DSC 140A - Super Homework

Due: Wednesday, March 20

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope at 11:59 PM.

Note: you *may* use a slip day on this assignment, if you have any remaining.

Programming Problem 1. (Classifier Competition)

In this problem, you'll use what you've learned this quarter to build a simple classifier that can tell the difference between Spanish and French. To make things interesting, we'll run a competition to see who can build the best classifier, extra credit being given for particularly accurate classifiers.

The problem. Consider the words "meilleur" and "mejor". In English, both of these words mean "best" – one in French and the other in Spanish. Even if you don't know how to speak either of these languages, you might be able to guess which word is Spanish and which is French based on the spelling. For example, the word "meilleur" *looks* more like a French word than a Spanish word due to it containing "ei" and ending in "eur". On the other hand, the word "mejor" *looks* more like a Spanish word than a French word due to it containing "j" and ending in "or". This suggests that there is some statistical structure in the words of these languages that we can use to build a machine learning model capable of distinguishing between French and Spanish without actually understanding the words themselves.

Your goal in this problem is to build a machine learning model that can take a word as input and predict whether it is Spanish or French.

What you'll turn in. Unlike other problems from this quarter, this problem will require you to turn in a code file. You will turn into Gradescope a Python file named `classify.py` containing a function `classify()` that takes three arguments:

- `train_words`: a list of n strings, each one of them a word (in either Spanish or French)
- `train_labels`: a list of n strings, each one of them either `"spanish"` or `"french"`, indicating the language of the corresponding word in `train_words`
- `test_words`: a list of m strings, each one of them a word (in either Spanish or French)

This function should return a list of m strings, each one of them either `"spanish"` or `"french"`, indicating your classifier's prediction for the language of the corresponding word in `test_words`.

Your `classify()` function is responsible for training your machine learning model on the training data and then using that model to make predictions on the test data.

You may include other helper functions in your `classify.py` file if you wish, but the only function that will be called by the autograder is `classify()`. You can import `numpy`, `pandas`, `scipy`, or any module in the Python standard library, but you may *not* use any other third-party libraries (like `sklearn`). That is, you will need to implement your classifier "from scratch".

The data. The data set linked below contains 1200 words, about half of which are Spanish and half of which are French:

`https://f000.backblazeb2.com/file/jeldridge-data/012-spanish_french/train.csv`

We will use this data set to train your model. It was constructed by translating common texts from English to Spanish and French and then replacing any accented characters with their unaccented equivalents and removing special characters.

Your model will be tested on a separate, unseen data set that was constructed in a similar way. The test data will be (approximately) balanced between Spanish and French words.

Rules.

- The code that you turn in can use `numpy`, `pandas`, `scipy`, or any module in the Python standard library, but you may *not* use any other third-party libraries (like `sklearn`). You will need to implement your classifier "from scratch".
- The machine learning model that you implement must be one that we covered this quarter.
- Your code should not use any data other than the training data passed to `classify()` to build the model. For example, you can't download a Spanish dictionary from the internet and use it to make predictions!

Competition and Grading You will achieve full credit on this problem if your classifier satisfies the above rules and achieves an accuracy of at least 75% on the unseen test data set. We've tried a few simple models and found that it's possible to achieve this level of accuracy with a relatively basic technique, so you will not need to do anything too fancy to achieve full credit.

If your classifier achieves an accuracy of less than 75%, you will receive partial credit.

If your classifier does *better* than 80% accuracy on the test set, you'll get extra credit. More precisely, we'll award 1 point of extra credit for every percentage point by which your classifier's accuracy exceeds 80%. For example, if your classifier achieves 85% test accuracy, you'll receive 5 points of extra credit. These extra credit points will be applied to your overall homework score for the quarter (not to the Super Homework itself) and can therefore earn credit back for points you might have missed on Homeworks 1-8. Your total homework credit for the quarter *will* be allowed to exceed 100%.

The winner(s) of the competition will be the student(s) who achieve the highest test accuracy. The winner(s) will not receive any additional extra credit on top of the extra credit that they receive for their test accuracy (they probably won't need it!). However, they will receive a paragraph in any future letter of recommendation that I write for them dscribing their achievement.

