CS145 Howework 2

Important Note: HW2 is due on **11:59 PM PT, Oct 30 (Friday, Week 4)**. Please submit through GradeScope.

Print Out Your Name and UID

Name: Yunong Ye, UID: 004757414

Before You Start

You need to first create HW2 conda environment by the given `cs145hw2.yml` file, which provides the name and necessary packages for this tasks. If you have `conda` properly installed, you may create, activate or deactivate by the following commands:

```
conda env create -f cs145hw2.yml
conda activate hw1
conda deactivate
```

OR

```
conda env create --name NAMEOFYOURCHOICE -f cs145hw2.yml
conda activate NAMEOFYOURCHOICE
conda deactivate
```

To view the list of your environments, use the following command:

```
conda env list
```

More useful information about managing environments can be found here (https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html).

You may also quickly review the usage of basic Python and Numpy package, if needed in coding for matrix operations.

In this notebook, you must not delete any code cells in this notebook. If you change any code outside the blocks (such as some important hyperparameters) that you are allowed to edit (between START/END YOUR CODE HERE), you need to highlight these changes. You may add some additional cells to help explain your results and observations.

In [66]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import sys
import random as rd
import matplotlib.pyplot as plt
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

If you can successfully run the code above, there will be no problem for environment setting.

1. Decision trees

This workbook will walk you through a decision tree.

1.1 Attribute selection measures

For classification models, misclassification rate is usually used as the final performance measurement. However, for classification trees, when selecting which attribute to split, measurements people often use includes information gain, gain ratio, and Gini index. Let's investigate these different measurements through the following problem.

Note: below shows how to calculate the misclassification rate of a classification tree with N total data points, K classes of the value we want to predict, and M leaf nodes.

In a node $m, m = 1, \ldots, M$, let's denote the number of data points using N_m, and the number of data points in class k as N_{mk}, so the class prediction under majority vote is $j = argmax_k N_{mk}$. The misclassification rate of this node m is $R_m = 1 - \frac{N_{mj}}{N_m}$. The total misclassification rate of the tree will be $R = \frac{\sum_{m=1}^{M} R_m * N_m}{N}$

Questions

Note: this question is a pure "question answer" problem. You don't need to do any coding.

Suppose our dataset includes a total of 800 people with 400 males and 400 females, and our goal is to do gender classification. Consider two different possible attributes we can split on in a decision tree model. Split on the first attribute results in a node11 with 300 male and 100 female, and a node12 with 100 male and 300 female. Split on the second attribute results in in a node21 with 400 male and 200 female, and a node22 with 200 female only.

1. Which split do you prefer when the measurement is misclassifcation rate and why?
2. What is the entropy in each of these four node?
3. What is the information gain of each of the two splits?

4. Which split do you prefer if the measurement is information gain. Do you see why it is an uncertainty or impurity measurement?
5. What is the gain ratio (normalized information gain) of each of the two splits? Which split do you prefer under this measurement. Do you get the same conclusion as information gain?

Your answer here:

Note: you can use several code cells to help you compute the results and answer the questions. Again you don't need to do any coding.

In [67]:
```
def entropy(n1,n2):
    total = n1 + n2
    p1 = n1 / total
    p2 = n2 / total
    res = -p1*np.log2(p1) - p2*np.log2(p2)
    return res

print(entropy(300,100))
print(entropy(400,200)*3/4)
print(entropy(400,400))
print(entropy(600,200))
```

```
0.8112781244591328
0.6887218755408672
1.0
0.8112781244591328
```

Please type your answer here!

answer 1:

The information gain of second attribute is larger than the first so I will prefer use second split. The fisrt split has a misclassification rate of 1/4.
The second split has a misclassification rate of $\frac{3}{4} * \frac{1}{3} = \frac{1}{4}$.
So I think these two are the same in misclassification rate.

