CMSC 25300: Mathematical Foundations of ML

Problem Set 3

Hung Le Tran

21 Oct 2023

Problem 3.1 (Problem 1)

Solution

(a) $\mathbf{X}_1 = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$ $\mathbf{X}_2 = \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix}$. Gram-Schimdt:

$$\mathbf{U}_1 = \frac{\mathbf{X}_1}{||\mathbf{X}_1||} = \frac{1}{3}\begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{X}_2' = \mathbf{X}_2 - \mathbf{U}_1(\mathbf{U}_1^T\mathbf{X}_2) = \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix} - 1\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 3 \end{bmatrix}$$

$$\Rightarrow \mathbf{U}_2 = \frac{1}{\sqrt{18}}\begin{bmatrix} 0 \\ 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

(b)

$$\mathbf{U} = \begin{bmatrix} 1 & 0 \\ 0 & 1/\sqrt{2} \\ 0 & 1/\sqrt{2} \end{bmatrix}$$

The LS estimate $\hat{\mathbf{y}}$ is nothing more than the point on the subspace S spanned by the columns of $\mathbf{X}$ that is closest to $\mathbf{y}$, i.e. the projection of $\hat{\mathbf{y}}$ on S. This is

$$\hat{\mathbf{y}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

However, this space S also spanned by the columns of $\mathbf{U}$, as it is simply the orthogonalization of the columns of $\mathbf{X}$. It follows that one can also retrieve the LS-estimate:

$$\hat{\mathbf{y}} = \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{y}$$

(c) Computing the LS-estimate is easier using $\mathbf{U}$ because

$$\mathbf{U}^T\mathbf{U} = 1$$

so the estimate is simply

$$\hat{\mathbf{y}} = \mathbf{U}(\mathbf{U}^T\mathbf{y})$$

In this example,

$$\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{U}\left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}\begin{bmatrix} 1 \\ 6 \\ -2 \end{bmatrix}\right) \\
&= \mathbf{U}\begin{bmatrix} 1 \\ 2\sqrt{2} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ 0 & 1/\sqrt{2} \\ 0 & 1/\sqrt{2} \end{bmatrix}\begin{bmatrix} 1 \\ 2\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}
\end{aligned}$$

□

Problem 3.2 (Problem 2)

Solution

(a)

$$S = \{\mathbf{x} \in \mathbb{R}^3 \mid 4x_1 - x_2 - 2x_3 = 0\}$$

If $(x_1, x_2, x_3) \in S$ then if $x_3 = 2t, x_2 = 4s$ then $x_1 = t + s$. It follows that

$$S = \{\mathbf{x} \in \mathbb{R}^3 \mid x_1 = t + s, x_2 = 4s, x_3 = 2t \text{ for some } t, s \in \mathbb{R}\}$$

It follows that any $\mathbf{x} \in S$ can be represented as

$$\begin{bmatrix} t+s \\ 4s \\ 2t \end{bmatrix} = t\begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} + s\begin{bmatrix} 1 \\ 4 \\ 0 \end{bmatrix}$$

We have found a non-orthonormal basis of S:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 0 & 4 \\ 2 & 0 \end{bmatrix}$$

And the corresponding projection matrix is

$$\begin{aligned}
\mathbf{P} &= \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \\
&= \begin{bmatrix} 1 & 1 \\ 0 & 4 \\ 2 & 0 \end{bmatrix} \left(\begin{bmatrix} 1 & 0 & 2 \\ 1 & 4 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 4 \\ 2 & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 0 & 2 \\ 1 & 4 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1 \\ 0 & 4 \\ 2 & 0 \end{bmatrix} \left(\begin{bmatrix} 5 & 1 \\ 1 & 17 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 0 & 2 \\ 1 & 4 & 0 \end{bmatrix} \\
&= \frac{1}{84} \begin{bmatrix} 1 & 1 \\ 0 & 4 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} 17 & -1 \\ -1 & 5 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 1 & 4 & 0 \end{bmatrix} \\
&= \frac{1}{21} \begin{bmatrix} 5 & 4 & 8 \\ 4 & 20 & -2 \\ 8 & -2 & 17 \end{bmatrix}
\end{aligned}$$

(b) Gram-Schmidt:

$$\mathbf{U}_1 = \frac{\mathbf{X}_1}{||\mathbf{X}_1||} = \begin{bmatrix} 1/\sqrt{5} \\ 0 \\ 2/\sqrt{5} \end{bmatrix}$$

$$\mathbf{X}_2' = \mathbf{X}_2 - \mathbf{U}_1(\mathbf{U}_1^T\mathbf{X}_2) = \begin{bmatrix} 1 \\ 4 \\ 0 \end{bmatrix} - \begin{bmatrix} 1/\sqrt{5} \\ 0 \\ 2/\sqrt{5} \end{bmatrix} \left(\begin{bmatrix} 1/\sqrt{5} & 0 & 2/\sqrt{5} \end{bmatrix} \begin{bmatrix} 1 \\ 4 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 4/5 \\ 4 \\ -2/5 \end{bmatrix}$$

