Understanding Decision Boundaries Using Setosa Data

Kate Do Project 3 10/13/22

This workbook is an introduction to the exploration of decision boundaries, how to use them, why we use them, and how to apply them without overly manipulating them. The data set used throughout is comparing sepal width vs length from setosa and non setosa plants. The first and third part of the notebook take different sets to understand how the boundaries are created and look compared to other graphs. Part two however, explores versicolor and virginica data comparing the sepal or petal length and widths. That data in particular is interesting because the boundary lines for sepal numbers aren't as clear as in the setosa data.

Problem 1:

1. Just by looking at your plot, which of the two decision boundaries does a better job of separating the two classes of points?

- The red line (y = 0.75x – 0.9) does the best job. There is only one point on the graph that does not align with it compared to the black line (y = 2x – 8) which has about 6 or 7 points which do not fit the boundary.

2. Which of the two decision boundaries gave the more accurate predictions?

- The red line had an accuracy of 99.3% , the black had an acurracy of 95.3%

3. How does the accuracy metric seem to relate to the ability of the decision boundary to separate the classes?

- The decision boundary is what drives the accuracy metric and vice versa. The boundary's ability to separate the classes can be seen on the graph itself, meaning how many points are on its "incorrect" side of the line. The more points on that side leads to a lower accuracy score because it's judging how many of the point are correctly sorted. So the more points correctly sorted, on its correct side of the boundary line, means a higher accuracy score.

Problem 2:

1. For which pair of features are the classes more easily separated?

- Petal width and length makes the separation much easier to see on the graph compared to sepal.

2. Just by looking at your plots, for which pair of features does the decision boundary do a better job of separating the classes?

- The decision boundary does its best separating classes on the petal feature with only 1-2 points on its incorrect side.

3. Which decision boundary gives the most accurate predictions?

- The petal feature gives the most accurate predictions at 96% accuracy compared to the sepal feature at only 70% accuracy.

4. How does the choice of features seem to impact the ability to make accurate predictions?

- Some items, or in this case flowers have multiple features that make them distinct compared to other flowers. In the first example it was its sepal feature that made it easy to separate setosa and non setosa items. However, there are some features that might be similar such as versicolor and virginica's sepal feature. But that doesn't mean there isn't some other difference they can't have. Looking at multiple features lets us decide which one would be the most helpful in separating classes from each other without having to run multiple trainings on a feature with too many similarities. Choice of feature provides simplicity of separation.

Problem 3:

NOTE: I plotted both equations black : $y = 2x - 8$ and red : $y = 0.75x - .9$ The first equation was asked of us to plot but after seeing success of the second one in the first part of the notebook I wanted to see what the results would look like in this third part that involves training and then evaluating. I will be discussing both lines using their associated color.

1. Compare the accuracies you calculated from the training and testing data sets. Predictions for which data set were evaluated as more accurate? Which accuracy score do you think is a more realistic representation of the performance of the model?

- The training set was the most accurate compared to testing because it's tested and then evaluated on the same data, making it biased towards its training. As you can see in the data below for the black line, the boundary line has near 100% accuracy for training, but only about 86% for the testing data. This is because testing data has information the method hasn't seen yet and is making assumptions of what it should be classified as based off of the training's data. Comparatively the red line is still incredibly accurate with 100% for training, and about 98% for testing.

- Training
  - Black: 99%
  - Red: 100%
- Testing
  - Black: 85.7%
  - Red: 97.6%
- In terms of realistic, the testing one is most aligned with that notion. You shouldn't be using the training data for its own evaluation. Especially multiple times as the line could overfit and only apply to that data set. New data that it hasn't seen or tested yet is what should be happening to understand if the training is working.

2. Look at the scatter plot of the validation data points. Will the model make errors in predicting the labels? Why?

- Yes, this is because the training data doesn't have many points less than 5.5cm in sepal length while the validation data has multiple points doing so.

3. Look at the scatter plots of the training and testing data points. Which data set is more representative of the validation data set? Which data set will demonstrate errors similar to the validation set?

