20F-COMSCI145 HW2

DANNING LIU YU

TOTAL POINTS

103 / 105

QUESTION 1

Decision trees 40 pts

1.1 Attribute selection measures 23 / 25

✓ **- 2 pts** answer 1 choice

1.2 Information gain 5 / 5

✓ **- 0 pts** Correct

1.3 Gain ratio 5 / 5

✓ **- 0 pts** Correct

1.4 Question 5 / 5

✓ **- 0 pts** Correct

QUESTION 2

SVM 65 pts

2.1 Support vectors and decision boundary 25 / 25

✓ **- 0 pts** Correct

2.2 Linear kernel 13 / 13

✓ **- 0 pts** Correct

2.3 Polynomial kernel 8 / 8

✓ **- 0 pts** Correct

2.4 Gaussian kernel 8 / 8

✓ **- 0 pts** Correct

2.5 Question 6 / 6

✓ **- 0 pts** Correct

2.6 Bonus 5 / 5

✓ **- 0 pts** Correct

```
%load_ext autoreload
%autoreload 2
```

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.

Please type your answer here!

Question 1

Split on attribute 1: R_m for node11 and node12 is both 0.25. Thus, misclassification rate is $\frac{(0.25)(400)+(0.25)(400)}{800} = 0.25$.
Split on attribute 2. R_m for node21 is 0.333 and 0 for node22.Thus, misclassification rate is $\frac{(0.333)(600)+(0)(200)}{800} = 0.25$.

Both attributes have the same misclassification rate, and thus one is not particularly better than the other. In terms of prefernece, since both have the same performance, I can't use that as a criteria to pick, but I prefer the resulting distribution of the data when we split on attribute 1, as there's a more even split of 400 values for each node.

Question 2

Node11: $H = -\left(\frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{3}{4}\log_2\left(\frac{3}{4}\right)\right) = 0.811$
Node12: $H = -\left(\frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{3}{4}\log_2\left(\frac{3}{4}\right)\right) = 0.811$
Node21: $H = -\left(\frac{1}{3}\log_2\left(\frac{1}{3}\right) + \frac{2}{3}\log_2\left(\frac{2}{3}\right)\right) = 0.918$
Node22: $H = -\left(1\log_2(1)\right) = 0$

Question 3

Before split: $H\left(\text{gender}\right) = -\left(\frac{1}{2}\log_2\left(\frac{1}{2}\right) + \frac{1}{2}\log_2\left(\frac{1}{2}\right)\right) = 1$

Split on attribute 1: $H\left(\text{gender}|\text{attribute 1}\right) = \frac{1}{2}(0.811) + \frac{1}{2}(0.811) = 0.811$
Info gain on attribute 1: $1 - 0.811 = 0.189$

Split on attribute 2: $H\left(\text{gender}|\text{attribute 2}\right) = \frac{3}{4}(0.918) + \frac{1}{4}(0) = 0.689$
Info gain on attribute 2: $1 - 0.689 = 0.311$

Question 4

I would prefer splitting on attribute 2 because it maximizes information gain. Entropy is a measure of uncertainty and maximizing information gain means minimizing entropy, thus making information gain a measurement of uncertainty.

Question 5

Attribute 1: $\text{SplitInfo}(\text{attribute 1}) = -\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right) = 1$
$\text{Gain\_ratio}(\text{attribute 1}) = \frac{0.189}{1} = 0.189$

Attribute 2: $\text{SplitInfo}(\text{attribute 2}) = -\left(\frac{3}{4}\log_2\frac{3}{4} + \frac{1}{4}\log_2\frac{1}{4}\right) = 0.811$
$\text{Gain\_ratio}(\text{attribute 2}) = \frac{0.311}{0.811} = 0.383$

Using gain ratio, I prefer splitting on attribute 2 because it has a higher gain ratio. This is the same result as splitting on information gain.

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.

