PROBLEM SET 3

You may also find the code here: https://github.com/helenxtian/math-for-ml.

1. (a) We need to find two orthonormal vectors that span the plane spanned by columns of $X = \begin{bmatrix} 6 & 7 \\ 8 & 1 \\ 0 & 5 \end{bmatrix}$.

   Note that we define columns as $\mathbf{x}_1 = \begin{bmatrix} 6 \\ 8 \\ 0 \end{bmatrix}$ and $\mathbf{x}_2 = \begin{bmatrix} 7 \\ 1 \\ 5 \end{bmatrix}$. Then, we'll use the Gram-Schmidt orthogonalization process.
   First, we normalize $\mathbf{x}_1$ to get $\mathbf{u}_1$: $\|\mathbf{x}_1\| = \sqrt{6^2 + 8^2 + 0^2} = \sqrt{36 + 64} = \sqrt{100} = 10$
   $\mathbf{u}_1 = \frac{\mathbf{x}_1}{\|\mathbf{x}_1\|} = \frac{1}{10} \begin{bmatrix} 6 \\ 8 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.8 \\ 0 \end{bmatrix}$
   Next, we compute $\mathbf{v}_2 = \mathbf{x}_2 - \mathrm{proj}_{\mathbf{u}_1}(\mathbf{x}_2)$: $\mathrm{proj}_{\mathbf{u}_1}(\mathbf{x}_2) = (\mathbf{x}_2 \cdot \mathbf{u}_1)\mathbf{u}_1 = (7 \cdot 0.6 + 1 \cdot 0.8 + 5 \cdot 0) \begin{bmatrix} 0.6 \\ 0.8 \\ 0 \end{bmatrix} = 5 \begin{bmatrix} 0.6 \\ 0.8 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}$
   $\mathbf{v}_2 = \mathbf{x}_2 - \mathrm{proj}_{\mathbf{u}_1}(\mathbf{x}_2) = \begin{bmatrix} 7 \\ 1 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ -3 \\ 5 \end{bmatrix}$
   Now we normalize $\mathbf{v}_2$ to get $\mathbf{u}_2$: $\|\mathbf{v}_2\| = \sqrt{4^2 + (-3)^2 + 5^2} = \sqrt{16 + 9 + 25} = \sqrt{50} = 5\sqrt{2}$
   $\mathbf{u}_2 = \frac{\mathbf{v}_2}{\|\mathbf{v}_2\|} = \frac{1}{5\sqrt{2}} \begin{bmatrix} 4 \\ -3 \\ 5 \end{bmatrix} = \frac{1}{\sqrt{50}} \begin{bmatrix} 4 \\ -3 \\ 5 \end{bmatrix} = \begin{bmatrix} \frac{4\sqrt{2}}{10} \\ \frac{-3\sqrt{2}}{10} \\ \frac{5\sqrt{2}}{10} \end{bmatrix} = \begin{bmatrix} \frac{2\sqrt{2}}{5} \\ \frac{-3\sqrt{2}}{10} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$
   Thus, the orthonormal basis consists of $\mathbf{u}_1 = \begin{bmatrix} 0.6 \\ 0.8 \\ 0 \end{bmatrix}$ and $\mathbf{u}_2 = \begin{bmatrix} \frac{2\sqrt{2}}{5} \\ \frac{-3\sqrt{2}}{10} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$

   (b) The columns of U span the same subspace as the columns of X, but U is orthonormal. This means that $U^T U = I$ which also implies that its inverse also equates to the identity matrix. This then simplifies the projection matrix to UU^T which is less computationally intensive than non-orthogonal matrices like if computed the inverse of $X^T X$.

   (c) Given $\mathbf{y} = \begin{bmatrix} 1 \\ 6 \\ -2 \end{bmatrix}$, we can compute $\hat{\mathbf{y}}$ as follows. Remember that since U has orthonormal columns, $U^T U = I$, so $\hat{\mathbf{y}} = UU^T\mathbf{y}$.

