

Name: _____ PUID: _____

Instructions and Policy: You are allowed to study with others and use online resources for reference, however, the work you turn in must be your own. This means do not copy/paste from Stack Exchange (or from another student.) If you have worked closely with other students, provide their name(s) and a brief (at most one paragraph) description of the interaction; if we feel this oversteps the bounds, we will discuss it with you. Each student should write up their own solutions independently.

The requirements below are supposed to be followed in this and further homework assignments.

- For your theoretical submission:
 - **YOU MUST INCLUDE YOUR NAME IN THE HOMEWORK.**
 - The answers **MUST** be submitted via Gradescope.
 - Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.
 - Theoretical questions **MUST include the intermediate steps to the final answer.**
- For your programming answers:
 - **The Python scripts will be submitted separately via Gradescope**
 - Zero points will be given in any question where the Python code answer doesn't match the answer on Gradescope.
 - If the answer is/includes a plot, it should be added to your theoretical submission unless otherwise specified.

Your code is **REQUIRED** to run on Python 3.11 in an environment detailed in Homework 0 under the Python Installation section. While your code may run in other environments, any points lost due to environmental issues will not be regraded.

Please make sure you don't use any libraries that are not imported for you. If such a library is used, you will get 0 pt for the coding part of the assignment.

If your code doesn't run on Gradescope, then even if it compiles on another computer, it will still be considered not running and the respective part of the assignment will receive 0 pt.

Theoretical Questions (16+14+16+8=54 pts)

Please submit your answers on Gradescope.

Q1 (16 pts): True or False Questions

Answer the following as True or False with a justification or example. Points are uniformly distributed within the questions.

1. If event E and event F are independent, then the complementary events, E^c and F^c are not independent.

2. Consider three events A, B, C , the identity always holds $P(A|B \wedge C)P(B|C) = P(A \wedge B|C)$.

3. Increasing k in the k-nearest-neighbors algorithm leads to a more complicated decision boundary.

4. If ran indefinitely, the decision tree will get eventually 0 training error (assuming all examples are distinct).

Q2 (14 pts): Probability Review

1. (4 pts) Independence

Let T_1, T_2, T_3 denote three independent Bernoulli random variables with probability of success $p = \frac{1}{2}$, $T_i \sim \text{Bernoulli}(\frac{1}{2})$ for $i = 1, 2, 3$. Construct the following random variables $X = T_1 \oplus T_2, Y = T_2 \oplus T_3, Z = T_3 \oplus T_1$, where \oplus indicates the XOR (exclusive OR) operator. Show that random variables X, Y, Z are pairwise independent but not mutually independent.

This truth table shows the XOR (exclusive OR) operation between two Bernoulli random variables T_1, T_2 , where the output is 1 if the inputs differ, and 0 if they are the same.

T_1	T_2	$T_1 \oplus T_2$
0	0	0
0	1	1
1	0	1
1	1	0

2. (6 pts) Bayes Theorem

A bin contains 3 types of disposable flashlights. The probability that a type 1 flashlight will give more than 100 hours of use is 0.6, with the corresponding probabilities for type 2 and type 3 flashlights being 0.4 and 0.2, respectively. Suppose that 20 percent of the flashlights in the bin are type 1, 30 percent are type 2, and 50 percent are type 3.

1. What is the probability that a randomly chosen flashlight will give more than 100 hours of use?
2. Given that a flashlight lasted more than 100 hours, what is the conditional probability that it was a type j flashlight, $j = 1, 2, 3$?

3. (4 pts) Probability Distribution

Consider two continuous random variables X and Y with joint probability density function

$$\begin{cases} f_{XY}(x, y) = \frac{1}{4}xy & 0 < x < k, \quad 0 < y < k, \\ f_{XY}(x, y) = 0 & \text{otherwise,} \end{cases}$$

Recall that a probability *density* function has the following properties:

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_{XY}(x, y) dx dy = 1$$

- (a) (1 pt) Compute the value of k so that $f_{XY}(x, y)$ is a valid joint probability distribution function.
- (b) (1 pt) Compute $P(X > Y)$ as a function of k .
- (c) (1 pt) Compute $P(X + Y > 1)$ as a function of k .
- (d) (1 pt) Using the k value obtained from (a), prove that random variables X and Y are independent.