As in the "real world", the autograder will not tell you your test accuracy – you will need to estimate it yourself using the data that we provided. However, to give you some peace of mind, it will warn you if your test accuracy is below 75% (though it won't tell you by how much).

Tips and Hints.

- A good choice of features is important. You might want to consider using the frequency of different letters or pairs of letters in each word as features. For example, one of your features might be whether "el" appears in the word. You do not need to create all of these features by hand – you can use Python to help you generate them.
- Don't confuse training accuracy with test accuracy. It is possible to achieve 90%+ training accuracy on this data set, but that doesn't mean your model will generalize well to the test set.
- Be careful to avoid overfitting! If you use too many features or too complex of a model, you may find that your model performs well on the training data but poorly on the test data.
- Start with the simplest models first. We have learned some models in this class that can be implemented using only a couple lines of code (with `numpy`).

Problem 1.

For this problem, you will need the data set below. It contains 50 points in $\mathbb{R}^2$ suitable for linear regression:

`https://f000.backblazeb2.com/file/jeldridge-data/013-regression/data.csv`

a) Suppose we wish to fit a linear model $H(x) = w_0 + w_1 x$ to this data, and it is known that $w_0 = 2$. Write a function, `mse(w_1)`, that takes in a possible value of w_1 and returns the mean squared error of the linear model $H(x) = 2 + w_1 x$ with respect to the data set. You may use `numpy`. This problem is not autograded, but turn in your code (either copy/pasting or as a screenshot).

Hint: The MSE at $w_1 = 1$ should be approximately 139.6.

> **Solution:** Looking at the code below, we can see that the function `mse(w_1)` returns the mean squared error of the linear model $H(x) = 2 + w_1 x$ with respect to the data set as:
>
> 139.60271462

b) Suppose we are now thinking probabilistically, and we believe that the data has been generated by the model $Y = w_0 + w_1 X + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Suppose it is known that $\sigma = 4$ and that $w_0 = 2$.

Write a function, `log_likelihood(w_1)`, that takes in a possible value of w_1 and returns the log likelihood of the linear model $Y = 2 + w_1 X + \epsilon$ with respect to the data set. You may use `numpy`. Show your code.

Hint: The log likelihood at $w_1 = 1$ should be approximately -333.39.

> **Solution:** Looking at the code below, we can see that the function `log_likelihood(w_1)` returns the log likelihood of the linear model $Y = 2 + w_1 X + \epsilon$ with respect to the data set as:
>
> -333.3908863099782

c) Using `scipy.optimize.fmin`, find the value of w_1 that minimizes the mean squared error.

Note: doing this numerically is overkill: we have a closed-form solution for the value of w_1 that minimizes the mean squared error.

> **Solution:** In the code below using `scipy.optimize.fmin`, we find that the value of w_1 that minimizes the mean squared error is:
>
> 3.0071875

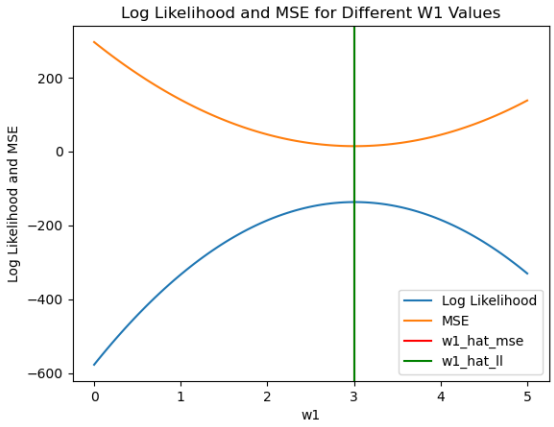
d) Again using `scipy.optimize.fmin`, find the value of w_1 that maximizes the log likelihood.

> **Solution:** In the code below using `scipy.optimize.fmin`, we find that the value of w_1 that maximizes the log likelihood is:
>
> 3.0071875

e) Plot the MSE for all values of w_1 between 0 and 5. On the same graph, plot the log likelihood for all of the same values. Plot a vertical line at the value of w_1 that minimizes the mean squared error (and maximize the log likelihood). Include a legend in your plot that tells which line is which.

