Homework 2

Due: **September 24th, 5pm** (late submission until September 27nd, 5pm -- no submission possible afterwards)

Written assignment: 13 points

Coding assignment: 16 points

Project report: 6 points

Name: [Yawen Tan]

Link to the github repo: [https://github.com/IsabellaTan/Brown-DATA2060-HW2.git]

Written Assignment

Problem 1: Halfspaces (6 points)

Let $X = \{0, 1\}_d$ and $Y = \{-1, 1\}$. Consider the example below and answer the questions:

Example:

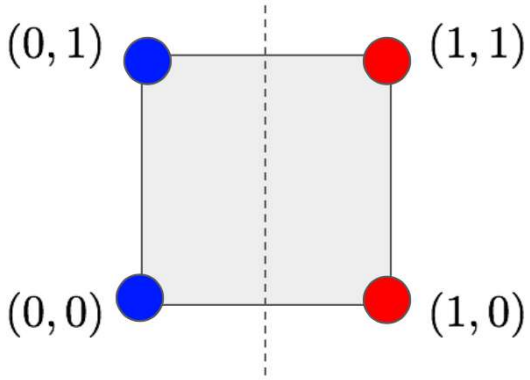
- Output is 1 if and only if x_1 is 0 (negation of x_1 or $\neg x_1$).
- Labeling function:

$$f(\mathbf{x}) = \neg x_1$$

- One possible halfspace *h:* $X \to Y$:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(-x_1 + 1/2)$$

- where the weight vector is w = (-1, 0, 0, ..., 0) and the bias term is 1/2.
- A graphical representation is shown below:



Questions:

Define a halfspace *h:* $X \to Y$ for the following functions:

1. **Conjunction:** Output is 1 if and only if all *d* attributes are 1.
2. **Majority:** Output is 1 if and only if more than half of the *d* attributes are 1.

Make sure to explain why you have chosen your weights and why they exhibit the desired behavior. You can assume that there is a bias term and that you can express your weights and bias in terms of *d*.

Note: The sign function can be considered as

sign

$$(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \text{ (Should not happen to achieve } 100\backslash\% \text{ accuracy)} \\ -1 & \text{if } x < 0 \end{cases}$$

for all problems on this homework.

Solution:

1.

Define $h(x) = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_dx_d + b$

Let w_i = 1 for i = 1,2,...,d.

Because we would like to determine when all d attributes are 1, output is 1, then when w=1, the sum of all d attributes are d; otherwise the sum of all d attributes are less than d.

Let b = -d + 0.5.

Because when all d attibutes are 1, the sum of xi is d, and we want the output 1 if and only if sum of x_i is d, so that d + b > 0 and d - 1 + b < 0. Then -d < b < -d + 1. We choose b = -d + 0.5

Thus, $w_1, \ldots, w_d$ = 1, and b = -d + 0.5,
$h(x) = sign(x_1 + x_2 + \ldots + x_d - d + 0.5)$

2.

Define $h(x) = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_dx_d + b$

Let w_i = 1 for i = 1,2,...,d.

Because we would like to determine when more than half d attributes are 1, output is 1, then when w=1, the sum of more than half d attributes are greater than d/2; otherwise the sum of more than half d attributes are less than or equal to d/2.

And we can express sum of more than half d attributes are greater than d/2 as the sum is greater or equal to ⌊d/2⌋ + 1 (because we need to consider if d is odd or even).

Let b = -⌊d/2⌋ - 0.5

Because the output is 1 if and only if sum of x_i is greater or equal to ⌊d/2⌋ + 1, so that ⌊d/2⌋ + 1 + b > 0 and ⌊d/2⌋ + b < 0. Then -⌊d/2⌋ - 1 < b < -⌊d/2⌋. We choose b= -⌊d/2⌋ - 0.5

Thus, $w_1, \ldots, w_d$ = 1, and b = -⌊d/2⌋ - 0.5,
$h(x) = sign(x_1 + x_2 + \ldots + x_d - \lfloor d/2 \rfloor - 0.5)$

Problem 2: More Halfspaces (3 points)

Consider the example below and answer the question:

Example:

- Here is the proof that XOR (exclusive or), $f^C(\mathbf{x}) = x_1 \text{ xor } x_2$, cannot be represented with a halfspace. You can use a similar argument for this problem.

$$h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(w_1x_1 + w_2x_2 + b)$$

- When x_1 and x_2 are both 0, exclusive or should be false (negative sign):

$$(0,0) : b < 0 \implies -b > 0$$

- When exactly one of x_1 or x_2 is 1, then exclusive or should be true (positive sign):