Q3 (16 pts): KNN

1. (4 pt) Consider you have the following 2D dataset (with binary class labels) as shown in Figure 1:

- Class +1 : $\{(1, 1), (1, 5), (5, 1)\}$
- Class -1: $\{(4, 6), (5, 5)\}$

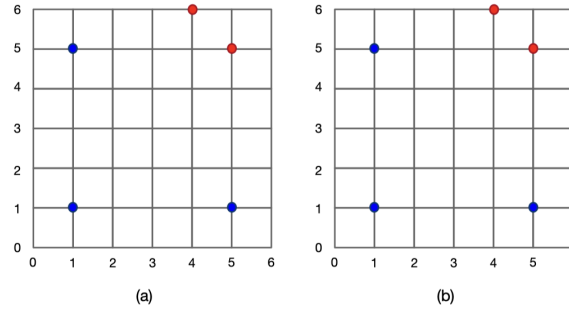


Figure 1: Data points: Blue = +1 and Red = -1

Suppose the data is strictly confined within the $[0, 6] \times [0, 6]$ grid. Draw the decision boundaries for a 1-NN classifier with Euclidean distance in Figure 1(a) and L1 distance in Figure 1(b). What class will the Euclidean distance classifier predict the test point $(2, 6)$ as?

2. (4 pt) Your Finnish friend Aleksandra works with the same data set. However, she measured the second coordinate in centimeters instead of meters (the first dimension is unchanged). The data thus becomes:

- Class +1 : $\{(1, 100), (1, 500), (5, 100)\}$
- Class -1: $\{(4, 600), (5, 500)\}$

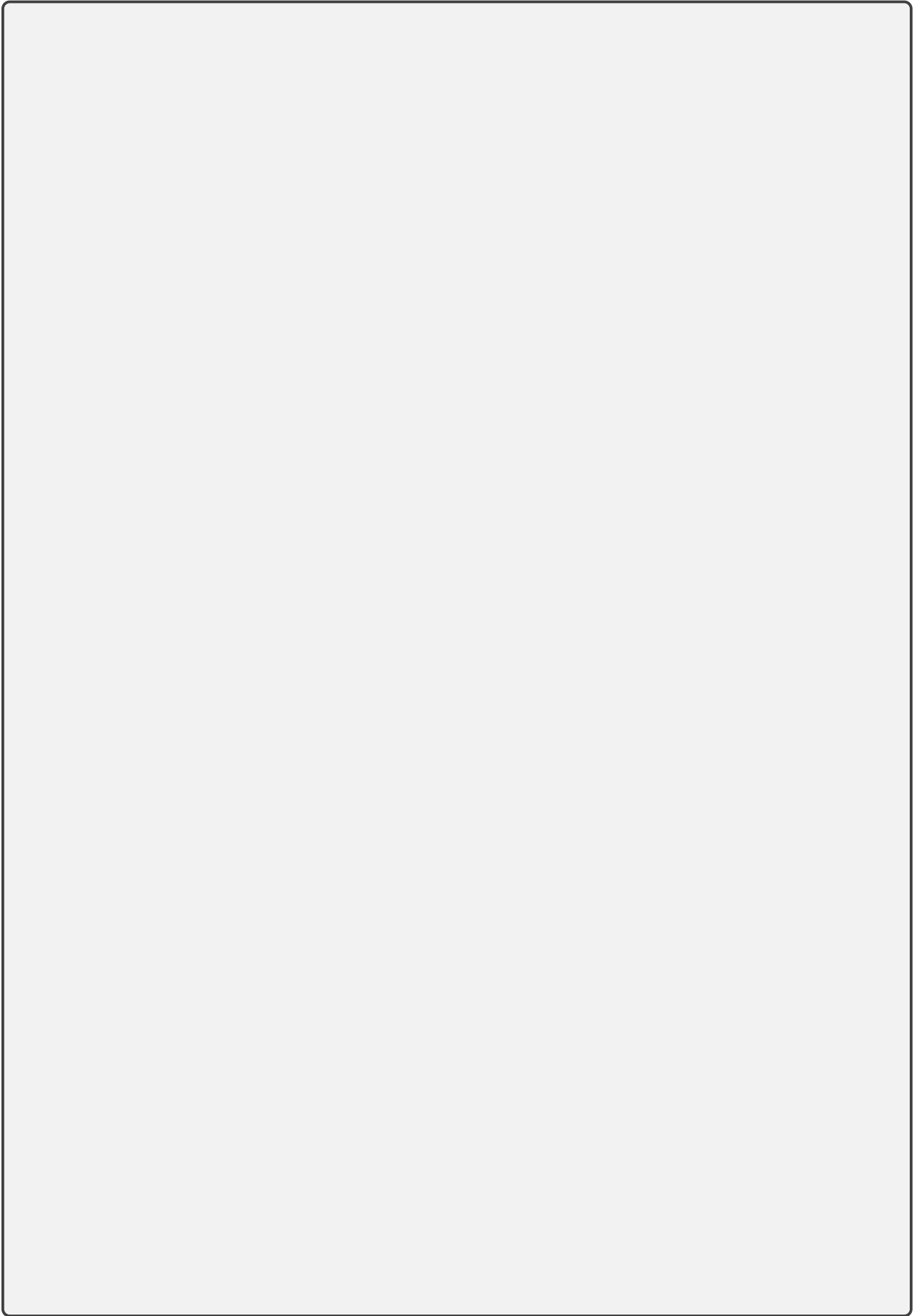
Does her 1-NN classifier with Euclidean distance classify points differently? What will she predict for the original test point $(2, 6)$, which she represents as $(2, 600)$?