Note: you may use `matplotlib` to make your plot. You might want to use `plt.axvline` to plot the vertical line and `plt.legend` to include a legend.

Solution: Using the code below we have the plot below. Also note that for both the MSE and log likelihood, the value of w_1 that minimizes the mean squared error (and maximizes the log likelihood) is 3.0071875 which is the same value for both.



Code

```
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt

# Load the data
data = np.loadtxt('data.csv', delimiter=',')
X = data[:, 0]
y = data[:, -1]

def mse(w1):
    y_pred = 2 + (w1 * X)
    return np.mean((y - y_pred) ** 2)

## print(mse(1))
print(mse(1))

def log_likelihood(w_1):
    # Calculate the residuals: (y_i - (2 + w_1*x_i))
    residuals = y - (2 + w_1 * X)
    sigma = 4
    sigma_squared = sigma**2

    # Calculate the log likelihood
    ll = -len(y)/2 * np.log(2 * np.pi * sigma_squared) - np.sum(residuals**2) / (2 * sigma_squared

    return ll

# print log_likelihood(1)
print(log_likelihood(1))

w1_init = 0
w1_hat_mse = opt.fmin(mse, w1_init)
print(w1_hat_mse)
```

```
def log_likelihood_inv(w_1):
    # Calculate the residuals: (y_i - (2 + w_1*x_i))
    residuals = y - (2 + w_1 * X)
    sigma = 4
    sigma_squared = sigma**2

    # Calculate the log likelihood
    ll = -len(y)/2 * np.log(2 * np.pi * sigma_squared) - np.sum(residuals**2) / (2 * sigma_squared

    return -ll

w1_init = 0
w1_hat_ll = opt.fmin(log_likelihood_inv, w1_init)
print(w1_hat_ll)


linspace = np.linspace(0, 5, 100)

plt.plot(linspace, [log_likelihood(w1) for w1 in linspace])
plt.plot(linspace, [mse(w1) for w1 in linspace])
plt.axvline(w1_hat_mse, color='r')
plt.axvline(w1_hat_ll, color='g')
plt.legend(['Log Likelihood', 'MSE', 'w1_hat_mse', 'w1_hat_ll'])
plt.title('Log Likelihood and MSE for Different W1 Values')
plt.xlabel('w1')
plt.ylabel('Log Likelihood and MSE')
plt.show()
```

Problem 2.

Suppose you are training a decision tree classifier to predict whether it will rain in the next hour or not. To do so, you will use two features: the current temperature (in Fahrenheit) and the current pressure (in millibars). Here is some training data that you have collected:

Temperature	Pressure	Rain?
65	1001	Yes
72	1003	Yes
79	1030	No
55	1022	Yes
62	1025	No
71	1010	Yes
73	1011	No

Train the decision tree to a depth of two. In other words, decide on a root question, splitting the data into two groups, then decide on another question for each group (if necessary). Your resulting decision tree should have two interior (question) nodes and three leaf nodes and should classify all of the training data correctly. Use the Gini coefficient to measure uncertainty. Every time you must decide on a question, choose the one that minimizes the uncertainty of the resulting split. Use as your questions thresholds of the form: "Is temp. < 55?", "Is temp. < 65?", ..., "Is pressure < 1030?".

You may solve this problem by hand, or you may write code to help you solve it. If you use code, you *may*