First, we compute $U^T\mathbf{y}$ (calculations are below):

$$
\begin{aligned}
U^T\mathbf{y} &= \begin{bmatrix} \frac{3}{5} & \frac{4}{5} & 0 \\ \frac{2\sqrt{2}}{5} & \frac{-3\sqrt{2}}{10} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 6 \\ -2 \end{bmatrix} \\
&= \begin{bmatrix} \frac{3}{5}\cdot 1 + \frac{4}{5}\cdot 6 + 0\cdot(-2) \\ \frac{2\sqrt{2}}{5}\cdot 1 + \frac{-3\sqrt{2}}{10}\cdot 6 + \frac{\sqrt{2}}{2}\cdot(-2) \end{bmatrix} \\
&= \begin{bmatrix} \frac{3}{5} + \frac{24}{5} \\ \frac{2\sqrt{2}}{5} + \frac{-18\sqrt{2}}{10} + \frac{-\sqrt{2}}{1} \end{bmatrix} \\
&= \begin{bmatrix} \frac{27}{5} \\ \frac{4\sqrt{2}}{10} + \frac{-18\sqrt{2}}{10} + \frac{-10\sqrt{2}}{10} \end{bmatrix} \\
&= \begin{bmatrix} \frac{27}{5} \\ \frac{4\sqrt{2}-18\sqrt{2}-10\sqrt{2}}{10} \end{bmatrix} \\
&= \begin{bmatrix} \frac{27}{5} \\ \frac{-24\sqrt{2}}{10} \end{bmatrix}
\end{aligned}
$$

Now we compute $UU^T\mathbf{y}$ (calculations are below):

$$
\begin{aligned}
\hat{\mathbf{y}} = UU^T\mathbf{y} &= \begin{bmatrix} \frac{3}{5} & \frac{2\sqrt{2}}{5} \\ \frac{4}{5} & \frac{-3\sqrt{2}}{10} \\ 0 & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \frac{27}{5} \\ \frac{-24\sqrt{2}}{10} \end{bmatrix} \\
&= \begin{bmatrix} \frac{3}{5}\cdot\frac{27}{5} + \frac{2\sqrt{2}}{5}\cdot\frac{-24\sqrt{2}}{10} \\ \frac{4}{5}\cdot\frac{27}{5} + \frac{-3\sqrt{2}}{10}\cdot\frac{-24\sqrt{2}}{10} \\ 0\cdot\frac{27}{5} + \frac{\sqrt{2}}{2}\cdot\frac{-24\sqrt{2}}{10} \end{bmatrix} \\
&= \begin{bmatrix} \frac{81}{25} + \frac{2\sqrt{2}\cdot(-24\sqrt{2})}{50} \\ \frac{108}{25} + \frac{-3\sqrt{2}\cdot(-24\sqrt{2})}{100} \\ 0 + \frac{\sqrt{2}\cdot(-24\sqrt{2})}{20} \end{bmatrix} \\
&= \begin{bmatrix} 1.32 \\ 5.76 \\ -2.4 \end{bmatrix}
\end{aligned}
$$

2. (a) We need to find a basis for the subspace $S = \{\mathbf{x} \in \mathbb{R}^3 : 2x_1 - x_2 + 3x_3 = 0\}$. To find a basis, we can express one variable in terms of the others such that $2x_1 - x_2 + 3x_3 = 0 \Rightarrow x_2 = 2x_1 + 3x_3$. Here, we choose x_1 and x_3 as free variables, means leads to a a non-orthonormal basis for S is $\{\mathbf{v}_1, \mathbf{v}_2\} = \left\{ \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix} \right\}$.

   To find the projection matrix, we use $P = X(X^TX)^{-1}X^T$ where $X = [\mathbf{v}_1, \mathbf{v}_2]$ such that $X = \begin{bmatrix} 1 & 0 \\ 2 & 3 \\ 0 & 1 \end{bmatrix}$.

From here, we can calculate $P = X(X^TX)^{-1}X^T = \begin{bmatrix} 1 & 0 \\ 2 & 3 \\ 0 & 1 \end{bmatrix} (\begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 3 \\ 0 & 1 \end{bmatrix})^{-1} \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 1 \end{bmatrix}$.

From hand calculations, note that:

$(X^TX)^{-1} = \frac{1}{\det(X^TX)} \begin{bmatrix} 10 & -6 \\ -6 & 5 \end{bmatrix} = \frac{1}{50-36} \begin{bmatrix} 10 & -6 \\ -6 & 5 \end{bmatrix} = \frac{1}{14} \begin{bmatrix} 10 & -6 \\ -6 & 5 \end{bmatrix}$

$\implies X(X^TX)^{-1} = \begin{bmatrix} 1 & 0 \\ 2 & 3 \\ 0 & 1 \end{bmatrix} \frac{1}{14} \begin{bmatrix} 10 & -6 \\ -6 & 5 \end{bmatrix} = \frac{1}{14} \begin{bmatrix} 10 & -6 \\ 2 & 3 \\ -6 & 5 \end{bmatrix}$