answer 2:
The entropy of node11 is $-3/4 * log_2(3/4) - 1/4 * log_2(1/4) = 0.811278$
The entropy of node12 is $-1/4 * log_2(1/4) - 3/4 * log_2(3/4) = 0.811278$
The entropy of node21 is $-2/3 * log_2(2/3) - 1/3 * log_2(1/3) = 0.918296$
The entropy of node22 is $0 * log_2(0) - 1 * log_2(1) = 0$

answer 3:
The entropy is $-1/2 * log_2(1/2) - 1/2 * log_2(1/2) = 1$
The condtiional entropy of first attribute is
$1/2 * entropy(node11) + 1/2 * entropy(node12) = 0.811278$
The conditional entropy of second attribute is

$3/4 * enrtopy(node21) + 1/4 * entropy(node22) = 0.688722$
The information gain of first attribute is 0.188722
The information gain of second attribute is 0.311278

answer 4:
I will choose second attribute using information gain. Yes by calculating entropy it favors the maximum reduction of uncertainty or impurity when choosing attribute.

answer 5:
The splitinfo of first attribute is $-1/2 * log_2(1/2) - 1/2 * log_2(1/2) = 1$
The splitinfo of second attribute is $-3/4 * log_2(3/4) - 1/4 * log_2(1/4) = 0.811278$
The gain ratio of first attreibute is 0.188722.
The gain ratio of second attribute is 0.383688.
I still prefer the second split which is the same as using information gain as measurement.

1.2 Coding decision trees

In this section, we are going to use the decision tree model to predict the the animal `type` class of the `zoo` dataset. The dataset has been preprocessed and splited into `decision-tree-train.csv` and `decision-tree-test.csv` for you.

In [68]:
```
from hw2code.decision_tree import DecisionTree
mytree = DecisionTree()
mytree.load_data('./data/decision-tree-train.csv','./data/decision-tree-test.csv'
# As a sanity check, we print out the size of the training data (80, 17) and test
print('Training data shape: ', mytree.train_data.shape)
print('Testing data shape:', mytree.test_data.shape)
```

```
Training data shape:  (80, 17)
Testing data shape: (21, 17)
```

1.2.1 Infomation gain

Complete the `make_tree` and `compute_info_gain` function in `decision_tree.py` .

Train you model using `info_gain` measure to classify `type` and print the test accuracy.

In [69]:
```python
from hw2code.decision_tree import DecisionTree
mytree = DecisionTree()
mytree.load_data('./data/decision-tree-train.csv','./data/decision-tree-test.csv'
test_acc= 0
#=========================#
# STRART YOUR CODE HERE  #
#=========================#
mytree.train('type', 'info_gain')
test_acc = mytree.test('type')

#=========================#
#   END YOUR CODE HERE   #
#=========================#
print('Test accuracy is: ', test_acc)
```

```
best_feature is:  legs
best_feature is:  fins
best_feature is:  toothed
best_feature is:  eggs
best_feature is:  hair
best_feature is:  hair
best_feature is:  toothed
best_feature is:  aquatic
Test accuracy is:  0.8571428571428571
```

1.2.2 Gain ratio

Complete the `compute_gain_ratio` function in `decision_tree.py` .

Train you model using `gain_ratio` measure to classify `type` and print the test accuracy.

In [70]:
```python
mytree = DecisionTree()
mytree.load_data('./data/decision-tree-train.csv','./data/decision-tree-test.csv'
test_acc = 0
#=========================#
# STRART YOUR CODE HERE  #
#=========================#
mytree.train('type', 'gain_ratio')
test_acc = mytree.test('type')
#=========================#
#   END YOUR CODE HERE   #
#=========================#
print('Test accuracy is: ', test_acc)
```

```
best_feature is:  feathers
best_feature is:  backbone
best_feature is:  airborne
best_feature is:  predator
best_feature is:  milk
best_feature is:  fins
best_feature is:  legs
Test accuracy is:  0.8095238095238095
```

Question

Which measure do you like the most and why?

Your answer here:

I prefer information gain as it has a higher training accuracy. I think this might be becuase that gain ratio prefers unbalanced split while the data here has a close to balanced split so the gain ratio is not needed to prevent huge-number splits getting preferred.

2. SVM

This workbook will walk you through a SVM.

2.1 Support vectors and decision boundary

Note: for this question you can work entirely in the Jupyter Notebook, no need to edit any .py files.

