ECE 586 Application: Least-Squares Solutions

Henry D. Pfister
Duke University

November 30th, 2019

1 A Few Questions

1.1 How does one fit a polynomial to a set of points?

For a given set of N data points $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N) \in \mathbb{R}^2$, what degree-$m$ polynomial,

$$f(x) = \sum_{j=0}^{m} a_j x^j,$$

best fits the data? Of course, the answer depends on what we mean by "best". If we focus on the *mean-squared error* (MSE), then the goal is to minimize the quantity

$$\frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2$$

over the polynomial coefficients $a_0, a_1, \ldots, a_m \in \mathbb{R}$. If $N > m + 1$, then the polynomial may not fit each point exactly and the result that minimizes this error is called the *least-squares solution.*

1.2 How can one learn to classify patterns from training examples?

Consider a system, which can be in one of two states, that generates independent random vectors using a fixed distribution for each state. Suppose we are given N observations of the system, $(\underline{x}_1, y_1), (\underline{x}_2, y_2), \ldots, (\underline{x}_N, y_N)$, where $\underline{x}_i \in \mathbb{R}^n$ is the observed vector and $y_i \in \{-1, 1\}$ is the system state, and asked to classify a new observation $\underline{x} \in \mathbb{R}^n$ for which the state is unknown.

One approach to this problem is to fit a function $f \colon \mathbb{R}^n \to \{-1, 1\}$ to the given N observations and then use that function to classify $\underline{x}$. This approach is known as *empirical risk minimization.* Specifically, we define a non-negative function $L \colon \{-1, 1\} \times \{-1, 1\} \to \mathbb{R}_{\geq 0}$ that quantifies the loss $L(\hat{y}, y)$ incurred by classifying the true state y to the estimate $\hat{y}$. Then, we minimize the empirical risk

$$R_N(f) \triangleq \frac{1}{N} \sum_{i=1}^{N} L\left(f(\underline{x}_i), y_i\right)$$

over all functions f in a some class $\mathcal{F}$. The resulting function is denoted by

$$\hat{f} = \arg\min_{f \in \mathcal{F}} R_N(f).$$

To compare with the optimal function, we assume that the training data is generated by i.i.d. samples from some true distribution $P_{\underline{X},Y}(\underline{x}, y)$. An important result from learning theory is that, if the space $\mathcal{F}$ of functions has nice properties and N is large enough, then $\hat{f}$ will be close to the function

$$f^* = \arg\min_{f \in \mathcal{F}} R(f) = \arg\min_{f \in \mathcal{F}} \mathbb{E}\left[L\left(f(\underline{X}), Y\right)\right]$$

that minimizes the expected risk $R(f) \triangleq \mathbb{E}\left[L\left(f(\underline{X}), Y\right)\right]$, where the expectation over $P_{\underline{X},Y}(\underline{x}, y)$.

For computational reasons, this approach is sometimes relaxed to fit a function $f\colon \mathbb{R}^n \to \mathbb{R}$ with the loss function

$$L(\hat{y}, y) = (y - \hat{y})^2.$$

Then, the vector is classified by rounding the output to the set $\{-1, 1\}$.

2 Linear Least-Squares

The most common linear least-squares problem is the solution of an overdetermined system of linear equations. Let $A \in \mathbb{R}^{N\times M}$ be an $N \times M$ matrix with $N > M$ that defines N linear equations in M variables,

$$y_i = \sum_{j=1}^{M} A_{i,j} z_j.$$

When $N > M$, it is likley that there is no $\underline{z}$ vector that satisfies all of these equations. Thus, one often asks for a least-squares solution that minimizes the squared error

$$\hat{\underline{z}} = \arg\min_{\underline{z}} \left(\frac{1}{N} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{M} A_{i,j} z_j \right)^2 \right).$$

As we will see in this class, if A is full rank, then this solution can be written as

$$\hat{\underline{z}} = (A^T A)^{-1} A^T \underline{y}.$$

This setup naturally extends to fitting a vector space of functions $\mathcal{F} = \operatorname{span}\{f_1, f_2, \ldots, f_M\}$, where $f_j\colon \mathbb{R}^n \to \mathbb{R}$ for $j = 1, \ldots, M$, to a set of input/output pairs, $(\underline{x}_1, y_1), (\underline{x}_2, y_2), \ldots, (\underline{x}_N, y_N)$. In particular, we define

$$f(\underline{x}; \underline{z}) \triangleq \sum_{j=1}^{M} z_j f_j(\underline{x}),$$

choose $L(\hat{y}, y) = (y - \hat{y})^2$, and observe that

$$\begin{aligned}
R_N(f) &\triangleq \frac{1}{N} \sum_{i=1}^{N} L\left(f(\underline{x}_i; \underline{z}), y_i\right) \\
&= \frac{1}{N} \sum_{i=1}^{N} \left(y_i - f(\underline{x}_i; \underline{z})\right)^2 \\
&= \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{M} z_j f_j(\underline{x}_i) \right)^2 \\
&= \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{M} A_{i,j} z_j \right)^2,
\end{aligned}$$

where we define $A_{i,j} \triangleq f_j(\underline{x}_i)$. Thus, using this setup, one can minimize the empirical risk by finding a least-squares solution to the linear system $\underline{y} = A\underline{z}$. For example, if A has full rank and $N > M$, then one can use the formula $\underline{z} = (A^T A)^{-1} A^T \underline{y}$.