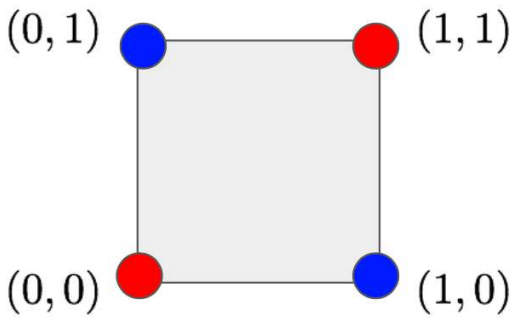
$$(1,0) : w_1 + b > 0$$

$$(0,1) : w_2 + b > 0$$

- When x_1 and x_2 are both 1, exclusive or should be false (negative sign):

$$(1,1) : w_1 + w_2 + b < 0$$

- The sum of the first three constraints gives: $w_1 + w_2 + b > 0$, which contradicts with the rule for $(1, 1)$! So we know XOR cannot be represented as a halfspace. A graphical representation is shown below:



Question:

Consider the function $h_{equiv} : \{0, 1\}^2 \to \{-1, 1\}$ (ie. $x_1, x_2 \in \{0, 1\}$) defined as

$$h_{equiv}(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 = x_2, \\ -1 & \text{otherwise.} \end{cases}$$

Show that h_{equiv} cannot be represented as a halfspace with a bias term.

Solution:

Define $h_{equiv} = \text{sign}(w_1 x_1 + w_2 x_2 + b)$

When x_1 and x_2 are both 0, function should be true (positive sign):

$$(0, 0) : b > 0 \implies -b < 0$$

When exactly one of x_1 or x_2 is 1, then function should be false (negative sign):

$$(1, 0) : w_1 + b < 0$$

$$(0, 1) : w_2 + b < 0$$

When x_1 and x_2 are both 1, function should be true (positive sign):

$$(1, 1) : w_1 + w_2 + b > 0$$

The sum of the first three constraints gives: $w_1 + w_2 + b < 0$, which contradicts with the rule for $(1, 1)$. So we know h_{equiv} cannot be represented as a halfspace with a bias term.

Problem 3: Decision Boundaries (4 points)

Consider an arbitrary halfspace classifier of the form $h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$ where $h : \mathbb{R}^n \to \{-1, 1\}, w, x \in \mathbb{R}^n$, and there is no bias term. Prove that the distance from an arbitrary example $\mathbf{x}$ to the decision boundary defined by $\mathbf{w}$ is:

$$\frac{|\langle \mathbf{w}, \mathbf{x} \rangle|}{||\mathbf{w}||_2}$$

The distance from an example $\mathbf{x}$ to the decision boundary is defined as the distance from the example to the point $\mathbf{a}$ on the decision boundary with minimum distance to the example $\mathbf{x}$. $||\mathbf{w}||_2$ denotes the L^2 norm, given by $||\mathbf{w}||_2 = \sqrt{w_1^2 + w_2^2 + \ldots + w_n^2}$.

Reminder: The decision boundary of a halfspace classifier is the set of points in X where the classifier's output changes from -1 to 1, i.e., the solution to $\langle \mathbf{w}, \mathbf{x} \rangle = 0$.

Solution:

Let the decision boundary be $H = z \in R^n : \langle w, z \rangle = 0$.

Let $a \in H$ be the point on the decision boundary that is closest to x.

The vector from a to x, which is x−a, must be orthogonal to the decision boundary. So it is parallel to the normal vector w (because ⟨w,z⟩=0). Then there exists a scalar $\lambda \in R$ such that $x - a = \lambda w$. So $a = x - \lambda w$.

Because a lies on decision boundary H, it satisfies ⟨w,a⟩=0. Then we substitute $a = x - \lambda w$ and get $\langle w, x - \lambda w \rangle = 0$

Since $\langle w, x - \lambda w \rangle = 0$, then $\langle w, x \rangle - \lambda \langle w, w \rangle = 0$, so that $\lambda = \frac{<w,x>}{<w,w>} = \frac{<w,x>}{||w||_2^2}$.

The distance d from x to the boundary decision H is $d = ||x - a||_2$.

Since $x - a = \lambda w$, then $d = ||x - a||_2 = ||\lambda w||_2 = |\lambda| ||w||_2$

We substitute $\lambda = \frac{<w,x>}{||w||_2^2}$ into $d = |\lambda| ||w||_2$, then we can get $d = |\frac{<w,x>}{||w||_2^2}| ||w||_2$.
After simplify, $d = |\frac{<w,x>}{||w||_2}|$

Thus, we have proved the distance from an arbitrary example $\mathbf{x}$ to the decision boundary defined by $\mathbf{w}$ is: $\frac{|\langle \mathbf{w}, \mathbf{x} \rangle|}{||\mathbf{w}||_2}$.

