call $\boldsymbol{w} \in \mathbb{R}^d$ the coefficient vector, and w_0 the bias term.

To minimize the error created by the ”inflexibility” of our decision boundary, we choose to solve the least square problem:

$$g^* = \underset{g}{\text{argmin}} \sum_{j=1}^{N} (g(\boldsymbol{x}_j) - y_j)^2 \tag{4}$$

Substituting (3) into (4) yields

$$\begin{bmatrix} \boldsymbol{w}^* \\ w_0^* \end{bmatrix} = \underset{\boldsymbol{w}, w_0}{\text{argmin}} \sum_{j=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_j + w_0 - y_j)^2 \tag{5}$$

Therefore, we simplified the optimization to finding $\boldsymbol{w}$ and w_0.

Tasks:

(a) Let $\boldsymbol{\theta} = (\boldsymbol{w}, w_0) \in \mathbb{R}^{d+1}$ be the concatenated classifier coefficient vector, and consider the optimization problem (5). Rewrite it into the form of

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmin}} \quad \|\boldsymbol{A\theta} - \boldsymbol{b}\|_2^2\,, \tag{6}$$

for some matrix $\boldsymbol{A} \in \mathbb{R}^{N\times(d+1)}$ and vector $\boldsymbol{b} \in \mathbb{R}^N$. Write $\boldsymbol{A}$ and $\boldsymbol{b}$ in terms of the $\boldsymbol{x}_j$'s and y_j's.

(b) Prove, by vector calculus, that the optimal solution for (6) is given by

$$\boldsymbol{\theta}^* = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}. \tag{7}$$

Under what condition on $\boldsymbol{A}$ is the above inversion valid? What does this mean to the measurement vectors $\boldsymbol{x}_i$? If $(\boldsymbol{A}^T\boldsymbol{A})$ cannot be inverted, what are the possible ways of resolving this issue?

(c) Construct the matrix $\boldsymbol{A}$ and $\boldsymbol{b}$ in Python for the NHANES **training** dataset. **In this project, we assign the label** $+1$ **to the male class, and** -1 **to the female class**. Compute the optimal weight vector $\boldsymbol{\theta}^*$.

Hint: You might find functions such as `numpy.concatenate`, and `numpy.ones` useful.

(d) Repeat the previous step using CVXPY instead. Compare the solution, i.e., $\boldsymbol{\theta}^*$ returned by CVXPY and the result you obtained in the previous step.

Exercise 3: Visualization and Testing

Having found the optimal $\boldsymbol{\theta}^*$ in exercise 2, and recalling that the decision boundary of our linear classifier $\{\boldsymbol{x} \in \mathbb{R}^2 |\, g^*(\boldsymbol{x}) = \boldsymbol{w}^{*T}\boldsymbol{x} + w_0^* = 0\}$ is a hyperplane (in our case, it is just a line), it is time for us to test the performance of the linear classifier.

Tasks:

(a) First, let us visualize our situation by plotting this linear decision boundary and the training data points of the two classes.

(i) Plot the training data points of the male and female classes using `matplotlib.pyplot.scatter`.

(ii) Writing $\boldsymbol{x} \in \mathbb{R}^2$ as $\boldsymbol{x} = (x_1, x_2)$, and $\boldsymbol{w}^* = (w_1^*, w_2^*)$, what is the explicit expression for the linear decision boundary of our classifier? Express x_2 in terms of w_1^*, w_2^*, w_0^* and x_1.

(ii) Plot the decision boundary of the classifier on top of the scatter plot.

(b) Finally, let us test the accuracy of our classifier.

(i) Read and parse the data files `male_test_data.csv` and `female_test_data.csv`.

(ii) Recall from exercise 2 that we associated the $+1$ label to the male class, and -1 to the female class, and our classifer h's prediction is defined as $h(\boldsymbol{x}) = \text{sign}(g^*(\boldsymbol{x}))$. Now we can use our trained classifier on the test dataset read from the previous part. Denoting s_m as the number of correctly predicted male test samples, and s_f as the number of correctly predicted female test samples, and t as the total number of test samples, we calculate the overall success rate of our classifier using the following formula:

$$\text{success rate} = \frac{s_m + s_f}{t}$$

Please submit your final plots and code, and report on your classifier's accuracy.

Exercise 4: Regularization

In this exercise we would like to explore the concept of regularization. For empirical data, let us keep using the $\boldsymbol{A}$ and $\boldsymbol{b}$ we obtained from the NHANES training data in the previous exercises.

Consider the following three optimization problems:

$$\boldsymbol{\theta}_\lambda = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \quad \|\boldsymbol{A}\boldsymbol{\theta} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2, \tag{8}$$

$$\boldsymbol{\theta}_\alpha = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \quad \|\boldsymbol{A}\boldsymbol{\theta} - \boldsymbol{b}\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{\theta}\|_2^2 \le \alpha, \tag{9}$$

$$\boldsymbol{\theta}_\epsilon = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \quad \|\boldsymbol{\theta}\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{A}\boldsymbol{\theta} - \boldsymbol{b}\|_2^2 \le \epsilon. \tag{10}$$

Caution: Before proceeding to the problems, please be careful that numerical solvers can perform very badly if the inputs to the problem are badly scaled, e.g. many coefficients being orders of magnitude apart; this is known as **numerical instability**. It is exactly the case for our matrix $\boldsymbol{A}$. To avoid CVXPY (by default, the ECOS solver in it) returning bizarre solutions, please divide every number in $\boldsymbol{A}$ corresponding to the `male_stature_mm` or `female_stature_mm` column by 100, so instead of mostly being on the order of 10^3, they should now mostly be on the order of 10^0. This will significantly reduce the numerical error. If curious, try solving part (c) with the raw $\boldsymbol{A}$.