So

$$\mathbf{U}_2 = \sqrt{\frac{5}{84}} \begin{bmatrix} 4/5 \\ 4 \\ -2/5 \end{bmatrix} = \begin{bmatrix} 4/\sqrt{420} \\ 20/\sqrt{420} \\ -2/\sqrt{420} \end{bmatrix} = \begin{bmatrix} 2/\sqrt{105} \\ 10/\sqrt{105} \\ -1/\sqrt{105} \end{bmatrix}$$

The projection matrix is therefore

$$\mathbf{U}\mathbf{U}^T = \begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{105} \\ 0 & 10\sqrt{105} \\ 2/\sqrt{5} & -1/\sqrt{105} \end{bmatrix} \begin{bmatrix} 1/\sqrt{5} & 0 & 2/\sqrt{5} \\ 2/\sqrt{105} & 10/\sqrt{105} & -1/\sqrt{105} \end{bmatrix} = \frac{1}{21}\begin{bmatrix} 5 & 4 & 8 \\ 4 & 20 & -2 \\ 8 & -2 & 17 \end{bmatrix}$$

(c)

$$\hat{\mathbf{y}}_1 = \frac{1}{21}\begin{bmatrix} 5 & 4 & 8 \\ 4 & 20 & -2 \\ 8 & -2 & 17 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{21}\begin{bmatrix} 5 \\ 4 \\ 8 \end{bmatrix}$$

$$\hat{\mathbf{y}}_2 = \frac{1}{21}\begin{bmatrix} 5 & 4 & 8 \\ 4 & 20 & -2 \\ 8 & -2 & 17 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \frac{1}{21}\begin{bmatrix} 21 \\ 42 \\ 21 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

So the distance from $\mathbf{x}_1$ is $d(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 5/21 \\ 4/21 \\ 8/21 \end{bmatrix}) = \sqrt{16/21} = 4/\sqrt{21}$

The distance from $\mathbf{x}_2$ is $d(\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}) = 0$ □

Problem 3.3 (Problem 3)

Solution

(a) Carry out mini-classification problems for each type of iris. e.g. The first mini-classification classifies if the iris is setosa or not. In this mini-classification, assign $\mathbf{y} = 1$ if setosa, -1 otherwise. Perform this across all $\mathbf{X}$, then choose the label with the highest $\hat{\mathbf{y}}$ as the final label.

(b) For each mini-classification problem:

```python
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
from matplotlib import colors
# load data, make sure 'fisheriris.mat' is in your working directory
data = scipy.io.loadmat("fisheriris.mat")
X = data['meas']
y_text = data['species']
############################
# YOUR CODE BELOW
# Process and assign numerical values to
# 'y' according to your (a), make sure 'y' is a 1d numpy array.
TYPES = ['setosa', 'versicolor', 'virginica']
# If dimensions are mismatching, you may find 'y = y.flatten()' useful.
# Compute the least squares weights
for type_index, type in enumerate(TYPES):
    y = y_text == type
    w = np.linalg.inv(X.T @ X) @ X.T @ y
    # Compute the residuals
    r = y - X @ w
    # Make a plot
    print((X@w).T @ r)
```

prints

```
[[-1.97915772e-13]]
[[-3.9014575e-13]]
[[-7.70567792e-13]]
```

which is essentially 0.