Coding Assignment (16 points)

Run the environment test below, make sure you get all green check, if not, you will lose 2 points for each red flag.

In [4]:
```
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"
```

```
try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.12.11 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
                  % (lib, min_ver, ver))
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.12.11"):
    print(OK, "Python version is %s" % pyversion)
elif pyversion < Version("3.12.11"):
    print(FAIL, "Python version 3.12.11 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'matplotlib': "3.10.5", 'numpy': "2.3.2",'sklearn': "1.7.1",
                'pandas': "2.3.2", 'pytest': "8.4.1", 'torch':"2.7.1"}

# now the dependencies
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.12.11

[ OK ] matplotlib version 3.10.5 is installed.
[ OK ] numpy version 2.3.2 is installed.
[ OK ] sklearn version 1.7.1 is installed.
[ OK ] pandas version 2.3.2 is installed.
[ OK ] pytest version 8.4.1 is installed.
[ OK ] torch version 2.7.1 is installed.
```

Introduction

After months of studying to become a pastry chef, Andras decided to take some time off and host a wine night for his friends. Unfortunately, with all the time it

takes to make desserts, he hasn't had time to become a wine connoisseur, so he's asked you to help him choose which wines to stock, with the help of machine learning!

From his group of friends, we've collected quite a lot of data, and need your help to electronically determine how each person rated a wine on a scale of 1 to 10.

In this assignment, you'll implement linear regression and use your model to predict wine quality.

Stencil Code & Data

We have provided the following stencil code within notebok:

- `Models` contains the `Linear Regression` model you will be implementing.
- `Check Model` contains a series of tests to ensure you are coding your model properly.
- `Main` is the entry point of program which will read in the dataset, run the model, and print the results.

You should not modify any code in the `main`. If you do for debugging or other purposes, please make sure any additions are commented out in the final handin. The autograder will run on an unmodified version of `main`. All the functions you need to fill in reside in this notebook, marked by `TODO`s. You can see a full description of them in the section below.

Datasets : UCI Wine Quality

For the Linear Regression model, you will be using the UCI Wine Quality Dataset, which contains information about various attributes of a wine and its corresponding quality rating (out of 10). It includes 4898 examples, which will be split into training and testing datasets using the `sklearn` library (you do not have to worry about this, as we have implemented it for you - you will be doing this in the next assignment!). Each example contains 12 attributes, and your model will train on the first 11 attributes (and a 12th bias term) to predict the last value. More information about the dataset can be found at https://archive.ics.uci.edu/ml/datasets/Wine+Quality.

In `Model`, there are three functions you will implement. They are:

- `squared_error()` is a helper function that calculates the sum squared difference between two arrays.
- `train()` uses matrix inversion to find the optimal set of weights for the given data.
- `predict()` predicts the values of test data points using the trained weights.

In addition, three methods are provided for you. You should not change them.

- `loss()` computes the squared error loss of the predicted labels over a dataset.
- `average_loss()` computes the average squared error loss per prediction over a dataset.

Note: You are not allowed to use any off-the-shelf packages that have already implemented these models, such as scikit-learn or any linear regression functions. We're asking you to implement them yourself.

Linear Regression

Linear Regression learns linear functions of the inputs:

$$h_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

Note: the bias term can be included here by padding the data with an extra feature of 1. This has been already done for you in the stencil.

As we are using squared loss, the ERM hypothesis has weights

$$\mathbf{w} = \text{argmin}_{\mathbf{w}} \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2$$

Squared Loss

For this assignment, we will be evaluating and training the Linear Regression model using sum squared loss (or L2 loss). Recall that the L2 loss function is defined as:

$$L_S(h_{\mathbf{w}}) = \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2$$

where y_i is the target value of i^{th} sample and $h_{\mathbf{w}}(\mathbf{x}_i)$ is the predicted value of that sample given the learned model weights. Your model should seek to minimize this loss through matrix inversion to learn the ERM hypothesis.

Matrix Inversion

You can assume that the examples in the training data are linearly independent. In that case, we showed in lecture that you can use matrix inversion to compute the vector of weights $\mathbf{w}$ that minimizes the squared loss. The equation to find $\mathbf{w}$, for a set of data points X and their labels $\mathbf{y}$ is

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

Note: X here is a matrix of examples stacked row-wise (i.e. $\mathbf{x_1}$ is the first row of X and so on). In lecture we saw that with X as a matrix of examples stacked column-wise (i.e. $\mathbf{x_1}$ is the first column of X and so on), the equation is equivalently:

$$\mathbf{w} = A^{-1}\mathbf{b} = (XX^T)^{-1}X\mathbf{y}$$

Implement the solution for $\mathbf{w}$ in `LinearRegression`, using the `np.linalg.pinv` function to calculate matrix inverses.