In some cases, it is more computationally efficient (or convenient) to minimize using gradient descent. Letting $\underline{a}_i = [A_{i,1}, \ldots, A_{i,M}]$ be the i-th row of A, we can write the gradient of the loss function as

$$\nabla_{\underline{z}} \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \underline{a}_i \cdot \underline{z}\right)^2 = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\underline{z}} \left(y_i - \underline{a}_i \cdot \underline{z}\right)^2 = -\frac{2}{N} \sum_{i=1}^{N} \left(y_i - \underline{a}_i \cdot \underline{z}\right) \underline{a}_i.$$

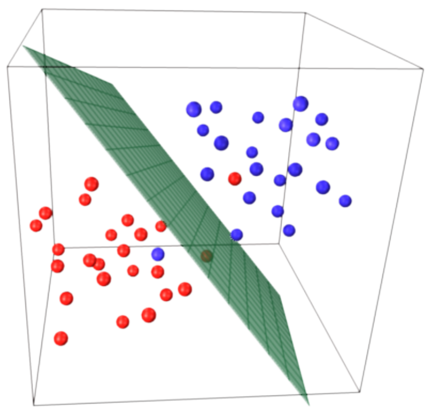

Figure 1: This hyperplane is designed to separate two classes in three dimensions. It classifies all but two points correctly.

Since the gradient is the sum of N simple terms, a standard approach is to use the terms separately to update $\underline{z}$ during each step. Starting from $\underline{z}^{(0)} = \underline{0}$, the update for cyclic partial gradient descent is

$$\underline{z}^{(t+1)} = \underline{z}^{(t)} + \gamma_t \left(y_{(t \bmod N)+1} - \underline{a}_{(t \bmod N)+1} \cdot \underline{z} \right) \underline{a}_{(t \bmod N)+1},$$

where γ_t is the step size.

Exercise 1. (5 pts Vandermonde function) When $n = 1$, we can fit a degree-m polynomial by choosing $f_j(x) = x^{j-1}$ and $M = m + 1$. In this case, it follows that $A_{i,j} = x_i^{j-1}$ and the matrix A is called a Vandermonde matrix. Implement a function to compute a Vandermonde matrix given the x values and polynomial degree.

Exercise 2. (5 pts solve_linear_LS function + 5 pts print MSE + 5 pts plot) Using the setup in the previous example, fit the points $(1, 2), (2, 3), (3, 5), (4, 7), (5, 11), (6, 13)$ to a degree-2 polynomial using linear algebra. What is the resulting mean squared error? Plot the resulting polynomial (for $x \in [0, 7]$) along with the data points to see the quality of fit.

Exercise 3. (5 pts solve_linear_LS_gd function + 5 pts print MSE + 5 pts plot) Using the setup in the previous problem, fit the given points to a degree-2 polynomial using cyclic partial gradient descent. For the fixed step size $\gamma_t = 0.0002$, determine a value of T such that $\underline{z}^{(T)}$ achieves a mean squared error that is at most 20% larger than the previous answer. Plot this polynomial (for $x \in [0, 7]$) along the previous polynomial and the data points.

3 Training a Linear Classifier

A linear classifier is a linear function that classifies points $\underline{x} \in \mathbb{R}^n$ by testing

$$\sum_{i=1}^{n} x_i z_i \gtrless 0, \qquad (1)$$

with ties broken arbitraily (see Figure 1). Since the LHS of (1) equals the standard inner product $\langle \underline{x} | \underline{z} \rangle$, this value is quite related to the vector projection of $\underline{x}$ onto $\underline{z}$. In fact, if $\|\underline{z}\| = 1$, then the LHS of (1) equals the coefficient of $\underline{z}$ for the vector projection of $\underline{x}$ onto $\underline{z}$. The orthogonal complement of $\underline{z}$ is the hyperplane whose normal vector is $\underline{z}$ and the LHS of (1) equals zero for all points in that hyperplane. Therefore, that hyperplane is the *decision boundary* for the test in (1).

Mathematically, the space $\mathbb{R}^n$ is divided into two disjoint sets by a hyperplane containing the origin. To allow instead for a general hyperplane, which may not contain the origin, one can append a "-1"

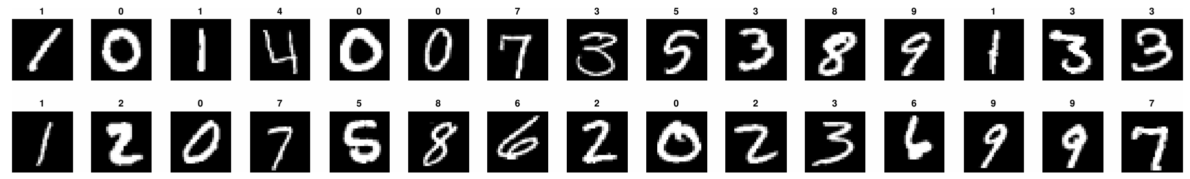

Figure 2: Sample digits from the MNIST database.

to the end of each $\underline{x}$ data vector (i.e., extending the length by one). In that case, we may assume that $x_n = -1$ and this formula can be rewritten as

$$\sum_{i=1}^{n-1} x_i z_i \gtrless z_n,$$

which defines a general hyperplane separation for $\underline{x} \in \mathbb{R}^{n-1}$.