(c)

```
# STARTER CODE
import numpy as np
import scipy.io
# load data, make sure 'fisheriris.mat' is in your working directory
data = scipy.io.loadmat("fisheriris.mat")
# training data
X = data['meas']
y_text = data['species']
y_numberized = (y_text == 'versicolor').astype(int) + (y_text == '
   virginica').astype(int)*2
# maps setosa to 0, versicolor to 1, virginica to 2
TYPES = ['setosa', 'versicolor', 'virginica']
# number of random trials
N = 10000
# array to store errors
errs = np.zeros(N)
# size of training set
num_train = 40
for i in np.arange(N):
# initialize 0-length arrays for the train and holdout indices. These
# arrays will be filled in the inner loop.
    idx_train = np.zeros(0, dtype=np.intp)
    idx_holdout = np.zeros(0, dtype=np.intp)

    # There are 3 label types and 50 samples of each type
    for label_type in range(3):
        # Choose a random ordering of the 50 samples
        r = np.random.permutation(50)
        # Add the first num_train indices of the random ordering to
        # the idx_train array
        idx_train = np.concatenate((idx_train,
        50 * label_type + r[:num_train]))
        # Add the rest of the indices to the idx_holdout array
        idx_holdout = np.concatenate((idx_holdout,
        50 * label_type + r[num_train:]))
        # divide data and labels into the train and holdout sets

    yhattypes = list()
    for type_index, type in enumerate(TYPES):
        y = (y_text == type).astype(int)
        Xt = X[idx_train]
        yt = y[idx_train]
        Xh = X[idx_holdout]
        yh = y[idx_holdout]
        w = np.linalg.inv(Xt.T @ Xt) @ Xt.T @ yt
        # Make predictions using the LS weights
        yhat = Xh @ w
        yhattypes.append(yhat)

    yhattypes = np.array(yhattypes)
    # gives best prediction
    y_pred_numberized = np.argmax(yhattypes, axis = 0)
    # # Turn the real-valued predictions into class labels
    # # Compute the errors
    errs[i] = np.sum(y_pred_numberized != y_numberized[idx_holdout])/30

print(np.average(errs))
```

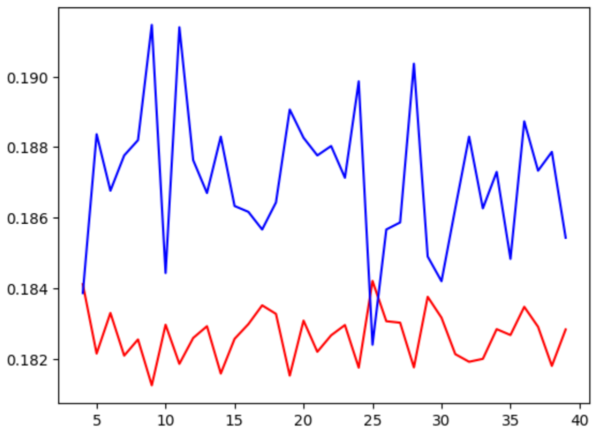
which prints out

```
0.18694666666666668
```

(d)

```
# STARTER CODE
import numpy as np
import scipy.io
import matplotlib.pyplot as plt
# load data, make sure 'fisheriris.mat' is in your working directory
data = scipy.io.loadmat("fisheriris.mat")
# training data
X = data['meas']
y_text = data['species']
y_numberized = (y_text == 'versicolor').astype(int) + (y_text == '
    virginica').astype(int)*2
# maps setosa to 0, versicolor to 1, virginica to 2
TYPES = ['setosa', 'versicolor', 'virginica']
N = 1000
# Min / Max size of the training set
min_num_train = 4
max_num_train = 40
# Arrays to store error rates
train_errs = np.zeros((max_num_train-min_num_train, N))
test_errs = np.zeros((max_num_train-min_num_train, N))
n_train_vals = np.arange(min_num_train, max_num_train)
for j, n_train in enumerate(n_train_vals):
    for i in np.arange(N):
        # initialize 0-length arrays for the train and holdout indices.
    These
        # arrays will be filled in the inner loop.
        idx_train = np.zeros(0, dtype=np.intp)
        idx_holdout = np.zeros(0, dtype=np.intp)

        # There are 3 label types and 50 samples of each type
        for label_type in range(3):
            # Choose a random ordering of the 50 samples
            r = np.random.permutation(50)
            # Add the first num_train indices of the random ordering to
            # the idx_train array
            idx_train = np.concatenate((idx_train,
            50 * label_type + r[:num_train]))
            # Add the rest of the indices to the idx_holdout array
            idx_holdout = np.concatenate((idx_holdout,
            50 * label_type + r[num_train:]))
            # divide data and labels into the train and holdout sets

        yhat_h_all = list()
        yhat_t_all = list()
        for type_index, type in enumerate(TYPES):
            y = (y_text == type).astype(int)
            Xt = X[idx_train]
            yt = y[idx_train]
            Xh = X[idx_holdout]
            yh = y[idx_holdout]
            w = np.linalg.inv(Xt.T @ Xt) @ Xt.T @ yt
            # Make predictions using the LS weights
            yhat_t = Xt @ w
            yhat_h = Xh @ w

            yhat_t_all.append(yhat_t)
            yhat_h_all.append(yhat_h)

        yhat_t_all = np.array(yhat_t_all)
        yhat_h_all = np.array(yhat_h_all)
        # gives best prediction
        yhat_t_numberized = np.argmax(yhat_t_all, axis = 0)
        yhat_h_numberized = np.argmax(yhat_h_all, axis = 0)
        # # Turn the real-valued predictions into class labels
        # # Compute the errors

        train_errs[j][i] = np.sum(yhat_t_numberized != y_numberized[
    idx_train])/len(yhat_t_numberized)
        test_errs[j][i] = np.sum(yhat_h_numberized != y_numberized[
    idx_holdout])/len(yhat_h_numberized)

# Make a plot of the train and test errors as a function of
# training set size
yplot_train = np.mean(train_errs, axis = 1)
yplot_test = np.mean(test_errs, axis = 1)

plt.plot(n_train_vals, yplot_train, color='red')
plt.plot(n_train_vals, yplot_test, color='blue')
```