Model

In [ ]:
```python
import random
import numpy as np


def squared_error(predictions, Y):
    '''
    Computes sum squared loss (the L2 loss) between true values, Y,
    and predictions.
    @params:
        Y: A 1D Numpy array with real values (float64)
        predictions: A 1D Numpy array of the same size of Y
    @return:
        sum squared loss (the L2 loss) using predictions for Y.
    '''
    # [TODO]
    # Compute the error
    error = Y - predictions
    # Square the error
    squared_error = error ** 2
    # Return the sum of squared errors
    return np.sum(squared_error)


class LinearRegression:
    '''
    LinearRegression model that minimizes squared error using matrix inversion.
    '''
    def __init__(self, n_features):
        '''
        @attrs:
            n_features: the number of features in the regression problem
            weights: The weights of the linear regression model.
        '''
        # An extra feature added for the bias value
        self.n_features = n_features + 1
        self.weights = np.zeros(n_features + 1)

    def train(self, X, Y):
        '''
        Trains the LinearRegression model by finding the optimal set of weights
        using matrix inversion.
        @params:
            X: 2D Numpy array where each row contains an example,
            padded by 1 column for the bias
            Y: 1D Numpy array containing the corresponding values
```

```
            for each example
        @return:
            the weights of the regression model
        '''
        # [TODO]
        # Compute X^T X
        XtX = np.dot(X.T, X)
        # Compute X^T Y
        XtY = np.dot(X.T, Y)
        # Compute weight w = (X^T X)^(-1) X^T Y
        self.weights = np.dot(np.linalg.pinv(XtX), XtY)
        return self.weights



    def predict(self, X):
        '''
        Returns predictions of the model on a set of examples X.
        @params:
            X: a 2D Numpy array where each row contains an example,
            padded by 1 column for the bias
        @return:
            A 1D Numpy array with one element for each row in X
            containing the predicted value.
        '''
        # [TODO]
        return np.dot(X, self.weights)


    def loss(self, X, Y):
        '''
        Returns the total squared error on some dataset (X, Y).
        @params:
            X: 2D Numpy array where each row contains an example,
            padded by 1 column for the bias
            Y: 1D Numpy array containing the corresponding values
            for each example
        @return:
            A float number which is the squared error of the model
            on the dataset
        '''
        predictions = self.predict(X)
        return squared_error(predictions, Y)


    def average_loss(self, X, Y):
        '''
        Returns the mean squared error on some dataset (X, Y).
        MSE = Total squared error/# of examples
        @params:
            X: 2D Numpy array where each row contains an example,
            padded by 1 column for the bias
            Y: 1D Numpy array containing the corresponding values
            for each example
        @return:
            A float number which is the mean squared error of
            the model on the dataset
        '''
        return self.loss(X, Y)/X.shape[0]
```

Check Model

In [ ]:
```
import pytest

# Test Squared Error
assert squared_error(np.array([0]),np.array([0])) == 0
assert squared_error(np.array([2,-2,3,1]),np.array([1,1,2,2])) == 12

# Create Test Model
test_model1 = LinearRegression(2)
x1 = np.array([[1,0],[1,2],[2,3]])
y1 = np.array([1,5,8])

x2 = np.array([[1, 2], [1, 3], [1, 4]])
y2 = np.array([3, 5, 7])
test_model2 = LinearRegression(2)

# Tests train
assert test_model1.train(x1, y1) == pytest.approx(np.array([1, 2]))
assert test_model2.train(x2, y2) ==  pytest.approx(np.array([-1, 2]))

# Test predict
assert test_model1.predict(np.array([[0,1], [-1,.5], [1,2]])) == pytest.approx(
    np.array([2,0,5]))
assert test_model2.predict(np.array([[0,1], [-1,.5], [1,2]])) == pytest.approx(
    np.array([2,2,3]))

# Test average_loss
assert test_model1.average_loss(np.array([[0,1], [-1,.5], [1,2]]),
                                test_model1.predict(np.array(
                                    [[0,1], [-1,.5], [1,2]]))) == pytest.approx(
                                        0)
assert test_model2.average_loss(np.array([[0,1], [-1,.5], [1,2]]),
                                test_model2.predict(np.array(
                                    [[0,1], [-1,.5], [1,2]]))) == pytest.approx(
                                        0)
```