- The testing and validation data sets are the most alike. The setosa points have a much wider spread on both plots compared to the training data's which have a huge cluster from 5.5cm to 7cm in sepal length. The wider spread of testing and validation points go all the way down to 5cm in sepal length and do not cluster like the training data. This means the testing data will also have similar errors to the validation set.

4. Compare the accuracies you calculated from the training and testing data sets to the validation data set. Which accuracy calculation is more representative of the accuracy for the validation data set?

- Training
    - Black: 99%
    - Red: 100%
- Testing
    - Black: 85.7%
    - Red: 97.6%
- Validation
    - Black: 86.3%
    - Red: 100%
- The testing data has the closest accuracy to the validation's black line. This is most likely because of their data set similarities as discussed above in question 3, as well as their training set being the same.

5. Explain why training and evaluating a model on the same data set can be deceptive.

- Decision boundary will be overfit
    - Using the same data set for training over and over again will lead to an overfit decision boundary making the accuracy/results be almost 100% accurate every time but only for that data.
    - Trying to apply it to other data will lead to wildly incorrect results because it's not generalize enough for what other data might be out there
    - That data set will be used as the generalized a specific pattern even though it's not generalized at all, this could lead to incorrect classification once applied to other scenarios like learning about consumer buying over years, but you only studied one year (even worse like a recession year).

6. Explain how dividing data into training and testing sets with no repeated points resolves some of the problems associated with training and evaluating a model on the same data set.

- By dividing the data set to have no repeated points, there's a decrease in risk of having/showing an overfitted boundary. - Giving new points to test on helps to avoid that issue and see how the training would apply in a more real world application. It's still not ideal to train and test on the same data set since the whole set could be biased a certain way, but it's still good to separate points and avoid repeated ones when dividing them.

7. Name 3 potential ramifications for publishing a model that was trained and characterized using the same data set.

- Consumer buying
  - Testing on a certain month or year rather than multiple months/years. This can lead to incorrect stocking and resource allocation
- Personal Interest
  - Testing on a data set of a survey from people who are biased like presidential preference, or music interest, this example will be for a survey if people are interested in taking a certain class like our AI cert classes
  - That data set could be students that are eligible to take it, not eligible etc... and what could happen is the class is set up, the teacher prepares all the material and there's work being allocated to this class, which might be a big hit for one semester but not every or any anymore.
- Exams
  - Studying the practice examples to the point you remember them only to not understand the concepts then taking the exam and not being able to answer any of them correctly and an F

In [1]:
```
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import svm
from sklearn.metrics import accuracy_score
from decision_boundaries import linear_decision_boundary_classifier
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#1a
setosa = pd.read_csv("setosa_data.csv")
```