One can train a linear classifier using least-squares by performing empirical risk minimization with the loss function $L(\hat{y}, y) = (y - \hat{y})^2$ and function class

$$\mathcal{F} = \left\{ f(\underline{x}) = \sum_{i=1}^{n} x_i z_i \,\middle|\, z_1, \ldots, z_n \in \mathbb{R} \right\}.$$

This problem can be solved using the approach described above with $f_j(\underline{x}) = \left[\underline{x}\right]_j = x_j$, which implies that $A_{i,j} = \left[\underline{x}_i\right]_j$. To evaluate a trained classifier, one needs additional samples that are independent from the samples used in training. In practice, this can be achieved by breaking the original data set into two pieces: a training set and a test set. This is called *cross validation*[1].

For the trained classifier, one should report the *classification error rate* (i.e., the fraction of data set that is misclassified) for both the training and test sets. Also, a more detailed way to present classification error rates is with a confusion matrix. A *confusion matrix* C is a matrix that contains whose $C_{i,j}$ entry equals the number of times a data vector is predicted to be in class j (by the classifier) when the its true class is i.

Definition 1. The *MNIST database* is a set of digitized handwritten digits that is used for testing and training classifiers (see Figure 2). Each sample in the dataset is stored as a 28×28 pixel image with each pixel taking on 256 grayscale levels. Look in the Projects directory of the Sakai resources for "mnist_train.csv". Each line contains a comma separated vector with 785 elements. The first element is the digit label (from 0-9) and the remaining $784 = 28 * 28$ elements are the pixel values for that sample. The following Matlab/Python snippets both load the MNIST training data and plot the first 30 images.

```
### Python Code
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# Read MNIST csv file into dataframe
df = pd.read_csv('mnist_train.csv')
# Merge pixels into feature column and keep only feature and label
df['feature'] = df.apply(lambda row: row.values[1:], axis=1)
df = df[['feature', 'label']]

# Plot MNIST
plt.figure(figsize=(15, 2.5))
```

[1]More generally, the set can be broken into k pieces and one can use k-fold cross validation.

```
for i, row in df.iloc[:30].iterrows():
    x, y = row['feature'], row['label']
    plt.subplot(2, 15, i + 1)
    plt.imshow(x.reshape(28, 28), cmap='gray')
    plt.axis('off')
    plt.title(y)
```

Exercise 4. (10 pts extract_and_split function + 15 pts mnist_pairwise_LS function) In this problem, we will use least-squares via linear algebra to train a linear classifier to distinguish between handwritten 0's and 1's. First, extract the indices of all the 0's and randomly separate the samples into equal-sized training and testing groups. Second, do the same for the 1's. Now, extend each vector to length 785 by appending a -1. This will allow the system to learn a general hyperplane separation. Now, use least squares to find a linear function that tries to map 0-samples to -1 and 1-samples to $+1$. Optional: Try also solving this problem with cyclic partial gradient descent and compare the two methods.

For the resulting linear function, report the classification error rate and confusion matrices for the both the training and test sets. Also, for the test set, compute the histogram of the function output separately for each class and then plot the two histograms together. This shows easy or hard it is to separate the two classes.

Note: If the A matrix is not full-rank (e.g., suppose a single pixel is 0 for all samples) then the $A^T A$ matrix will be singular. In this case, the Matlab expression `z=A\y` still finds a reasonable (i.e., minimum-norm) least-squares solution. In Python, the NumPy function `np.linalg.lstsq` also finds a reasonable solution for all A.

Exercise 5. (10 pts for reasonable values) Repeat the above problem (solving only via the normal equations) for all pairs of digits. For each pair of digits, report the classification error rates for the training and testing sets. The error rates can be formatted nicely into a triangular matrix. For storage and display efficiency, store the testing error in the lower triangle and the training error in the upper triangle.

Exercise 6. (20 pts for multiclass code and results) But, what about a multi-class classifier for MNIST digits? For multi-class linear classification with d classes, one standard approach is to learn a linear mapping $f\colon \mathbb{R}^n \to \mathbb{R}^d$ where the "y"-value for the i-th class is chosen to be the standard basis vector $\underline{e}_i \in \mathbb{R}^d$. This is sometimes called *one-hot* encoding. Using the same A matrix as before and a matrix Y, defined by $Y_{i,j} = 1$ if observation i in class j and $Y_{i,j} = 0$ otherwise, we can solve for the coefficient matrix $Z \in \mathbb{R}^{n \times d}$ coefficients with the Matlab expression `Z=A\Y`. Then, the classifier maps a vector $\underline{x}$ to class i if the i-th element of $Z^T \underline{x}$ is the largest element in the vector.