3. (4 pt) k -NN can also be used for regression (i.e. your labels are real values now). Here, instead of predicting the most common label amongst the neighbors, we predict the label average. Assume we are using a non-weighted average. Suppose you have the following dataset:

\mathcal{X}	\mathcal{Y}
(0, 0)	1.0
(1, 1)	2.5
(2, 3)	3.0
(3, 1)	1.0
(2, 1)	2.5

where \mathcal{X} is the feature vector and \mathcal{Y} is the label. What would be the label for $(0, 1)$ if we use 2-NN with Euclidean distance?

4. (4 pt) Does it take more time to train a k -NN classifier or to apply a k -NN classifier? Explain your reasoning. Please assume that the data is on the magnitude of millions of points.



Q4 (8 pts): Decision Tree

We are trying to build a classifier to figure out if a patient is likely to have heart disease. We gathered data about 15 different patients, including their glucose level (high, normal, low), gender (male, female), blood pressure (high, normal, low) and cholesterol (high, normal). The data is reported in the table below:

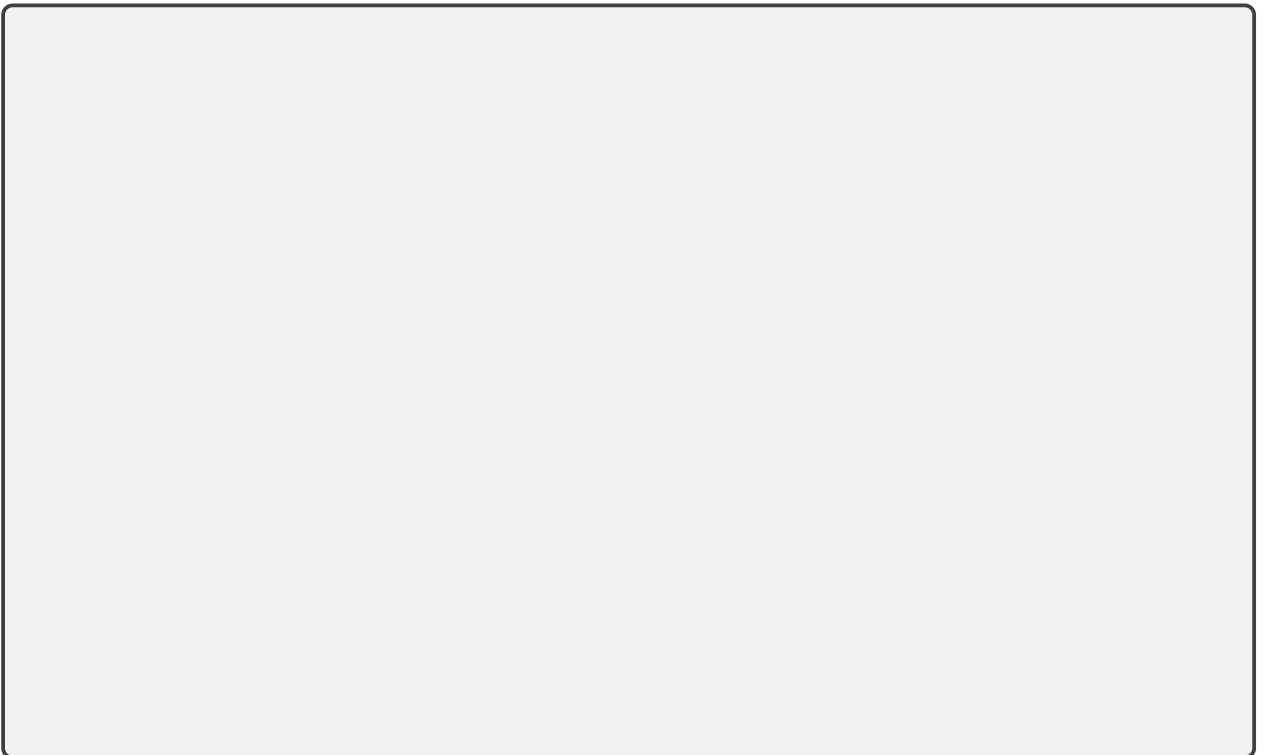
Table 1: Patient Dataset

ID	Glucose Level	Gender	Blood Pressure	Cholesterol	Heart Disease
1	Normal	Male	High	High	Yes
2	Normal	Female	Normal	Normal	No
3	High	Male	High	High	Yes
4	High	Female	Low	Normal	No
5	Low	Male	High	Normal	No
6	Low	Female	Low	High	Yes
7	High	Female	Low	High	No
8	Low	Male	High	Normal	Yes
9	Normal	Female	Normal	Normal	No
10	Normal	Male	Low	High	Yes
11	Normal	Female	High	Normal	Yes
12	Low	Male	Normal	High	Yes
13	High	Male	Normal	Normal	No
14	High	Female	High	Normal	No
15	Low	Male	Low	High	No