This leads to $P = X(X^TX)^{-1}X^T = \frac{1}{14} \begin{bmatrix} 10 & -6 \\ 2 & 3 \\ -6 & 5 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 1 \end{bmatrix}$

$= \frac{1}{14} \begin{bmatrix} 10 \cdot 1 + (-6) \cdot 0 & 10 \cdot 2 + (-6) \cdot 3 & 10 \cdot 0 + (-6) \cdot 1 \\ 2 \cdot 1 + 3 \cdot 0 & 2 \cdot 2 + 3 \cdot 3 & 2 \cdot 0 + 3 \cdot 1 \\ (-6) \cdot 1 + 5 \cdot 0 & (-6) \cdot 2 + 5 \cdot 3 & (-6) \cdot 0 + 5 \cdot 1 \end{bmatrix}$

$= \frac{1}{14} \begin{bmatrix} 10 & 2 & -6 \\ 2 & 13 & 3 \\ -6 & 3 & 5 \end{bmatrix}$

Therefore, the projection matrix is: $P = \frac{1}{14} \begin{bmatrix} 10 & 2 & -6 \\ 2 & 13 & 3 \\ -6 & 3 & 5 \end{bmatrix} = \begin{bmatrix} \frac{5}{7} & \frac{1}{7} & \frac{-3}{7} \\ \frac{1}{7} & \frac{13}{14} & \frac{3}{14} \\ \frac{-3}{7} & \frac{3}{14} & \frac{5}{14} \end{bmatrix}$

(b) Starting with our non-orthonormal basis $\{\mathbf{v}_1, \mathbf{v}_2\} = \left\{ \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix} \right\}$,

we apply the Gram-Schmidt process.

First, we normalize $\mathbf{v}_1$ to get $\mathbf{u}_1$: $\|\mathbf{v}_1\| = \sqrt{1^2 + 2^2 + 0^2} = \sqrt{1+4} = \sqrt{5}$

$\mathbf{u}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|} = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \\ 0 \end{bmatrix}$

Next, we compute $\mathbf{w}_2 = \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_2)$:

$\text{proj}_{\mathbf{u}_1}(\mathbf{v}_2) = (\mathbf{v}_2 \cdot \mathbf{u}_1)\mathbf{u}_1 = \left(0 \cdot \frac{1}{\sqrt{5}} + 3 \cdot \frac{2}{\sqrt{5}} + 1 \cdot 0\right) \mathbf{u}_1 = \frac{6}{\sqrt{5}} \mathbf{u}_1 = \frac{6}{\sqrt{5}} \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \\ 0 \end{bmatrix} = \frac{6}{5} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{6}{5} \\ \frac{12}{5} \\ 0 \end{bmatrix}$

$\mathbf{w}_2 = \mathbf{v}_2 - \text{proj}_{\mathbf{u}_1}(\mathbf{v}_2) = \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix} - \begin{bmatrix} \frac{6}{5} \\ \frac{12}{5} \\ 0 \end{bmatrix} = \begin{bmatrix} -\frac{6}{5} \\ \frac{15-12}{5} \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{6}{5} \\ \frac{3}{5} \\ 1 \end{bmatrix}$

Now we normalize $\mathbf{w}_2$ to get $\mathbf{u}_2$: $\|\mathbf{w}_2\| = \sqrt{\left(-\frac{6}{5}\right)^2 + \left(\frac{3}{5}\right)^2 + 1^2} = \sqrt{\frac{36}{25} + \frac{9}{25} + 1} = \sqrt{\frac{45}{25} + \frac{25}{25}} = \sqrt{\frac{70}{25}} = \frac{\sqrt{70}}{5}$

$$\mathbf{u}_2 = \frac{\mathbf{w}_2}{\|\mathbf{w}_2\|} = \frac{5}{\sqrt{70}} \begin{bmatrix} -\frac{6}{5} \\ \frac{3}{5} \\ 1 \end{bmatrix} = \frac{1}{\sqrt{70}} \begin{bmatrix} -6 \\ 3 \\ 5 \end{bmatrix}$$

Therefore, an orthonormal basis for S is $\{\mathbf{u}_1, \mathbf{u}_2\} = \left\{ \begin{bmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \\ 0 \end{bmatrix}, \frac{1}{\sqrt{70}} \begin{bmatrix} -6 \\ 3 \\ 5 \end{bmatrix} \right\}$