Consider classifying the following 20 data points in the 2-d plane with class label y

In [71]:
```
ds = pd.read_csv('data/svm-2d-data.csv')
ds.head()
# This command above will print out the first five data points
# in the dataset with column names as "x1", "x2" and "y"
# You may use command "ds" to show the entire dataset, which contains 20 data poi
```

Out[71]:

	x1	x2	y
0	0.52	-1.00	1
1	0.91	0.32	1
2	-1.48	1.23	1
3	0.01	1.44	1
4	-0.46	-0.37	1

Suppose by solving the dual form of the quadratic programming of svm, we can derive the α_i's for each data point as follows: Among $j = 0, 1, \cdots, 19$ (note that the index starts from 0), α_1 = 0.5084, α_5 = 0.4625, α_{17} = 0.9709, and α_j = 0 for all other j.

Questions

1. Which vectors in the training points are support vectors?
2. What is the normal vector of the hyperplane w?
3. What is the bias b?
4. With the parameters w and b, we can now use our SVM to do predictions. What is predicted label of $x_{new} = (2, -0.5)$? Write out your $f(x_{new})$.

5. A plot of the data points has been generated for you. Please change the `support_vec` variable such that only the support vectors are indicated by red circles. Please also fill in the code to draw the decision boundary. Does your prediction of part 4 seems right visually on the plot?

Your answer here

Note: you can use several code cells to help you compute the results and answer the questions. Again you don't need to edit any .py files.

Please type your answer here!

answer 1:
Vector 1, vector 5 and vector 17 are the support vectors.

answer 2:
x1 = (0.91,0.32), x5 = (0.41,2.04), x17 = (2.05,1.54)
$w = \alpha_1 * x_1 + \alpha_5 * x_5 + \alpha_{17} * x_{17} = (2.642614, 2.601374)^T$

answer 3:
$b = 1/3 * (y_1 - w^T x_1) + 1/3 * (y_5 - w^T x_5) + 1/3 * (y_{17} - w^T x_{17}) = 1/3 * (-2.23721842)$ -
-6.01698926

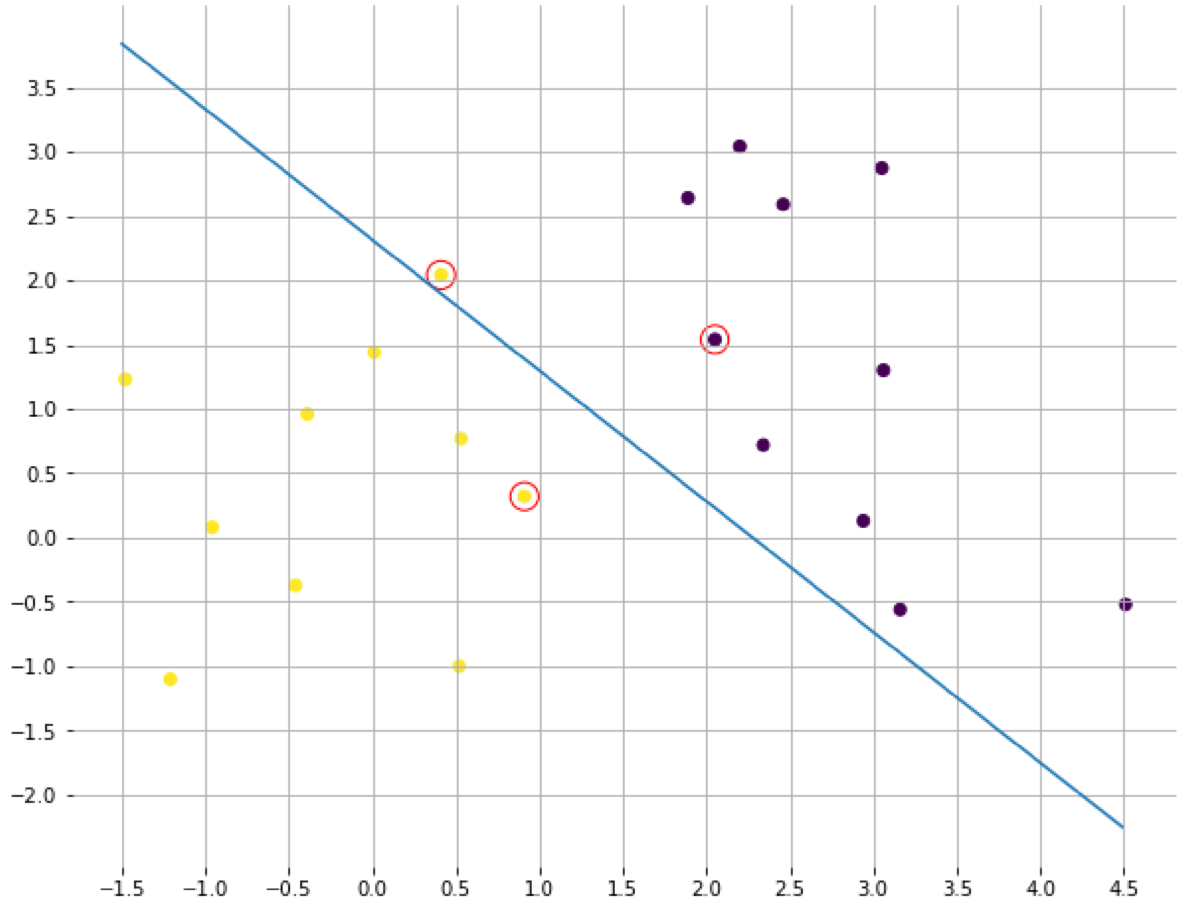
answer 4:
$f(x_{new}) = w * x_{new} + b = 2.642614 * 2 + 2.601374 * (-0.5) - 6.01698926 = -2.0324482$
Predict label of x_{new} is -1.