1.1 Attribute selection measures **23 / 25**

✓ **- 2 pts** answer 1 choice

In [3]:
```
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 testing da
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 [32]:
```
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 [26]:
```
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
```

1.2 Information gain **5 / 5**

✓ **- 0 pts** **Correct**

In [3]:
```
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 testing da
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 [32]:
```
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 [26]:
```
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 like information gain more because it seems to be more suited for this particular dataset, as it had a testing accuracy of 0.8571, while gain ratio had an accuracy of 0.8095. Both methods have their pros and cons, so picking which one to use really depends on the data set and the specific situation. In this case, each attribute was binary (has or does not), so there is not the issue of information gain being biased towards multivalued attributes. In general, I think both are cool and provide a very understandable and sensical way of constructing a decision tree.

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 [2]:
```
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 points
print(ds)
```

```
      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
5   0.41  2.04  1
6   0.53  0.77  1
7  -1.21 -1.10  1
8  -0.39  0.96  1
9  -0.96  0.08  1
10  2.46  2.59 -1
11  3.05  2.87 -1
12  2.20  3.04 -1
13  1.89  2.64 -1
14  4.51 -0.52 -1
15  3.06  1.30 -1
16  3.16 -0.56 -1
17  2.05  1.54 -1
18  2.34  0.72 -1
19  2.94  0.13 -1
```

Suppose by solving the dual form of the quadratic programming of svm, we can derive the α_i's for

1.3 Gain ratio **5 / 5**

✓ **- 0 pts** **Correct**

```
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 like information gain more because it seems to be more suited for this particular dataset, as it had a testing accuracy of 0.8571, while gain ratio had an accuracy of 0.8095. Both methods have their pros and cons, so picking which one to use really depends on the data set and the specific situation. In this case, each attribute was binary (has or does not), so there is not the issue of information gain being biased towards multivalued attributes. In general, I think both are cool and provide a very understandable and sensical way of constructing a decision tree.

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 [2]:
```
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 points
print(ds)
```

```
      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
5   0.41  2.04  1
6   0.53  0.77  1
7  -1.21 -1.10  1
8  -0.39  0.96  1
9  -0.96  0.08  1
10  2.46  2.59 -1
11  3.05  2.87 -1
12  2.20  3.04 -1
13  1.89  2.64 -1
14  4.51 -0.52 -1
15  3.06  1.30 -1
16  3.16 -0.56 -1
17  2.05  1.54 -1
18  2.34  0.72 -1
19  2.94  0.13 -1
```

Suppose by solving the dual form of the quadratic programming of svm, we can derive the α_i's for

1.4 Question 5 / 5

✓ - 0 pts Correct

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!

Question 1

Any point with $\alpha_i \neq 0$ is a support vector: thus, the vectors $v_1 = [\,0.91 \quad 0.32\,]^T$, $v_5 = [\,0.41 \quad 2.04\,]^T$, and $v_{17} = [\,2.05 \quad 1.54\,]^T$.

Question 2

$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x_i}$
$= (0.5084)(1)[\,0.91 \quad 0.32\,]^T$
$+(0.4625)(1)[\,0.41 \quad 2.04\,]^T$
$+(0.9709)(-1)[\,2.05 \quad 1.54\,]^T = [\,-1.338 \quad -0.389\,]^T$

Question 3

$b = y_k - \mathbf{w^T x_k}$ for any $\mathbf{x_k}$ such that $\alpha_k \neq 0$.
We'll use point 1:
$b = 1 - ((-1.338)(0.91) + (-0.389)(0.32)) = 2.342$

Question 4

$f(\mathbf{x}_{new}) = \mathbf{w^T x}_{new} + b = ((-1.338)(2) + (-0.389)(-0.5)) + 2.342 = -0.1395$