To find the projection matrix using this orthonormal basis, we use $P = UU^T$ where $U = [\mathbf{u}_1, \mathbf{u}_2]$.

$$U = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{-6}{\sqrt{70}} \\ \frac{2}{\sqrt{5}} & \frac{3}{\sqrt{70}} \\ 0 & \frac{5}{\sqrt{70}} \end{bmatrix}$$

$$U^T = \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ \frac{-6}{\sqrt{70}} & \frac{3}{\sqrt{70}} & \frac{5}{\sqrt{70}} \end{bmatrix}$$

$$\begin{aligned} P = UU^T &= \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{-6}{\sqrt{70}} \\ \frac{2}{\sqrt{5}} & \frac{3}{\sqrt{70}} \\ 0 & \frac{5}{\sqrt{70}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ \frac{-6}{\sqrt{70}} & \frac{3}{\sqrt{70}} & \frac{5}{\sqrt{70}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{5} + \frac{36}{70} & \frac{2}{5} - \frac{18}{70} & \frac{-30}{70} \\ \frac{2}{5} - \frac{18}{70} & \frac{4}{5} + \frac{9}{70} & \frac{15}{70} \\ \frac{-30}{70} & \frac{15}{70} & \frac{25}{70} \end{bmatrix} \\ &= \begin{bmatrix} \frac{14}{70} + \frac{36}{70} & \frac{28}{70} - \frac{18}{70} & \frac{-30}{70} \\ \frac{28}{70} - \frac{18}{70} & \frac{56}{70} + \frac{9}{70} & \frac{15}{70} \\ \frac{-30}{70} & \frac{15}{70} & \frac{25}{70} \end{bmatrix} \\ &= \frac{1}{70} \begin{bmatrix} 50 & 10 & -30 \\ 10 & 65 & 15 \\ -30 & 15 & 25 \end{bmatrix} \end{aligned}$$

Therefore, the projection matrix using the orthonormal basis is $P = \frac{1}{70} \begin{bmatrix} 50 & 10 & -30 \\ 10 & 65 & 15 \\ -30 & 15 & 25 \end{bmatrix} = \frac{1}{14} \begin{bmatrix} 10 & 2 & -6 \\ 2 & 13 & 3 \\ -6 & 3 & 5 \end{bmatrix} = \begin{bmatrix} \frac{5}{7} & \frac{1}{7} & \frac{-3}{7} \\ \frac{1}{7} & \frac{13}{14} & \frac{3}{14} \\ \frac{-3}{7} & \frac{3}{14} & \frac{5}{14} \end{bmatrix}$

(c) To calculate the distance of $\mathbf{x} = \begin{bmatrix} 1 \\ 5 \\ 1 \end{bmatrix}$ to S, we use $d(\mathbf{x}, S) = \|\mathbf{x} - P\mathbf{x}\|$

First, we compute $P\mathbf{x}$.

$$
\begin{aligned}
& P\mathbf{x} \\
&= \frac{1}{70}\begin{bmatrix} 50\cdot 1 + 10\cdot 5 + (-30)\cdot 1 \\ 10\cdot 1 + 65\cdot 5 + 15\cdot 1 \\ (-30)\cdot 1 + 15\cdot 5 + 25\cdot 1 \end{bmatrix} \\
&= \frac{1}{70}\begin{bmatrix} 50 + 50 - 30 \\ 10 + 325 + 15 \\ -30 + 75 + 25 \end{bmatrix} \\
&= \begin{bmatrix} 1 \\ 5 \\ 1 \end{bmatrix}
\end{aligned}
$$

Now we compute $\mathbf{x} - P\mathbf{x}$: $\mathbf{x} - P\mathbf{x} = \begin{bmatrix} 1 \\ 5 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 5 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. Therefore, $d(\mathbf{x}, S) = \|\mathbf{x} - P\mathbf{x}\| = \| \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \| = 0$. The distance is 0, which means $\mathbf{x}$ is already in the subspace S.

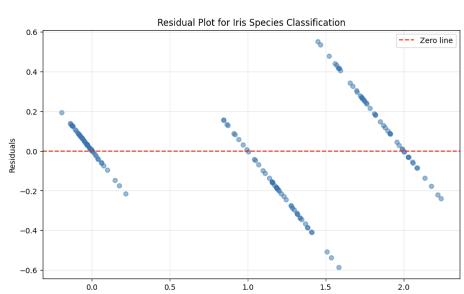
3. (a) One possibility is to assign numerical values to the labels. For example: setsoa = 0, versicolor = 1, virginica = 2. Then it becomes standard least-squares. The standard least-square problem includes putting the measurements of each flower into a data table X with one flower per row. Then, we have to find the weight vector that tracks y via X using least squares. Then, the prediction is the dot product of the measurement row and w and then round to 0, 1, 2 (solving the real number problem).