In [72]:
```
# answer 5
# The prediction seems correct as (2,0,5) belongs in the -1 group.
x1_range = np.arange(-2, 5, 0.5)
x2_range = np.arange(-2, 4., 0.5)

fig, ax = plt.subplots(figsize=(10, 8))
ax = fig.gca()
ax.set_xticks(x1_range)
ax.set_yticks(x2_range)
ax.grid()
ax.scatter(ds['x1'], ds['x2'], c=ds['y'])

support_vec = ds
#=========================#
# STRART YOUR CODE HERE  #
#=========================#
support_vec = ds.loc[[1,5,17]]
x1_values = [i for i in np.arange(-1.5,4.5,0.01)]
w = [2.642614,2.601374]
b = -6.01698926
#w1*x1+w2*x2+b = 0 ---> x2 = (-b-w1x1)/w2
x2_values = []
for x in x1_values:
    x2_values.append((-b-w[0]*x)/w[1])
plt.plot(x1_values,x2_values)

#=========================#
#   END YOUR CODE HERE   #
#=========================#
ax.scatter(support_vec['x1'], support_vec['x2'], marker='o', facecolor='none', s=
sns.despine(ax=ax, left=True, bottom=True, offset=0)
plt.show()
```



2.2 Coding SVM

In this section, we are going to use SVM for classifying the `y` value of 4-dimensional data points. The dataset has been preprocessed and splited into `svm-train.csv` and `svm-test.csv` for you.

For this question we are going to use the `cvxopt` package to help us solve the optimization problem of SVM. You will see it in the .py files, but you don't need to any coding with it. For this question, you only need to implement the right kernel function, and your kernel matrix `K` in `svm.py` line 135 will be pluged in the cvxopt optimization problem solver.

For more information about `cvxopt` please refer to [http://cvxopt.org/ (http://cvxopt.org/)](http://cvxopt.org/)

In [73]:
```
from hw2code.svm import SVM
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
# As a sanity check, we print out the size of the training data (1098, 4) and (1
print('Training data shape: ', svm.train_x.shape, svm.train_y.shape)
print('Testing data shape:', svm.test_x.shape, svm.test_y.shape)
```

```
Training data shape:  (1098, 4) (1098,)
Testing data shape: (274, 4) (274,)
```

2.2.1 Linear kernel

Complete the `SVM.predict` and `linear_kernel` function in `svm.py` . Train a hard margin SVM and a soft margin SVM with linear kernel. Print the test accuracy for both cases.