which plots



The trend of test set accuracy and train set error rates seem to be in opposite directions. When train set error rate decreases (fitting better to train set), error rate for test set increases (fitting worse to test set, generalizing worse). The balance between test set and train set error rates seems to be at train set size of 25.

(e) Use training set size as 25.

```
# STARTER CODE
import numpy as np
import scipy.io
# load data, make sure 'fisheriris.mat' is in your working directory
data = scipy.io.loadmat("fisheriris.mat")
# training data
X = data['meas']
# only choosing 1st and 3rd column of X‘’
X = X[:, [0, 2]]
y_text = data['species']
y_numberized = (y_text == 'versicolor').astype(int) + (y_text == '
    virginica').astype(int)*2
# maps setosa to 0, versicolor to 1, virginica to 2
TYPES = ['setosa', 'versicolor', 'virginica']
# number of random trials
N = 10000
# array to store errors
errs = np.zeros(N)
# size of training set
num_train = 25
for i in np.arange(N):
    # initialize 0-length arrays for the train and holdout indices.
    These
    # arrays will be filled in the inner loop.
    idx_train = np.zeros(0, dtype=np.intp)
    idx_holdout = np.zeros(0, dtype=np.intp)

    # There are 3 label types and 50 samples of each type
    for label_type in range(3):
        # Choose a random ordering of the 50 samples
        r = np.random.permutation(50)
        # Add the first num_train indices of the random ordering to
        # the idx_train array
        idx_train = np.concatenate((idx_train,
        50 * label_type + r[:num_train]))
        # Add the rest of the indices to the idx_holdout array
        idx_holdout = np.concatenate((idx_holdout,
        50 * label_type + r[num_train:]))
        # divide data and labels into the train and holdout sets

    yhat_h_all = list()
    yhat_t_all = list()
    for type_index, type in enumerate(TYPES):
        y = (y_text == type).astype(int)
        Xt = X[idx_train]
        yt = y[idx_train]
        Xh = X[idx_holdout]
        yh = y[idx_holdout]
        w = np.linalg.inv(Xt.T @ Xt) @ Xt.T @ yt
        # Make predictions using the LS weights
        yhat_t = Xt @ w
        yhat_h = Xh @ w

        yhat_t_all.append(yhat_t)
        yhat_h_all.append(yhat_h)

    yhat_t_all = np.array(yhat_t_all)
    yhat_h_all = np.array(yhat_h_all)
    # gives best prediction
    yhat_t_numberized = np.argmax(yhat_t_all, axis = 0)
    yhat_h_numberized = np.argmax(yhat_h_all, axis = 0)
    # # Turn the real-valued predictions into class labels
    # # Compute the errors
    errs[i] = np.sum(yhat_h_numberized != y_numberized[idx_holdout])/30

print(np.average(errs))
```

which prints out

```
0.8332233333333335
```

Accuracy is drastically reduced when only using 2 out of 4 available features. □

Problem 3.4 (Problem 4)

Solution

(a) Projecting some points in $\mathbb{R}^3$ onto S,

$$P_S \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.4 \\ 0 \end{bmatrix}$$

$$P_S \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

And the 2 projected points are linearly independent. Since $Rank(P_S) = 2$, they span the entire subspace S. We've found a basis

$$\mathbf{X} = \begin{bmatrix} 0.2 & 0 \\ 0.4 & 0 \\ 0 & 1 \end{bmatrix}$$

(b) We can take the residual from $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$. Projecting it to S, we have

$$P_S \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.4 \\ 0.8 \\ 0 \end{bmatrix}$$

which yields residual $\begin{bmatrix} -0.4 \\ 0.2 \\ 0 \end{bmatrix}$.

Since the point lies on S, they have to satisfy

$$\begin{bmatrix} -0.4 & 0.2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = 0$$

It follows that

$$-0.4x_1 + 0.2x_2 + 0x_3 = 0$$

(c)

```python
### STARTER CODE
import numpy as np
import numpy.linalg as la
import scipy
p = np.array(
[[0.2, 0.4, 0. ],
[0.4, 0.8, 0. ],
[0. , 0. , 1. ]]
)
### YOUR CODE BELOW
print(scipy.linalg.orth(p))
```

□