   (b)

```python
# STARTER CODE for Question 3b
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.
```

```python
# If dimensions are mismatching, you may find 'y = y.flatten()' useful.
y = np.array([0]*50 + [1]*50 + [2]*50) # setosa=0, versicolor=1, virginica=2
y = y.flatten()

# Compute the least squares weights
XtX = X.T @ X
Xty = X.T @ y
w = np.linalg.solve(XtX, Xty)
print("Least squares weights:", w)

# Compute the residuals
y_hat = X.dot(w)
residuals = y - y_hat

# Check orthogonality (X^T r      0)
orthogonality = X.T.dot(residuals)
print("X^T    residuals =", orthogonality) # check if close to 0

# Make a plot
plt.figure(figsize=(10, 6))
plt.scatter(y_hat, residuals, alpha=0.5)
plt.axhline(0, color='red', linestyle='--', label='Zero line')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot for Iris Species Classification")
plt.grid(True, alpha=0.3)
plt.legend()
```

```
Least squares weights: [-0.0844926  -0.02356211
 0.22487123  0.59972247]
X^T    residuals = [9.94759830e-14 2.09610107e-13
 1.28785871e-13 1.16351373e-13]
```



(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']

############################
# YOUR CODE BELOW
# Process and assign numerical values to
# 'y' according to your (a), make sure 'y' is a 1d
    numpy array.
# If dimensions are mismatching, you may find 'y = y
    .flatten()' useful.

# number of random trials
N = 10_000
# 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:]))
```

```python
    # divide data and labels into the train and holdout sets
    Xt = X[idx_train]
    yt = y[idx_train]
    Xh = X[idx_holdout]
    yh = y[idx_holdout]

    ########################
    # YOUR CODE BELOW
    XtX = Xt.T @ Xt
    Xty = Xt.T @ yt
    w = np.linalg.solve(XtX, Xty)

    # Make predictions using the LS weights on holdout set
    y_pred = Xh @ w

    # Turn the real-valued predictions into class labels
    y_pred_class = np.round(y_pred).astype(int)

    # Compute the errors
    errs[i] = np.mean(y_pred_class != yh)

avg_error = np.mean(errs)
print(f"Average test error over {N} trials: {avg_error:.3f}")
```

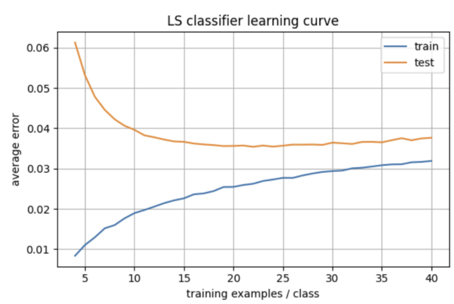
```
Average test error over 10000 trials: 0.038
```

(d)

```python
# STARTER CODE for Question 3d
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']
############################
# YOUR CODE BELOW
# Process and assign numerical values to
# 'y' according to your (a), make sure 'y' is a 1d numpy array.
# If dimensions are mismatching, you may find 'y = y.flatten()' useful.
```

```
y = np.array([0]*50 + [1]*50 + [2]*50)
y = y.flatten()

# number of random trials
N = 1_000
# 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
    Xt = X[idx_train]
    yt = y[idx_train]
    Xh = X[idx_holdout]
    yh = y[idx_holdout]
    ########################
    # YOUR CODE BELOW
```

```
    XtX = Xt.T @ Xt
    Xty = Xt.T @ yt
    w = np.linalg.solve(XtX, Xty)

    # Make predictions on both train and test sets
    y_pred_train = np.round(Xt @ w).astype(int)
    y_pred_test = np.round(Xh @ w).astype(int)

    train_errs[j,i] = np.mean(y_pred_train != yt)
    test_errs[j,i] = np.mean(y_pred_test != yh)