- (a) **(6 pts)** Using this data build a decision tree (use **information gain**) to decide whether a patient would have heart disease or not, showing at each level how you decided which attribute to expand next. Stop when the training set error (i.e. the fraction of points in the training set that it misclassified) drops below 0.30. **USE LOGARITHM WITH BASE 2. In case of a tie, choose the attributes in alphabetical order.**



- (b) (**2 pts**) State the IDs of the patients who are most likely to have heart disease by referring to the obtained decision tree.



Programming Part (12 + 23 + 11 = 46 pts)

General Instructions

For this assignment, please download the provided `hw1.zip` file on **Brightspace**.

Overview

In this assignment, you will learn and implement a KNN classifier and also a decision tree classifier.

Files

You will fill in functions in `knn.py`, `scorer.py`, and `dt.py`. After you finish, you will submit these as well as any other files mentioned in the assignment to Gradescope.

Asides from these three files, you will also find `utils.py` in the same folder. You do not need to modify these two files. `utils.py` contains some helper functions.

Packages

You will need the following packages for this assignment:

- `numpy`
- `pandas`
- `scipy` (only allowed for implementing chi-square gain with the `scipy.stats.chi2_contingency` method)

If you want, you can install `tqdm` to see a progress bar when running the autograder. This is not required for this class and might not be heavily focused on in future assignments, but it is a useful package to know if you are doing any machine learning work in Python.

These packages should be installed if you have installed Anaconda. If you are not using Anaconda, you can install them using `pip` or `conda`. For example, you can run `pip install numpy` in the terminal to install `numpy`.