Tasks:

(a) Consider problem (8). We study the effect of adding a penalty term in the objective function of the least square problem.

(i) Plot $\|\boldsymbol{A}\boldsymbol{\theta}_\lambda - \boldsymbol{b}\|_2^2$ with respect to $\|\boldsymbol{\theta}_\lambda\|_2^2$, and both of them with respect to λ, by sampling the two quantities at $\lambda = 0.1, 0.2, 0.3, ..., 10$. Also plot the decision boundary generated by the $\boldsymbol{\theta}_\lambda$'s for every 10 λ, i.e., obtain plots for $\boldsymbol{\theta}_{\lambda=0.1}, \boldsymbol{\theta}_{\lambda=1.1}, ..., \boldsymbol{\theta}_{\lambda=9.1}$. If you prefer, you can use the following partial code for plotting the decision boundaries (of course, you need to write your own `INTERCEPT` and `SLOPE`, and make minor changes based on your implementations).

```
lambda_list = np.arange(0.1, 10, 0.1)
for lambda in lambda_list:
    # Solve the regularized least-squares problem depending on lambda
...
# Plot the decision boundaries with theta_{lambda=0.1, 1.1, ..., 9.1}
x = np.linspace(x_min, x_max, 200)
legend_str = []
plt.figure(figsize=(15,7.5))
for i in range(len(lambda_list))[0::10]:
    y = INTERCEPT(lambda_list[i]) + SLOPE(lambda_list[i])*x
    plt.plot(x, y.T)
    legend_str.append('$\lambda = $' + str(lambda_list[i]))
plt.legend(legend_str)
plt.xlabel('BMI')
plt.ylabel('Stature')
```

(ii) Comment on your solutions to (i), and feel free to include extra experimental results. For instance, what effects do you observe when we impose the penalty term in (8)? How does varying λ affect $\|\boldsymbol{A}\boldsymbol{\theta}_\lambda - \boldsymbol{b}\|_2^2$ and $\|\boldsymbol{\theta}_\lambda\|_2^2$? What about the decision boundaries? Why? Justify your answer.

(b) Let us consider some theoretical details of the three optimization problems. Assume solutions exist for the three problems in this part.

(i) Write down the Lagrangian for the three problems, and find the Karush-Kuhn-Tucker (KKT) optimality conditions for them. You do not need to obtain the solutions.

(ii) Fix $\lambda > 0$. Prove that when $\alpha = \|\boldsymbol{\theta}_\lambda\|_2^2$, optimization problems (8) and (9) are equivalent, i.e., show that $\boldsymbol{\theta}_\lambda = \boldsymbol{\theta}_\alpha$.

Hint: Let γ_α denote the Lagrange multiplier to (9). Does $(\boldsymbol{\theta}_\lambda, \gamma_\alpha = \lambda)$ satisfy the KKT conditions of (9) when $\alpha = \|\boldsymbol{\theta}_\lambda\|_2^2$?

(iii) Again, fix $\lambda > 0$. Prove that when $\epsilon = \|\boldsymbol{A}\boldsymbol{\theta}_\lambda - \boldsymbol{b}\|_2^2$, the optimization problems (8) and (10) are equivalent, i.e. show that $\boldsymbol{\theta}_\lambda = \boldsymbol{\theta}_\epsilon$.

(c) Let us now obtain some empirical insight about problems (9) and (10). Also fix $\lambda = 0.1$ in this part.

(i) Denote $\alpha^* = \|\boldsymbol{\theta}_{\lambda=0.1}\|_2^2$. Repeat part (a)(i) (except the plotting decision boundary task) and (ii) for problem (9), for $\alpha = \alpha^* - 2 \times 50, \alpha^* - 2 \times 49, ..., \alpha^*, ..., \alpha^* + 2 \times 50$. Additionally, verify your finding in (b)(ii).

Remark: If there are slight disagreements between the solution to (8) and (9), try multiplying the objective function of (9) by a small number less than 1, e.g. 0.1. This does not change the solution theoretically, and reduces the numerical error from CVXPY.

(ii) Denote $\epsilon^* = \|\boldsymbol{A}\boldsymbol{\theta}_{\lambda=0.1} - \boldsymbol{b}\|_2^2$. Repeat part (a)(i) (except the plotting decision boundary task) and (ii) for problem (10), for $\epsilon = \epsilon^* + 2 \times 0, \epsilon^* + 2 \times 1, ..., \epsilon^* + 2 \times 100$. Additionally, verify your finding in (b)(iii).

(d) Having shown equivalence of these optimization problems when the λ, α and ϵ are carefully chosen, the question now is: what can we say about the physical interpretation of these problems? Does one optimization problem work better/appeal better to intuition over another in different contexts? Explain your answers.