Question 5

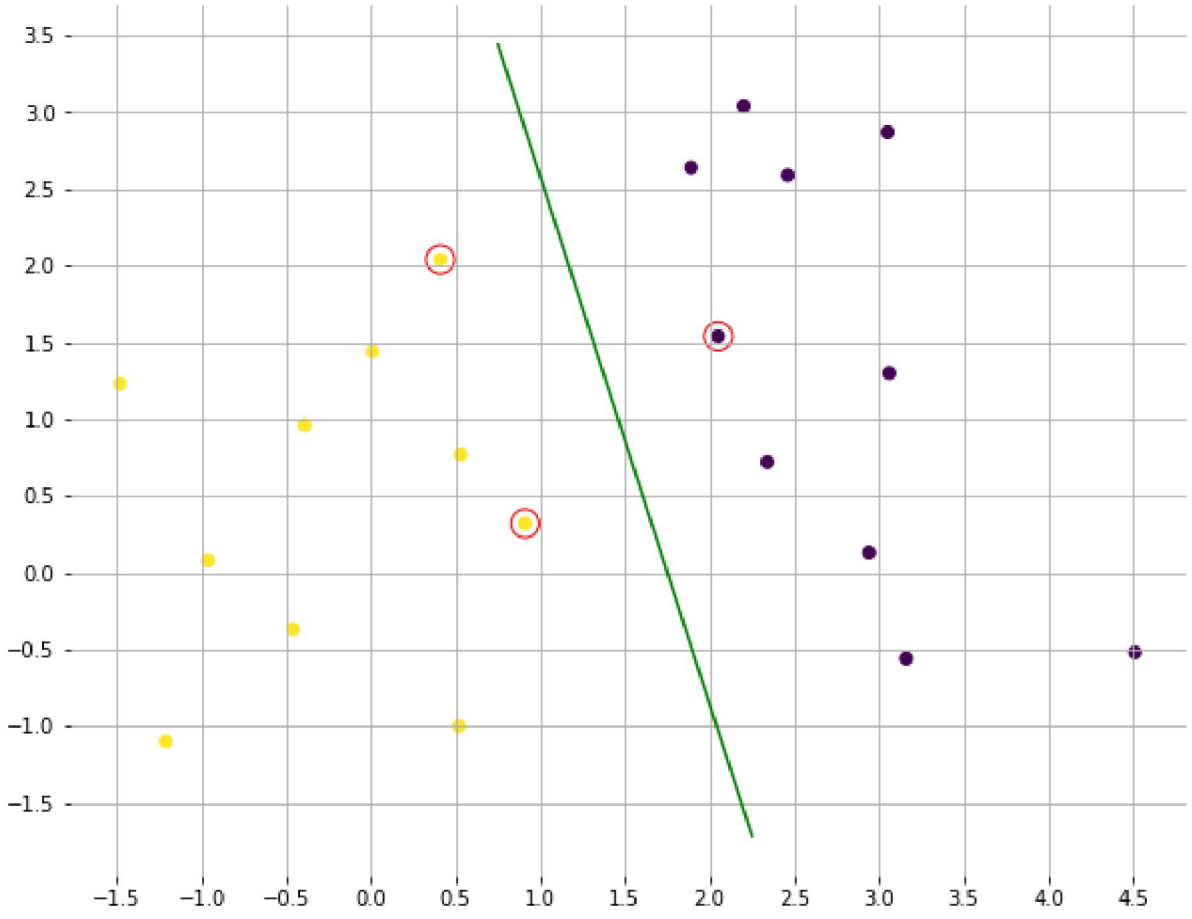
Based on the plot of the decision boundary (see below) and the fact that the 3 closest points are circled red, the plot makes sense.