Note that for this assignment, you should **not** use any other packages. You may see some other packages in the skeleton code, but they are only used for testing and grading. If you use any other packages, you may lose points. If you are not sure whether you can use a package, please ask the course staff.

Evaluation

Unless otherwise specified, your code will be evaluated by an autograder on Gradescope using the same environment as detailed below. You should make sure your code runs without errors in this environment. Otherwise, you may lose points. You can submit your code to Gradescope as many times as you want. We will only grade the latest submission.

There will be two types of test cases in the autograder: public (local) and hidden. Public test cases are visible to you, and you can see the results after you submit your code. You can also run the public test cases locally by running `python <filename>.py` in the same folder as your code. Passing all public test cases **does not** guarantee you will get full credit for the assignment.

Hidden test cases, however, are not visible to you, and you will not be able to see the results until your grade is published. The final score you get for this assignment is the sum of the scores of all public and hidden test cases.

Getting Help

If you have any questions about this assignment, please contact the course staff for help, preferably during office hours. You can also post your questions on the course forum. However, please do not post any code publicly. When asking questions, you should describe your problem in words and post the relevant code snippet. Please avoid showing a screenshot of your code to TAs and ask “why my code is not working”.

1 KNN (1 + 1 + 2 + 2 + 6 = 12 pts)

In this part, you will implement a KNN classifier. From our lecture, you should know that KNN is a non-parametric classifier, which means it does not make any assumption about the underlying distribution of the data. Instead, it uses the training data directly to make predictions. In this assignment, you will implement the KNN classifier using the Minkowski distance with order 3 as the distance metric.

You will write your code in `knn.py`.

1.1 (1 pts) KNN - Constructor

Under class `KNearestNeighbor`, fill in the constructor (`__init__`) so that it takes in the number of neighbors `k` and stores it as an attribute `self.k`. You should also initialize variables `self.X_train` and `self.y_train` to be `None`. These two attributes will be set in the `fit` method.

1.2 (1 pts) KNN - Fit

Under class `KNearestNeighbor`, fill in the method `fit` so that it takes in the training data `X_train` and `y_train` and stores them as attributes `self.X_train` and `self.y_train`. Note that `X_train` is a 2D array of shape `(n_samples, n_features)`, and `y_train` is a 1D array of shape `(n_samples,)`.

The two `assert` statements are there to help you debug. The first one checks whether the number of training samples is the same as the number of training labels. The second one checks whether the number of training samples is greater than or equal to the number of neighbors `k`. If either of them fails, you should raise a `ValueError` with an appropriate error message.

1.3 (2 pts) KNN - Minkowski Distance

Under class `KNearestNeighbor`, fill in the method `calc_distance` so that it takes in a single test sample `x`, a integer `p` for the order of the distance, and returns a 1D array of shape `(n_samples,)` containing the Minkowski distance between `x` and each training sample in `self.X_train`.

About Minkowski distance: The Minkowski distance is a versatile metric used in statistics and machine learning to compute the distance between two points in a vector space. Defined by a parameter p , it generalizes several distances: Euclidean ($p = 2$) and Manhattan ($p = 1$). The distance between two points x and y is calculated as:

$$D(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}} \quad (1)$$

where n is the number of dimensions. **In this homework, we will have $p = 3$.**

1.4 (2 pts) KNN - Get Neighbors

Under class `KNearestNeighbor`, fill in the method `get_top_k` so that it takes in a 1D array `distances` and returns a 1D array of shape `(self.k,)` containing the indices of the top `self.k` smallest elements in `distances`. You may find the function `np.argsort` useful.

1.5 (6 pts) KNN - Predict

Under class `KNearestNeighbor`, fill in the method `predict` so that it takes in a 2D array `X_predict` of shape `(n_samples, n_features)` and returns a 1D array of shape `(n_samples,)` containing the predicted labels for each sample in `X_predict`. You should use the `get_top_k` method you implemented in Q1.4 to get the indices of the top `self.k` smallest elements in `distances`. Then, you can use these indices to get the corresponding labels from `self.y_train` and return them as the predicted labels. You may find the function `np.bincount` and the function `np.argmax` useful.