In [74]:
```
from hw2code.svm import SVM
svm_hard = SVM()
svm_hard.load_data('./data/svm-train.csv', './data/svm-test.csv')
hard_test_acc = 0
#========================#
# STRART YOUR CODE HERE  #
#========================#
svm_hard.train('linear_kernel')
hard_test_acc = svm_hard.test()
#========================#
#   END YOUR CODE HERE   #
#========================#

svm_soft = SVM()
svm_soft.load_data('./data/svm-train.csv', './data/svm-test.csv')
soft_test_acc = 0
#========================#
# STRART YOUR CODE HERE  #
#========================#
svm_soft.train('linear_kernel',100)
soft_test_acc = svm_soft.test()
#========================#
#   END YOUR CODE HERE   #
#========================#
print('Hard margin test accuracy is: ', hard_test_acc)
print('Soft margin test accuracy is: ', soft_test_acc)
```

```
1098 support vectors out of 1098 points
30 support vectors out of 1098 points
Hard margin test accuracy is:  0.5547445255474452
Soft margin test accuracy is:  0.9890510948905109
```

Questions

Are these two results similar? Why or why not?

Your Answer

No the answers are different. For Hard Margin kernel, the cvxopt cannot find the solution as the test dataset is not linearly separable. So it aborted the job and made all vectors the support vectors which is an ill case. This results in bad test accuracy with only 55%, similar to random guessing. For Soft Margin, the model now tolerates some misclassified data and in this case had a optimal performance with 98% test accuracy.

2.2.2 Polynomial kernel

Complete the `polynomial_kernel` function in `svm.py` . Train a soft margin SVM with degree 3 polynomial kernel and parameter `C = 100` for the regularization term. Print the test accuracy.

In [75]:
```
from hw2code.svm import SVM
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#========================#
# STRART YOUR CODE HERE  #
#========================#
svm.train("polynomial_kernel", 100)
test_acc = svm.test()
#========================#
#   END YOUR CODE HERE   #
#========================#
print('Test accuracy is: ', test_acc)
```

```
19 support vectors out of 1098 points
Test accuracy is:  0.9562043795620438
```

Questions

Is the result better than linear kernel? Why or why not?

Your Answer

It does not seem better compared with the soft margin linear kernel. I am guessing the linearity is enough to classify the test dataset so adding higher-dimension attribute will not improve the accuracy.

2.2.3 Gaussian kernel

Complete the `gaussian_kernel` function using the `gaussian_kernel_point` in `svm.py` . Train a soft margin SVM with Gaussian kernel and parameter `C = 100` for the regularization term. Print the test accuracy.

In [76]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#=========================#
# STRART YOUR CODE HERE   #
#=========================#
svm.train("gaussian_kernel", 100)
test_acc = svm.test()
#=========================#
#   END YOUR CODE HERE    #
#=========================#
print('Test accuracy is: ', test_acc)
```

```
35 support vectors out of 1098 points
Test accuracy is:  1.0
```

Questions

1. Is the result better than linear kernel and polynomial kernel? Why or why not?
2. Which one of these four models do you like the most and why?
3. (Bonus question, optional) Can you come up with a vectorized implementation of `gaussian_kernel` without calling `gaussian_kernel_point` ? Fill that in svm.py.

Your Answer

Please write down your answers and/or observations here

answer 1: The result is better than both, probably because the test dataset has a normal distribution tendency so it fits the model well.

answer 2: I actually like the Soft Margin Linear Kernel as this reaches a high test accuracy with much less computation.

End of Homework 2 :)

After you've finished the homework, please print out the entire `ipynb` notebook and two `py` files into one PDF file. Make sure you include the output of code cells and answers for questions. Prepare submit it to GradeScope. Also this time remember assign the pages to the questions on GradeScope

```python
import pandas as pd
import numpy as np
from pprint import pprint
import sys

# Reads the data from CSV files, each attribute column can be obtained via its name, e.g., y = data['y']
def getDataframe(filePath):
    data = pd.read_csv(filePath)
    return data

