Charlie Misbach

CSCI-3470

Assignment #1

Due: 2/17/25

**Question 1.1**

from matplotlib import pyplot as plt

import numpy as np

from sklearn.linear\_model import LinearRegression

import sklearn.model\_selection

import pandas as pd

*# Question: 1.1*

*# Read / prep dataset*

data = pd.read\_excel('dataset.xlsx')

df = pd.DataFrame(data)

print(df)

*# Split into test / train*

X = df[['YearsExperience']]

y = df['Salary']

X\_train, X\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=42)

*# Train Linear Regression Model*

model = LinearRegression()

model.fit(X\_train, y\_train)

w = model.coef\_[0]

b = model.intercept\_

print("\n--- Model Parameters ---")

print("Coefficient (w):", w)

print("Intercept (b):", b)

*# Now predict on the test set*

y\_pred = model.predict(X\_test)

comparison\_df = pd.DataFrame({

    'YearsExperience': X\_test['YearsExperience'],

    'Actual Salary': y\_test,

    'Predicted Salary': y\_pred

})

print("\n--- Comparison of Actual vs Predicted ---")

print(comparison\_df)

*# Plotting the results*

plt.scatter(X\_train, y\_train, *color*='blue', *label*='Training Data')

plt.scatter(X\_test, y\_test, *color*='red', *label*='Testing Data')

X\_range = np.linspace(df['YearsExperience'].min(), df['YearsExperience'].max(), 100)

Y\_range\_pred = model.predict(X\_range.reshape(-1, 1))

plt.plot(X\_range, Y\_range\_pred, *color*='green', *label*='Regression Line')

plt.title('Linear Regression: Years of Experience vs. Salary')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.legend()

plt.show()

**2. Plots**

****

In the graph is a linear regression model predicting salary based upon years of experience   
Red dots – testing data (for checking model accuracy)  
Blue dots – training data (for training the model)  
Green line – best fit line according to the regression model

**3. Discussion**

I learned about how to properly use train\_test\_split on a dataset and using that library I managed to separate training from test data and train a model on this data correlation using linear regression which I then displayed later on a graph with the actual vs predicted numbers.

**Question 1.2:**

*# Question: 1.2*

*# Naive Bayes (from scrath)*

*# Training data*

C1\_data = [

    [1, 1, 1],

    [0, 1, 0],

    [1, 1, 0]

]

C2\_data = [

    [0, 0, 0],

    [1, 0, 1],

    [1, 0, 0]

]

*# New sample to classify*

test\_x = [1, 0, 0]

*# 1) Calculate Priors: P(C1) and P(C2)*

num\_C1 = len(C1\_data)

num\_C2 = len(C2\_data)

total\_samples = num\_C1 + num\_C2

P\_C1 = num\_C1 / total\_samples

P\_C2 = num\_C2 / total\_samples

print("P(C1) =", P\_C1)

print("P(C2) =", P\_C2)

*# 2) Estimate Likelihoods P(x|C) = product of P(x\_i|C)*

*#    For each feature i, we need P(feature i = 1 | C) and P(feature i = 0 | C)*

def feature\_probabilities(*class\_data*):

    """

    parameter - class\_data, list of samples beloning to

    particular class, each sample is a list of features (0, 1)

    returns - probabilites, list of probabilites where each element

    represents P(feature i = 1 | class), how often each feature

    appears as 1 in the class

    """

*# Number of samples for this class*

    n\_samples = len(*class\_data*)

*# Number of features (assuming all samples have same length)*

    n\_features = len(*class\_data*[0])

*# Initialize counters for how many times each feature is "1"*

    counts = [0] \* n\_features

*# Count how often each feature is 1*

    for sample in *class\_data*:

        for i in range(n\_features):

            counts[i] += sample[i]

*# Probability of feature i = 1 => counts[i] / n\_samples*

*# We'll store them in a list*

    probabilities = [counts[i] / n\_samples for i in range(n\_features)]

    return probabilities

*# Get probabilities for each feature P(feature i=1|C1), P(feature i=1|C2)*

C1\_feature\_probs = feature\_probabilities(C1\_data)

C2\_feature\_probs = feature\_probabilities(C2\_data)

def naive\_likelihood(*x*, *feature\_probability*):

    """"

    parameters:

    - x (list): The new test sample, a list of binary values (0, 1).

    - feature\_probability (list): A list of probabilities where each element represents

      P(feature i = 1 | class).

    returns - likelihood (float): The product of probabilities computed using

    the Naïve Bayes assumption:

    P(x | C) = P(x\_1 | C) \* P(x\_2 | C) \* ... \* P(x\_n | C)

    """

    likelihood = 1.0

    for i, val in enumerate(*x*):

        p1 = *feature\_probability*[i]

        if val == 1:

            likelihood \*= p1

        else:

            likelihood \*= (1 - p1)

    return likelihood

P\_x\_given\_C1 = naive\_likelihood(test\_x, C1\_feature\_probs)

P\_x\_given\_C2 = naive\_likelihood(test\_x, C2\_feature\_probs)

print("\nP(x|C1) =", P\_x\_given\_C1)

print("P(x|C2) =", P\_x\_given\_C2)

*# 3) Compute posteriors (we can compare P\_x\_given\_C \* P\_C)*

posterior\_C1 = P\_x\_given\_C1 \* P\_C1

posterior\_C2 = P\_x\_given\_C2 \* P\_C2

P\_C1\_given\_x = posterior\_C1 / (posterior\_C1 + posterior\_C2)

P\_C2\_given\_x = posterior\_C2 / (posterior\_C1 + posterior\_C2)

print("\nP(C1|x) =", P\_C1\_given\_x)

print("P(C2|x) =", P\_C2\_given\_x)

*# 4) Classification decision*

if P\_C1\_given\_x > P\_C2\_given\_x:

    print("\nWe classify x as Class 1 (C1).")

else:

    print("\nWe classify x as Class 2 (C2).")

**2. Output**

**A screenshot of a computer

AI-generated content may be incorrect.**

Output for sample [1, 0, 0] classified as class 2 which   
makes sense because class 2 has many features = 1   
in col 1 but not 2 or 3

A screenshot of a computer

AI-generated content may be incorrect.

Output for sample [0, 1, 0] which also makes perfect  
sense as all of the features in C1 of the second column are  
1 along with the sample being classified

**3. Discussion**

This question helped a lot for me to properly understand the process that Naïve Bayes algorithm goes through. Particularly in terms of probability estimation, likelihood calculations and classification decisions. Since we implemented it from scratch in this case I was able to break down every individual part of the algorithm and gain insights into how it learns from proper data to make accurate predictions.