###################
# YOUR CODE BELOW
# Make a plot of the train and test errors as a
    function of
# training set size
plt.figure(figsize=(10,6))
plt.plot(n_train_vals, train_errs.mean(1), label='
    train')
plt.plot(n_train_vals, test_errs.mean(1), label='
    test')
plt.xlabel('training examples / class')
plt.ylabel('average error')
plt.title('LS classifier learning curve')
plt.grid(True)
plt.legend()
plt.tight_layout()
```



As training size increases, we see that the training error decreases because the model better fits the training data. Additionally, the test error also decreases but stabilizes after a point, indicating diminishing returns. We see as the size increases, the gap between train/test errors narrows, suggesting improved generalization. The test error stabilizes around 20-30 examples per class, meaning the optimal choice would be around there (25) which implies there are 75 total (since there are 3 classes). We choose this because we want to avoid overfitting and underfitting the model to the training data.

(e)

```python
### CODE for Question 3e
# Select only sepal length (column 0) and petal
    length (column 2)
X_reduced = X[:, [0, 2]]  # Keep only columns 0 and
    2

N = 1_000
num_train = 40
errs = np.zeros(N)

for i in np.arange(N):
    idx_train = np.zeros(0, dtype=np.intp)
    idx_holdout = np.zeros(0, dtype=np.intp)
    for label_type in range(3):
        r = np.random.permutation(50)
        idx_train = np.concatenate((idx_train,
                                    50 * label_type +
                                        r[:num_train]))
        idx_holdout = np.concatenate((idx_holdout,
                                      50 * label_type
                                          + r[num_train
                                          :]))

    # Get training and holdout sets using reduced
        features
    Xt = X_reduced[idx_train]
    yt = y[idx_train]
    Xh = X_reduced[idx_holdout]
    yh = y[idx_holdout]

    # Compute LS weights
    XtX = Xt.T @ Xt
    Xty = Xt.T @ yt
    w = np.linalg.solve(XtX, Xty)

    y_pred = np.round(Xh @ w).astype(int)
    errs[i] = np.mean(y_pred != yh)

print(f"Average test error using only sepal length
    and petal length: {np.mean(errs):.3f}")
```

```
Average test error using only sepal length and petal
     length: 0.035
```

4. (a) Let the columns of X be $\mathbf{v}_1 = \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix}$, $\mathbf{v}_2 = \begin{bmatrix} 4 \\ 6 \\ 2 \end{bmatrix}$, $\mathbf{v}_3 = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$. We

can row reduce to determine the rank of X

$$\begin{bmatrix} 0 & 4 & 3 \\ 3 & 6 & 0 \\ 1 & 2 & 0 \end{bmatrix} \implies \begin{bmatrix} 1 & 2 & 0 \\ 3 & 6 & 0 \\ 0 & 4 & 3 \end{bmatrix} \implies \begin{bmatrix} 1 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 4 & 3 \end{bmatrix}.$$

Since the rank is 2, any two linearly independent columns form a basis. We choose $\left\{ \begin{bmatrix} 0 \\ 3 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 4 \\ 6 \\ 2 \end{bmatrix} \right\}$ is a basis for S.

(b) We want a nonzero vector $\mathbf{n} = (a, b, c)$ such that $\mathbf{n}^\top \mathbf{v}_1 = 0, \quad \mathbf{n}^\top \mathbf{v}_2 = 0$. This leads to the system

$$\begin{aligned} 3b + c &= 0 \\ 4a + 6b + 2c &= 0 \end{aligned}$$

Substitute $c = -3b$ into the second equation to get $4a + 6b - 6b = 0 \Rightarrow a = 0$. So the normal vector is $\mathbf{n} = \begin{bmatrix} 0 \\ b \\ -3b \end{bmatrix} = b \begin{bmatrix} 0 \\ 1 \\ -3 \end{bmatrix}$. Thus, the subspace S is defined by $x_2 - 3x_3 = 0$ such that $a = 0, b = 1, c = -3$.

(c)

```python
### STARTER CODE for Question 4c
import numpy as np
import numpy.linalg as la
p = np.array(
  [[0, 4, 3],
  [3, 6, 0],
  [1, 2, 0]]
)
### YOUR CODE BELOW
c1 = p[:, 0]
c2 = p[:, 1]
u1 = c1 / la.norm(c1)
proj = np.dot(u1, c2) * u1
v2 = c2 - proj
u2 = v2 / la.norm(v2)
U = np.column_stack((u1, u2))
print("result of GS: \n", U)
```

```
result of GS:
 [[0.00000000e+00 1.00000000e+00]
 [9.48683298e-01 2.22044605e-16]
 [3.16227766e-01 0.00000000e+00]]
```