2 Scorer (1 + 4 + 4 + 1 + 3 + 2 + 2 + 6 = 23 pts)

In this part, you will implement a decision tree classifier. From our lecture, you should know that a decision tree classifier is a tree-structured classifier that makes decisions based on the values of features. In this assignment, you will first implement a scorer that measures the quality of a split in terms of both information gain, Gini impurity and chi-square test. Then, you will implement a decision tree classifier that uses the scorer to make decisions.

You will write your code in `dt.py`.

2.1 (1 pts) Scorer - Constructor

Under class `Scorer`, fill in the constructor (`__init__`) so that it takes in a string type and stores it as an attribute `self.type`. The string type can be either "information" or "gini". The function also takes in a 1D array `class_labels` containing the class labels of the training data and `alpha` which is the parameter used for Laplace smoothing. You should store the unique class labels in `self.class_labels` and the parameter `alpha` in `self.alpha`.

2.2 (4 pts) Scorer - Class Probabilities

Under class `Scorer`, fill in the method `compute_class_probabilities` so that it takes in a 1D array `labels` containing the class labels of the training data and returns a dictionary containing the class probabilities. The keys of the dictionary are the unique class labels and the values are the corresponding class

probabilities. You should use Laplace smoothing with parameter `self.alpha` to compute the class probabilities. Make sure your code handles the case where the input array `labels` is empty and returns a non-empty dictionary with all class probabilities.

2.3 (4 pts) Scorer - Data Subsets

Under class `Scorer`, fill in the method `subset_data` so that it takes in a 2D array `data` containing the training data, a 1D array `labels` containing the class labels of the training data, an integer `split_attribute` indicating the index of the feature to split on, and a value `split_value` indicating the value of the feature to split on. The method should return a tuple of two arrays (`data_subsets`, `labels_subsets`) where `data_subsets` is a 2D array and `labels_subsets` is a 1D array. Each row in `data_subsets` and `labels_subsets` should correspond to a subset of the training data and the corresponding class labels that originally have the feature value `split_value` at the feature index `split_attribute`. You may find the function `np.where` useful.

2.4 (1 pts) Scorer - Information Score

Under class `Scorer`, fill in the method `information_score` so that it takes in a 1D array `labels` containing the class labels of the training data and returns the information score (entropy) of the class labels. You should use the class probabilities computed in Q2.2 to compute the information score. We use `log2` as the base of the logarithm.

2.5 (3 pts) Scorer - Scorer - Gini Score

Under class `Scorer`, fill in the method `gini_score` so that it takes in a 1D array `labels` containing the class labels of the training data and returns the Gini score of the class labels. You should use the class probabilities computed in Q2.2 to compute the Gini score.

2.6 (2 pts) Scorer - Information Gain

Under class `Scorer`, fill in the method `information_gain` so that it takes in a 2D array `data`, a 1D array `labels`, and an integer `split_attribute` indicating the index of the feature to split on. The method should return the information gain of splitting on the feature at index `split_attribute`. You should use the information score (entropy) computed in Q2.4 to compute the information gain.

2.7 (2 pts) Scorer - Gini Gain

Under class `Scorer`, fill in the method `gini_gain` so that it takes in a 1D array `labels`, and an integer `split_attribute` indicating the index of the feature to split on. The method should return the Gini gain of splitting on the feature at index `split_attribute`. You should use the Gini score computed in Q2.5 to compute the Gini gain.

2.8 (6 pts) Scorer - Chi-square Gain

Under class `Scorer`, fill in the method `chi_square_gain` so that it takes in a 2D array `data`, a 1D array `labels`, and an integer `split_attribute` indicating the index of the feature to split on. This method should calculate the Chi-square gain of splitting the dataset based on the feature at index `split_attribute`.

The Chi-square test is used to determine whether there is a significant association between the observed frequencies in the categorical data. In the context of a decision tree, the Chi-square gain measures how well a feature splits the data into pure classes, indicating the effectiveness of a split.