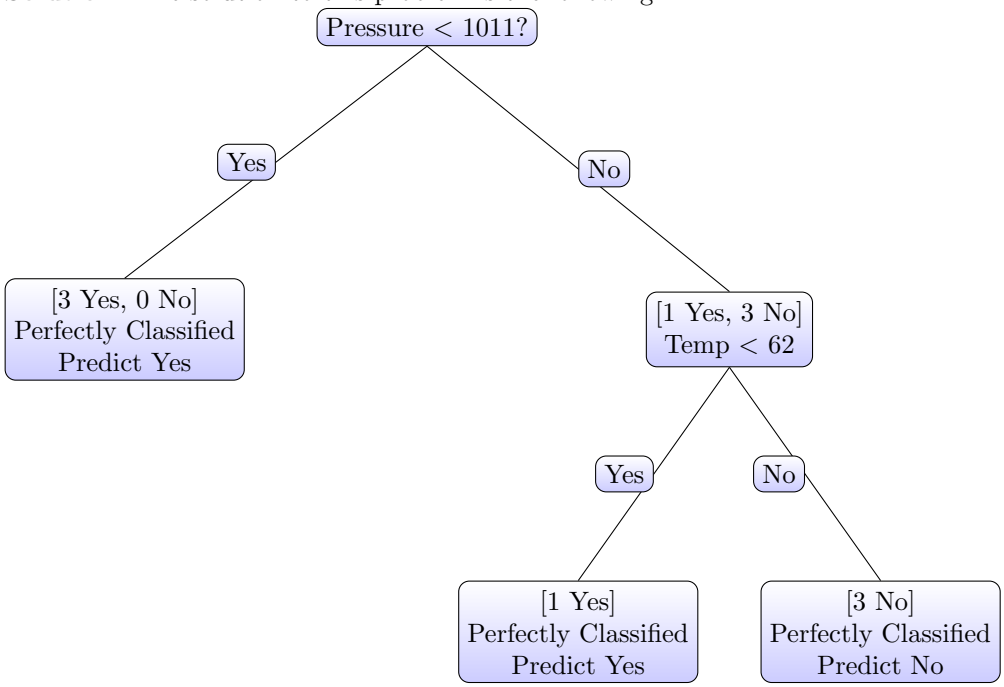
not use any third party libraries besides `numpy` and `pandas`. If you use code, turn in your code with your solution.

Your answer should include:

- A drawing of the resulting decision tree that labels each interior node with the question that it asks and each leaf node with the class that it predicts.
- For each interior node, a calculation of the uncertainty of all possible splits, with the best split highlighted in some way.

Note: you should choose the *best split* at every step. Namely, your root node should ask the question that minimizes uncertainty. It is possible to choose a different root question and still get a decision tree of depth 2 that perfectly classifies the training data, but this tree will not be considered correct because there is a unique root question minimizing uncertainty.

Solution: The solution to this problem is the following:



The code for getting the Gini uncertainty is given below. However, looking at the Tree, we can see that the best split is at the root node, where the pressure is less than 1011. This is because the Gini uncertainty is minimized at this split and we can see given the training data, the 'Yes' split is perfectly classified, therefore we predict 'Yes'.

Looking at the right split, for the points where the pressure is > 1011, we can see that the best split is at the temperature < 62. This is because the Gini uncertainty is minimized at this split and we can see given the training data, the 'Yes' split is perfectly classified, therefore we predict 'Yes' and the 'No' split is also perfectly classified, therefore we predict 'No'.

Here are the uncertainties for each split calculated from the code,

Root node

Split	Gini Uncertainty
65	0.4857142857142857
72	0.40476190476190477
79	0.38095238095238093
55	0.4897959183673469
62	0.42857142857142855
71	0.47619047619047616
73	0.22857142857142854
1001	0.4897959183673469
1003	0.42857142857142855
1030	0.38095238095238093
1022	0.40476190476190477
1025	0.22857142857142854
1010	0.34285714285714286
1011	**0.21428571428571427**

Right Split

Split	Gini Uncertainty
79	0.33333333333333337
55	0.375
62	0.0
73	0.25
1030	0.33333333333333337
1022	0.33333333333333337
1025	0.25
1011	0.375