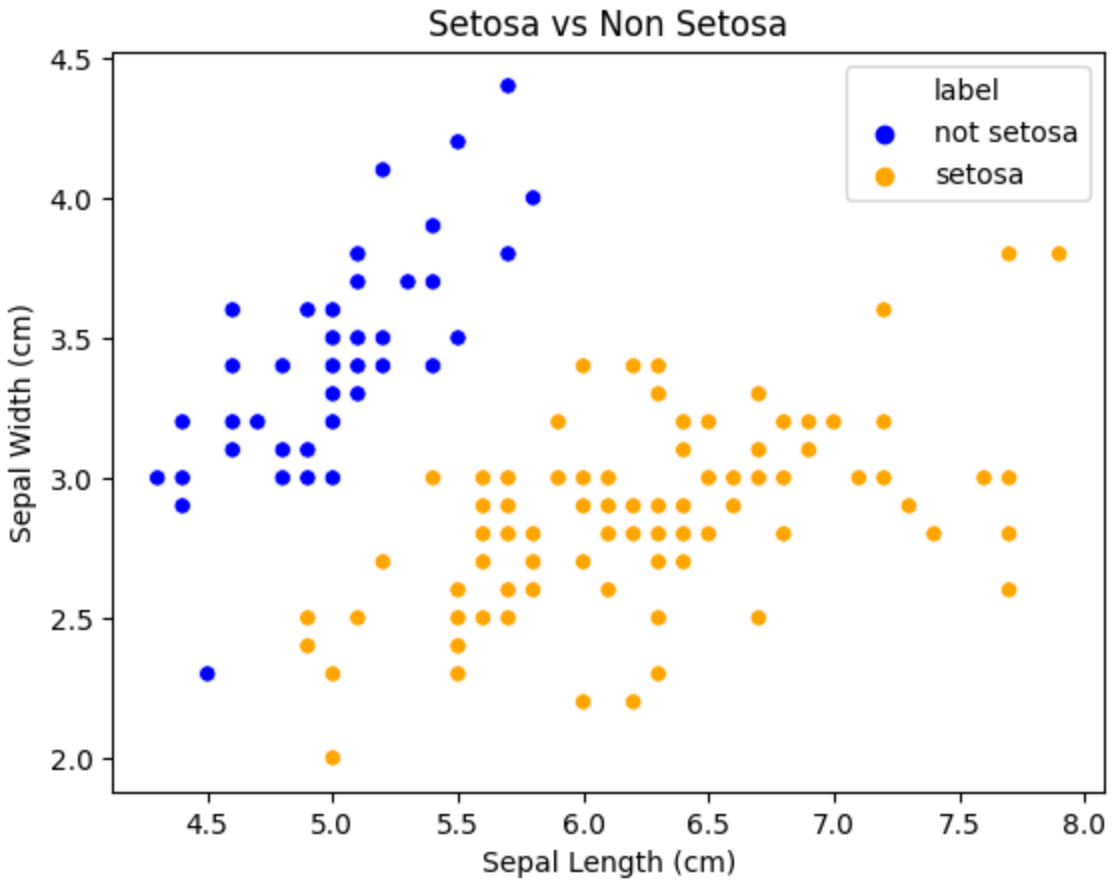
In [3]:
```
#1b
setosa.head()
```

Out[3]:

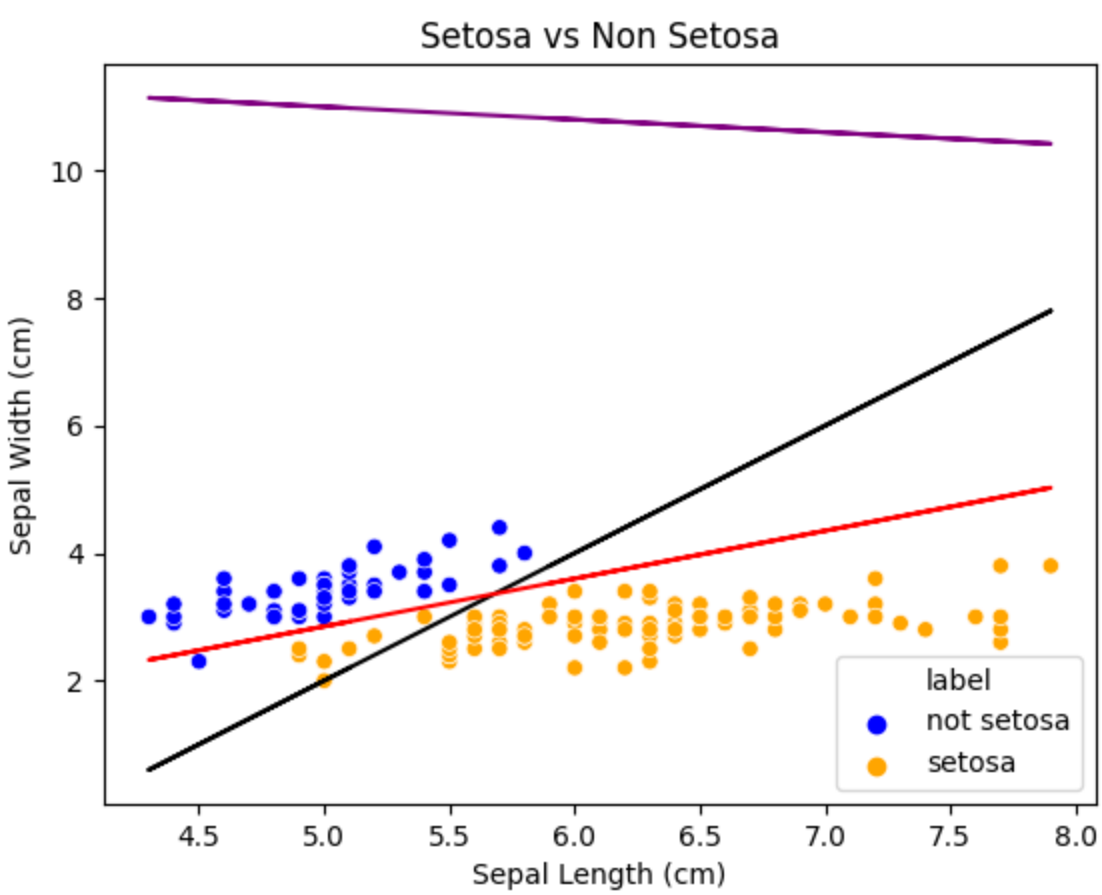
	label	sepal_length (cm)	sepal_width (cm)
0	not setosa	5.1	3.5
1	not setosa	4.9	3.0
2	not setosa	4.7	3.2
3	not setosa	4.6	3.1
4	not setosa	5.0	3.6