Task Details: You are required to construct the contingency table that captures the frequency of each class label for each unique value of the attribute specified by `split_attribute`. This table is critical for computing the Chi-square statistic, which will be done using `scipy.stats.chi2_contingency`, provided in your starter code.

Steps to Create the Contingency Table:

1. Identify all unique values of the attribute given by `split_attribute` in the `data` array.
2. For each unique value, create a list that counts the number of times each class label appears in `labels` where the corresponding value in `data` matches the unique attribute value.

Example: Suppose your `data` array is:

```
[[ 'green', 'M'],  
 [ 'red', 'L'],  
 [ 'green', 'S'],  
 [ 'blue', 'M']]
```

And your `labels` array is:

```
['Apple', 'Apple', 'Banana', 'Apple']
```

If you are to construct a contingency table based on the first attribute (color), your task is:

1. Extract unique values: `['green', 'red', 'blue']`
2. Count each class label for 'green': - Apple: 1 - Banana: 1
3. Count each class label for 'red': - Apple: 1 - Banana: 0 (no Banana labeled item for 'red')
4. Count each class label for 'blue': - Apple: 1 - Banana: 0

The resulting contingency table will be:

```
[[1, 1], # Counts for 'green'  
 [1, 0], # Counts for 'red'  
 [1, 0]] # Counts for 'blue'
```

This table shows the frequency of each label (Apple, Banana) for each unique attribute value.

Note: Implement the loop that fills the contingency table based on the observed counts of labels for each unique value of the attribute. Your correct implementation is essential for the `scipy` function call to compute the Chi-square statistic accurately. The documentation of the `scipy` function is here https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2_contingency.html

Additional resources: Check this very efficient and clear youtube video if you would like an alternative explanation of chi-square statistics for decision trees: <https://www.youtube.com/watch?v=8cwewynQ6k>.

3 Decision Tree (2 + 2 + 7 = 11 pts)

In this part, you will implement a decision tree classifier using the scorer you implemented in Part 2. You will write your code in `dt.py`.

You will see four classes in `dt.py`:

- `class Node(ABC)`, which is an abstract class that represents a node in the decision tree. You may ignore this class.
- `class Leaf(Node)`, which is a subclass of `Node` that represents a leaf node in the decision tree.
- `class Split(Node)`, which is a subclass of `Node` that represents a split node in the decision tree.
- `class DecisionTree`, which is a class that represents a decision tree classifier.

3.1 (2 pts) Decision Tree - Predicting Class Probabilities

Under `class Leaf`, fill in the method `predict_class_probabilities` so that it takes in a 2D array `X` containing the data and returns a 2D array containing the predicted class probabilities for each data point in `X`.

Note that since this is a leaf node, the predicted class probabilities should be the same for all data points in `X`. You should use `self.class_probabilities` and `self.class_labels` to compute the predicted class probabilities, as they represent the class probabilities of the training data that have reached this leaf node.

3.2 (2 pts) Decision Tree - Predicting Leaf Node Class

Under `class Leaf`, fill in the method `predict` so that it takes in a 2D array `X` containing the data and returns a 1D array containing the predicted class for each data point in `X`. The probabilities of each class for each data point in `X` are already computed in `probabilities` in the skeleton code. You should use `self.class_labels` to compute the predicted class for each data point in `X` based on the probabilities.

3.3 (7 pts) Decision Tree - Building the Tree

Under `class DecisionTree`, fill in the method `build_tree` so that it takes in a 2D array `data` containing the training data, a 1D array `labels` containing the class labels of the training data, an integer `max_depth` indicating, and a set `exclude` containing the indices of the features to exclude. The method should return a `Node` object representing the node in the decision tree. You should use the scorer you implemented in Part 2 to compute the gain of each feature and split the feature with the highest gain. You should use the `Split` and `Leaf` classes to represent the split and leaf nodes in the decision tree. Do not modify the code that is not guarded by `YOUR CODE HERE` and `END OF YOUR CODE`.

Submission

You will submit the following files to Gradescope:

- `knn.py`

- `scorer.py`
- `dt.py`