In [4]:
```
# answer 5
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  #
#=======================#
x1_vals = np.arange(0.75, 2.25, 0.001)
x2_vals = -1 * (x1_vals * -1.338 + 2.342) / -0.389 # taken from w and b
plt.plot(x1_vals, x2_vals, "g-")
support_vec = pd.concat([ds.iloc[1], ds.iloc[5], ds.iloc[17]])
#=======================#
#   END YOUR CODE HERE   #
#=======================#
ax.scatter(support_vec['x1'], support_vec['x2'], marker='o', facecolor='none', s=200, c
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.

2.1 Support vectors and decision boundary **25 / 25**

✓ **- 0 pts** **Correct**

For more information about `cvxopt` please refer to http://cvxopt.org/

In [3]:
```
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 (1098,) a
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 [42]:
```
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(kernel_name="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(kernel_name="linear_kernel", C=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

The results are very different - in hard margin, all 1098 points are support vectors, while for soft margin SVM, only 30 data points are support vectors. This also shows in the test accuracy, which is only 0.555 for hard margin but 0.989 for soft margin. This is probably because the data is very difficult to "classify" or linearly separate, with many points laying close to the decision boundary or

on the wrong side, but once some slackness is allowed, the SVM can handle the points that are on the wrong side of the boundary.

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 [41]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#=======================#
# STRART YOUR CODE HERE  #
#=======================#
svm.train(kernel_name="polynomial_kernel", C=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:  0.927007299270073
```

Questions

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

Your Answer

The result is better than linear kernel with hard margin (0.5547) but not as good as linear kernel with soft margin (0.9891). It makes sense that the accuracy is better than the hard margin linear kernel, as that one had a very low test accuracy rate and a polynomial of degree 3 can better handle data that is hard to linearly separate. However, a polynomial of degree 3 may not be the most ideal for this mode, causing it to have a slightly lower test accuracy compared to the soft margin linear SVM. This could be due to overfitting or the transformation to degree 3 not being expressive enough. But either way, the accuracy of 0.927 is still pretty good.

Note: As expected, when I changed the degree to 1, I got the same testing accuracy as soft-margin linear SVM.

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 [46]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#=======================#
# STRART YOUR CODE HERE  #
#=======================#
svm.train(kernel_name="gaussian_kernel", C=100)
test_acc = svm.test()
#=======================#
```

2.2 Linear kernel 13 / 13

✓ - 0 pts Correct

on the wrong side, but once some slackness is allowed, the SVM can handle the points that are on the wrong side of the boundary.

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 [41]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#=======================#
# STRART YOUR CODE HERE  #
#=======================#
svm.train(kernel_name="polynomial_kernel", C=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:  0.927007299270073
```

Questions

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

Your Answer

The result is better than linear kernel with hard margin (0.5547) but not as good as linear kernel with soft margin (0.9891). It makes sense that the accuracy is better than the hard margin linear kernel, as that one had a very low test accuracy rate and a polynomial of degree 3 can better handle data that is hard to linearly separate. However, a polynomial of degree 3 may not be the most ideal for this mode, causing it to have a slightly lower test accuracy compared to the soft margin linear SVM. This could be due to overfitting or the transformation to degree 3 not being expressive enough. But either way, the accuracy of 0.927 is still pretty good.

Note: As expected, when I changed the degree to 1, I got the same testing accuracy as soft-margin linear SVM.

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 [46]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#=======================#
# STRART YOUR CODE HERE  #
#=======================#
svm.train(kernel_name="gaussian_kernel", C=100)
test_acc = svm.test()
#=======================#
```

2.3 Polynomial kernel **8 / 8**

✓ **- 0 pts** **Correct**

on the wrong side, but once some slackness is allowed, the SVM can handle the points that are on the wrong side of the boundary.

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 [41]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#=======================#
# STRART YOUR CODE HERE  #
#=======================#
svm.train(kernel_name="polynomial_kernel", C=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:  0.927007299270073
```

Questions

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

Your Answer

The result is better than linear kernel with hard margin (0.5547) but not as good as linear kernel with soft margin (0.9891). It makes sense that the accuracy is better than the hard margin linear kernel, as that one had a very low test accuracy rate and a polynomial of degree 3 can better handle data that is hard to linearly separate. However, a polynomial of degree 3 may not be the most ideal for this mode, causing it to have a slightly lower test accuracy compared to the soft margin linear SVM. This could be due to overfitting or the transformation to degree 3 not being expressive enough. But either way, the accuracy of 0.927 is still pretty good.