# predicted_y and y are the predicted and actual y values respectively as numpy arrays
# function prints the accuracy
def compute_accuracy(predicted_y, y):
    acc = 100.0
    acc = np.sum(predicted_y == y)/predicted_y.shape[0]
    return acc

#Compute entropy according to y distribution
def compute_entropy(y):
    entropy = 0.0
    elements,counts = np.unique(y, return_counts = True)
    n = y.shape[0]

    for i in range(len(elements)):
        prob = counts[i]/n
        if prob!= 0:
            entropy -= prob * np.log2(prob)
    return entropy

#att_name: attribute name; y_name: the target attribute name for classification
def compute_info_gain(data, att_name, y_name):
    info_gain = 0.0

    #Calculate the values and the corresponding counts for the select attribute
    vals, counts = np.unique(data[att_name], return_counts=True)
    total_counts = np.sum(counts)

    #Calculate the conditional entropy
    #========================#
    # STRART YOUR CODE HERE  #
    #========================#
    info = compute_entropy(data[y_name])
    entropy = 0
    for i in range(len(vals)):
        probability = counts[i] / total_counts
        val_data = data.loc[data[att_name] == vals[i]]
        entropy += probability * compute_entropy(val_data[y_name])
    info_gain = info - entropy
    #========================#
    #   END YOUR CODE HERE   #
    #========================#

    return info_gain


def comput_gain_ratio(data, att_name, y_name):
    gain_ratio = 0.0
    #Calculate the values and the corresponding counts for the select attribute
```

```python
    vals, counts = np.unique(data[att_name], return_counts=True)
    total_counts = np.sum(counts)

    #Calculate the information for the selected attribute
    att_info = 0.0
    #========================#
    # STRART YOUR CODE HERE  #
    #========================#
    for i in range(len(counts)):
        probability = counts[i] / total_counts
        att_info -= probability * np.log2(probability)
    #========================#
    #   END YOUR CODE HERE   #
    #========================#
    gain_ratio = 0.0 if np.abs(att_info) < 1e-9 else min(1, compute_info_gain(data, att_name, y_name) / att_info)
    return gain_ratio

# Class of the decision tree model based on the ID3 algorithm
class DecisionTree(object):
    def __init__(self):
        self.train_data = pd.DataFrame()
        self.test_data = pd.DataFrame()

    def load_data(self, train_file, test_file):
        self.train_data = getDataframe(train_file)
        self.test_data = getDataframe(test_file)

    def train(self, y_name, measure, parent_node_class= None):
        self.y_name = y_name
        self.measure = measure
        self.tree = self.make_tree(self.train_data, parent_node_class)

    def make_tree(self, train_data, parent_node_class = None):
        data = train_data
        features = data.drop(self.y_name, axis = 1).columns.values
        measure =  self.measure
        #Stopping condition 1: If all target_values have the same value, return this value
        if len(np.unique(data[self.y_name])) <= 1:
            leaf_value = -1
            #========================#
            # STRART YOUR CODE HERE  #
            #========================#
            leaf_value = data[self.y_name].values[0]
            #========================#
            #   END YOUR CODE HERE   #
            #========================#
            return leaf_value

        #Stopping condition 2: If the dataset is empty, return the parent_node_class
        elif len(data)== 0:
            return parent_node_class

        #Stopping condition 3: If the feature space is empty, return the majority class
        elif len(features) == 0:
            return np.unique(data[self.y_name])[np.argmax(np.unique(data[y_name],return_counts=True)[1])]

```

```python
        # Not a leaf node, create an internal node
        else:
            #Set the default value for this node --> The mode target feature value of the current node
            parent_node_class = np.unique(data[self.y_name])[np.argmax(np.unique(data[self.y_name],return_counts=True)[1])]

            #Select the feature which best splits the dataset
            if measure == 'info_gain':
                item_values = [compute_info_gain(data, feature, self.y_name) for feature in features] #Return the information gain values for the features in the dataset
            elif measure == 'gain_ratio':
                item_values = [comput_gain_ratio(data, feature, self.y_name) for feature in features] #Return the gain_ratio for the features in the dataset
            else:
                raise ValueError("kernel not recognized")

            best_feature_index = np.argmax(item_values)
            best_feature = features[best_feature_index]
            print('best_feature is: ', best_feature)