In [11]:
```
import pandas as pd
# Read sklearn.datasets.make_regression data - test case 3
X = pd.read_csv("./test_X.csv", header=None).to_numpy()
y = pd.read_csv("./test_y.csv", header=None).to_numpy()
test_model = LinearRegression(4)
# Test train
assert test_model.train(X, y) == pytest.approx(
    np.array([[89.88],[77.64],[97.00],[94.30]]), rel=0.01)
# Test predict
assert test_model.predict(X) == pytest.approx(
    np.array([[177.00],[-286.41],[-88.97],[-338.27],[2.95],\
    [28.82],[-118.96],[-239.44],[235.41],[-115.59]]), rel=0.01)
# Test Squared Error
assert squared_error(test_model.predict(X), y) == pytest.approx(3.58, rel=0.01)
```

Main

In [ ]:
```
from sklearn.model_selection import train_test_split
```

```
WINE_FILE_PATH = 'wine.txt'

def import_wine(filepath, test_size=0.2):
    '''
        Helper function to import the wine dataset
        @param:
            filepath: path to wine.txt
            test_size: the fraction of the dataset set aside for testing
        @return:
            X_train: training data inputs
            Y_train: training data values
            X_test: testing data inputs
            Y_test: testing data values
    '''

    # Load in the dataset
    data = np.loadtxt(filepath, skiprows=1)
    X, Y = data[:, 1:], data[:, 0]

    # Normalize the inputs
    X = (X-np.mean(X, axis=0))/np.std(X, axis=0)

    X_train, X_test, Y_train, Y_test = train_test_split(
        X, Y, test_size=test_size)
    return X_train, X_test, Y_train, Y_test


def test_linreg():
    '''
        Helper function that tests LinearRegression.
    '''

    X_train, X_test, Y_train, Y_test = import_wine(WINE_FILE_PATH)

    num_features = X_train.shape[1]

    # Padding the inputs with a bias
    X_train_b = np.append(X_train, np.ones((len(X_train), 1)), axis=1)
    X_test_b = np.append(X_test, np.ones((len(X_test), 1)), axis=1)

    #### Matrix Inversion ######
    print('---- LINEAR REGRESSION w/ Matrix Inversion ---')
    solver_model = LinearRegression(num_features)
    solver_model.train(X_train_b, Y_train)
    print('Average Training Loss:',
          solver_model.average_loss(X_train_b, Y_train))
    print('Average Testing Loss:',
          solver_model.average_loss(X_test_b, Y_test))


# Set random seeds. DO NOT CHANGE THIS IN YOUR FINAL SUBMISSION.
random.seed(0)
np.random.seed(0)
test_linreg()
```

```
---- LINEAR REGRESSION w/ Matrix Inversion ---
Average Training Loss: 0.5403729394828255
Average Testing Loss: 0.6598453517957841
```

Project Report (6 points)

Each programming assignment in this course will be accompanied by a short report in which you will answer a few guiding questions. These questions are included to promote critical thinking about the results of your algorithms, both mathematically and societally. By the end of this course you will not only be able to implement common machine-learning algorithms but also develop intuition as to how the results of a given algorithm should be interpreted and can impact the world around you.

Question 1

Linear regression analysis makes several assumptions. For one, all observations in the data must be independent of each other (e.g., the data should not include more than one observation on any individual/unit). Furthermore, the data should avoid including extreme values since these will skew the results and create a false sense of relationship in the data. In general, linear regression gives more weight to cases that are far from the average. Can you think of any examples or datasets in which this might pose an issue?

Solution:

Example 1: Multiple measurements of blood pressure or test results for the same set of patients over time.This dataset is non-iid because each patient's results come from different distributions.Using linear regression directly would incorrectly treat each observation as independent, leading to biased estimates.

Example 2: Product prices may spike or drop sharply during holidays or promotional events. These extreme values can disproportionately influence the linear regression model, causing the overall predictions to deviate from the true trend.

Question 2

Suppose a machine learning researcher at a company learns a model to automate the hiring process. This company sells software based on this model to other companies looking to expedite their hiring. However, it is later discovered that the algorithm heavily favors members of a certain class unfairly.

1. In this situation, who should be to blame for the unfair hiring?
2. On whom does the responsibility fall to check the fairness of automated systems?

Solution:

1. Machine learning researcher: When the researcher collect and clean data, if the collected data is biased (e.g., only includes certain groups) or improperly processed, the algorithm will learn these biases.

   Company which sells software: Must test the algorithm for fairness before distributing it; failure to do so incurs responsibility.

2. Machine learning researcher: Must ensure the algorithm treats all groups fairly and does not introduce systematic bias.