Note: As expected, when I changed the degree to 1, I got the same testing accuracy as soft-margin linear SVM.

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 [46]:
```
svm = SVM()
svm.load_data('./data/svm-train.csv', './data/svm-test.csv')
test_acc = 0
#=======================#
# STRART YOUR CODE HERE  #
#=======================#
svm.train(kernel_name="gaussian_kernel", C=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

Question 1

The result, with a test accuracy of 1.0, is better than that of both linear and polynomial kernel. This is because the RBF kernel is a mapping from the given domain to an infinite-domain, and as long as you keep increasing the dimensionality of the transformed data, you can eventually separate it. Thus, this is what has been done with the RBF kernel, resulting in a data that is well separated even for the testing set and thus attaining a very high accuracy of 1.0

Question 2

I like linear kernel with soft margin the most because it is a simple idea that is easily interpretable--draw a linear boundary and have some slack for difficult to classify points. When you apply transformations such as polynomial or RBF, those are simply elaborations that turn data that might not be linearly separable into data that is, but they still build on the idea of the linear kernel. In terms of accuracy, RBF would be the best choice, with soft-margin coming in a close second.

Question 3

See `svm.py` for the implementation. Basically, it takes advantage of the fact that $\|\boldsymbol{x}-\boldsymbol{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\mathbf{x}^\mathbf{T}\mathbf{y}$.

Using `%timeit`, the nonvectorized verion runs for ~22.5 seconds on average, while the vectorized version runs for ~5.5 seconds on average, which is much faster.

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

2.4 Gaussian kernel **8 / 8**

✓ **- 0 pts** **Correct**

```
#   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

Question 1

The result, with a test accuracy of 1.0, is better than that of both linear and polynomial kernel. This is because the RBF kernel is a mapping from the given domain to an infinite-domain, and as long as you keep increasing the dimensionality of the transformed data, you can eventually separate it. Thus, this is what has been done with the RBF kernel, resulting in a data that is well separated even for the testing set and thus attaining a very high accuracy of 1.0

Question 2

I like linear kernel with soft margin the most because it is a simple idea that is easily interpretable--draw a linear boundary and have some slack for difficult to classify points. When you apply transformations such as polynomial or RBF, those are simply elaborations that turn data that might not be linearly separable into data that is, but they still build on the idea of the linear kernel. In terms of accuracy, RBF would be the best choice, with soft-margin coming in a close second.

Question 3

See `svm.py` for the implementation. Basically, it takes advantage of the fact that $\|\boldsymbol{x}-\boldsymbol{y}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\mathbf{x}^\mathbf{T}\mathbf{y}$.

Using `%timeit`, the nonvectorized verion runs for ~22.5 seconds on average, while the vectorized version runs for ~5.5 seconds on average, which is much faster.

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

2.5 Question 6 / 6

✓ - 0 pts Correct

```
#    #=======================#
#    # STRART YOUR CODE HERE  #
#    #=======================#
#    for i in range(m):
#        for j in range(n):
#            kernel_matrix[i][j] = gaussian_kernel_point(X[i], Y[j])
#    #=======================#
#    #   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):
    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))
    x_norm = np.linalg.norm(X, axis=1) ** 2
    y_norm = np.linalg.norm(Y, axis=1) ** 2
    kernel_matrix = np.exp((-1 / (2 * (sigma ** 2)))
                           * (x_norm.reshape(-1, 1)
                              + y_norm.reshape(1, -1)
                              - 2 * np.matmul(X, Y.T)))
    return kernel_matrix

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):
```

2.6 Bonus 5 / 5

✓ - 0 pts Correct