            #Create the tree structure. The root gets the name of the feature (best_feature)
            tree = {best_feature:{}}


        #Grow a branch under the root node for each possible value of the root node feature

        for value in np.unique(data[best_feature]):
            #Split the dataset along the value of the feature with the largest information gain and therwith create sub_datasets
            sub_data = data.where(data[best_feature] == value).dropna()

            #Remove the selected feature from the feature space
            sub_data = sub_data.drop(best_feature, axis = 1)

            #Call the ID3 algorithm for each of those sub_datasets with the new parameters --> Here the recursion comes in!
            subtree = self.make_tree(sub_data, parent_node_class)

            #Add the sub tree, grown from the sub_dataset to the tree under the root node
            tree[best_feature][value] = subtree

        return tree


    def test(self, y_name):
        accuracy = self.classify(self.test_data, y_name)
        return accuracy

    def classify(self, test_data, y_name):
        #Create new query instances by simply removing the target feature column from the test dataset and
        #convert it to a dictionary
        test_x = test_data.drop(y_name, axis=1)
        test_y = test_data[y_name]

```

```
        n = test_data.shape[0]
        predicted_y = np.zeros(n)

        #Calculate the prediction accuracy
        for i in range(n):
            predicted_y[i] = DecisionTree.predict(self.tree, test_x.iloc[i])

        output = np.zeros((n,2))
        output[:,0] = test_y
        output[:,1] = predicted_y
        accuracy = compute_accuracy(predicted_y, test_y.values)
        return accuracy

    def predict(tree, query):
        # find the root attribute
        default = -1
        for root_name in list(tree.keys()):
            try:
                subtree = tree[root_name][query[root_name]]
            except:
                return default ## root_name does not appear in query attribute list (it is an error!)

            ##if subtree is still a dictionary, recursively test next attribute
            if isinstance(subtree,dict):
                return DecisionTree.predict(subtree, query)
            else:
                leaf = subtree
                return leaf


```

```python
import numpy as np
from numpy import linalg
import cvxopt
import cvxopt.solvers
import sys
import pandas as pd
cvxopt.solvers.options['show_progress'] = False

# Reads the data from CSV files, converts it into Dataframe and returns x and y dataframes
def getDataframe(filePath):
    dataframe = pd.read_csv(filePath)
    y = dataframe['y']
    x = dataframe.drop('y', axis=1)
    y = y*2 -1.0
    return x.to_numpy(), y.to_numpy()

def compute_accuracy(predicted_y, y):
    acc = 100.0
    acc = np.sum(predicted_y == y)/predicted_y.shape[0]
    return acc

def gaussian_kernel_point(x, y, sigma=5.0):
    return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))

def linear_kernel(X, Y=None):
    Y = X if Y is None else Y
    m = X.shape[0]
    n = Y.shape[0]
    assert X.shape[1] == Y.shape[1]
    kernel_matrix = np.zeros((m, n))
    #========================#
    # STRART YOUR CODE HERE  #
    #========================#
    for i in range(m):
        for j in range(n):
            kernel_matrix[i,j] = np.dot(X[i], Y[j])
    #========================#
    #   END YOUR CODE HERE   #
    #========================#
    return kernel_matrix

def polynomial_kernel(X, Y=None, degree=3):
    Y = X if Y is None else Y
    m = X.shape[0]
    n = Y.shape[0]
    assert X.shape[1] == Y.shape[1]
    kernel_matrix = np.zeros((m, n))
    #========================#
    # STRART YOUR CODE HERE  #
    #========================#
    for i in range(m):
        for j in range(n):
            kernel_matrix[i,j] = (1+np.dot(X[i],Y[j])) ** degree
    #========================#
    #   END YOUR CODE HERE   #
    #========================#
    return kernel_matrix

def gaussian_kernel(X, Y=None, sigma=5.0):
```

```python
    Y = X if Y is None else Y
    m = X.shape[0]
    n = Y.shape[0]
    assert X.shape[1] == Y.shape[1]
    kernel_matrix = np.zeros((m, n))
    #=======================#
    # STRART YOUR CODE HERE  #
    #=======================#
    for i in range(m):
        for j in range(n):
            kernel_matrix[i,j] = np.exp((-np.linalg.norm(X[i]-Y[j])**2)/ (2*(sigma**2)))
    #=======================#
    #   END YOUR CODE HERE   #
    #=======================#
    return kernel_matrix