Code

```
import numpy as np
import pandas as pd

# data
df = pd.DataFrame({
    'Temperature': [65, 72, 79, 55, 62, 71, 73],
    'Pressure': [1001, 1003, 1030, 1022, 1025, 1010, 1011],
    'Rain?': ['Yes', 'Yes', "No", 'Yes', 'No', 'Yes', 'No']
})\

def cal_gini(df):
    """
        Given split data, calculate the gini index
    """
    yes_count = (df['Rain?'] == 'Yes').sum()
    no_count = (df['Rain?'] == 'No').sum()
    if yes_count == 0 and no_count == 0:
        return 0
    yes_p = yes_count/len(df)
    return 2*yes_p*(1-yes_p)

def split_data(df, feature, value):
```

```
    """
        Given a feature and value, split the data into two groups
    """
    left = df[df[feature] < value]
    right = df[df[feature] >= value]
    return left, right

#########################################################
# Root node split
#########################################################
# Get the gini index for the original data for all possible splits
temp_values = df['Temperature'].unique()
pressure_values = df['Pressure'].unique()

gini_dict = {}

for temp in temp_values:
    left, right = split_data(df, 'Temperature', temp)
    gini_left = cal_gini(left)
    gini_right = cal_gini(right)
    gini_dict[temp] = (left.shape[0]/df.shape[0])*gini_left + (right.shape[0]/df.shape[0])*gini_right

for pressure in pressure_values:
    left, right = split_data(df, 'Pressure', pressure)
    gini_left = cal_gini(left)
    gini_right = cal_gini(right)
    gini_dict[pressure] = (left.shape[0]/df.shape[0])*gini_left + (right.shape[0]/df.shape[0])*gini_rig

# Find the best split for root node
best_split = min(gini_dict, key=gini_dict.get)
x = 'Temperature' if best_split in temp_values else 'Pressure'
print(f"Best split is {x} <", best_split, "with gini index", gini_dict[best_split])

#########################################################
# Left node split
#########################################################
left_data_layer_1 = df[df[x] < best_split]
right_data_layer_1 = df[df[x] >= best_split]

# Calculate the gini index for the left
left_temp_values = left_data_layer_1['Temperature'].unique()
left_pressure_values = left_data_layer_1['Pressure'].unique()

left_gini_dict_layer_1 = {}

for temp in left_temp_values:
    left, right = split_data(left_data_layer_1, 'Temperature', temp)
    gini_left = cal_gini(left)
    gini_right = cal_gini(right)
    left_gini_dict_layer_1[temp] = (left.shape[0]/left_data_layer_1.shape[0])*gini_left + (right.shape[

for pressure in left_pressure_values:
    left, right = split_data(left_data_layer_1, 'Pressure', pressure)
    gini_left = cal_gini(left)
```

```
    gini_right = cal_gini(right)
    left_gini_dict_layer_1[pressure] = (left.shape[0]/left_data_layer_1.shape[0])*gini_left + (right.sha

# Find the best split for left node
best_split_left_layer_1 = min(left_gini_dict_layer_1, key=left_gini_dict_layer_1.get)
x = 'Temperature' if best_split_left_layer_1 in left_temp_values else 'Pressure'
print(f"Best split for left layer 1 is {x} <", best_split_left_layer_1, "with gini index", left_gini_di

########################################################
# Right node split
########################################################
right_temp_values = right_data_layer_1['Temperature'].unique()
right_pressure_values = right_data_layer_1['Pressure'].unique()

right_gini_dict_layer_1 = {}

for temp in right_temp_values:
    left, right = split_data(right_data_layer_1, 'Temperature', temp)
    gini_left = cal_gini(left)
    gini_right = cal_gini(right)
    right_gini_dict_layer_1[temp] = (left.shape[0]/right_data_layer_1.shape[0])*gini_left + (right.shape

for pressure in right_pressure_values:
    left, right = split_data(right_data_layer_1, 'Pressure', pressure)
    gini_left = cal_gini(left)
    gini_right = cal_gini(right)
    right_gini_dict_layer_1[pressure] = (left.shape[0]/right_data_layer_1.shape[0])*gini_left + (right.s

best_split_right_layer_1 = min(right_gini_dict_layer_1, key=right_gini_dict_layer_1.get)
x = 'Temperature' if best_split_right_layer_1 in right_temp_values else 'Pressure'
print(f"Best split for right layer 1 is {x} <", best_split_right_layer_1, "with gini index", right_gini_
```