# Bonus question: vectorized implementation of Gaussian kernel
# If you decide to do the bonus question, comment the gaussian_kernel function above,
# then implement and uncomment this one.
# def gaussian_kernel(X, Y=None, sigma=5.0):
#     return

class SVM(object):
    def __init__(self):
        self.train_x = pd.DataFrame()
        self.train_y = pd.DataFrame()
        self.test_x = pd.DataFrame()
        self.test_y = pd.DataFrame()
        self.kernel_name = None
        self.kernel = None

    def load_data(self, train_file, test_file):
        self.train_x, self.train_y = getDataframe(train_file)
        self.test_x, self.test_y = getDataframe(test_file)


    def train(self, kernel_name='linear_kernel', C=None):
        self.kernel_name = kernel_name
        if(kernel_name == 'linear_kernel'):
            self.kernel = linear_kernel
        elif(kernel_name == 'polynomial_kernel'):
            self.kernel = polynomial_kernel
        elif(kernel_name == 'gaussian_kernel'):
            self.kernel = gaussian_kernel
        else:
            raise ValueError("kernel not recognized")

        self.C = C
        if self.C is not None:
            self.C = float(self.C)

        self.fit(self.train_x, self.train_y)

    # predict labels for test dataset
    def predict(self, X):
        if self.w is not None: ## linear case
            n = X.shape[0]
            predicted_y = np.zeros(n)
```

```
            #========================#
            # STRART YOUR CODE HERE  #
            #========================#
            predicted_y = np.dot(self.w, np.transpose(X)) + self.b
            #========================#
            #   END YOUR CODE HERE   #
            #========================#
            return predicted_y

        else: ## non-linear case
            n = X.shape[0]
            predicted_y = np.zeros(n)
            #========================#
            # STRART YOUR CODE HERE  #
            #========================#
            kernel_result = self.kernel(X,self.sv)
            for i in range(n):
                for j in range(self.sv.shape[0]):
                    predicted_y[i] += self.a[j]*self.sv_y[j]*kernel_result[i,j]
            #========================#
            #   END YOUR CODE HERE   #
            #========================#
            return predicted_y

    #==============================================#
    # Please DON'T change any code below this line!  #
    #==============================================#
    def fit(self, X, y):
        n_samples, n_features = X.shape
        # Kernel matrix
        K = self.kernel(X)

        # dealing with dual form quadratic optimization
        P = cvxopt.matrix(np.outer(y,y) * K)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y, (1,n_samples),'d')
        b = cvxopt.matrix(0.0)

        if self.C is None:
            G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            tmp1 = np.diag(np.ones(n_samples) * -1)
            tmp2 = np.identity(n_samples)
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(n_samples)
            tmp2 = np.ones(n_samples) * self.C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        # solve QP problem
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)
        # Lagrange multipliers
        a = np.ravel(solution['x'])

        # Support vectors have non zero lagrange multipliers
        sv = a > 1e-5
        ind = np.arange(len(a))[sv]
        self.a = a[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]
```

```python

        print("%d support vectors out of %d points" % (len(self.a), n_samples))

        # Intercept via average calculating b over support vectors
        self.b = 0
        for n in range(len(self.a)):
            self.b += self.sv_y[n]
            self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv])
        self.b /= len(self.a)

        # Weight vector
        if self.kernel_name == 'linear_kernel':
            self.w = np.zeros(n_features)
            for n in range(len(self.a)):
                self.w += self.a[n] * self.sv_y[n] * self.sv[n]
        else:
            self.w = None


    def test(self):
        accuracy = self.classify(self.test_x, self.test_y)
        return accuracy

    def classify(self, X, y):
        predicted_y = np.sign(self.predict(X))
        accuracy = compute_accuracy(predicted_y, y)
        return accuracy
```