In [23]:
```
#1c
palette = {
    "setosa":"orange",
    "not setosa": "blue"
}

plot = sns.scatterplot(data=setosa, x="sepal_length (cm)", y="sepal_width (cm)", hue="label", pa
plot.set(title="Setosa vs Non Setosa", xlabel="Sepal Length (cm)", ylabel="Sepal Width (cm)")
```

Out[23]:
```
[Text(0.5, 1.0, 'Setosa vs Non Setosa'),
 Text(0.5, 0, 'Sepal Length (cm)'),
 Text(0, 0.5, 'Sepal Width (cm)')]
```



In [24]:
```
#1d

line_equation_black = lambda x : 2 * x - 8
line_equation_red = lambda x : 0.75 * x - .9
line_equation_purple = lambda x : -.2 * x + 12

resultsBlack = line_equation_black(setosa["sepal_length (cm)"])
resultsRed = line_equation_red(setosa["sepal_length (cm)"])
resultsPurple = line_equation_purple(setosa["sepal_length (cm)"])

plot = sns.scatterplot(data=setosa, x="sepal_length (cm)", y="sepal_width (cm)", hue="label", pa
plt.plot(setosa["sepal_length (cm)"], resultsBlack, color = "black")
plt.plot(setosa["sepal_length (cm)"], resultsRed, color = "red")
plt.plot(setosa["sepal_length (cm)"], resultsPurple, color = "purple")
plot.set(title="Setosa vs Non Setosa", xlabel="Sepal Length (cm)", ylabel="Sepal Width (cm)")

plt.show()
```



1e

Ax + By + c = 0

y = 2x – 8 --> 2x - By - 8

y = 0.75x – 0.9 --> .75x - By - 0.9

y = -0.2x + 12 --> -.2x - By + 12

In [6]:
```
#1 f-g
#constants used for parameters that won't change
features = setosa[["sepal_length (cm)", "sepal_width (cm)"]].values
true_labels = setosa["label"].values

#Different decision boundaries using equations given earlier (1e)
dec_bound_vec = np.array([2, -1.00, -8])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, features, true_labels, features
print("Accuracy score for setosa sepal length (Black line):", accuracy_score(setosa["label"], pr

dec_bound_vec = np.array([0.75, -1.00, -0.90])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, features, true_labels, features
print("Accuracy score for setosa sepal length (Red line):",accuracy_score(setosa["label"], pred_


dec_bound_vec = np.array([-0.2, -1.00, 12])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, features, true_labels, features
print("Accuracy score for setosa sepal length (Purple line):",accuracy_score(setosa["label"], pr
```

```
Accuracy score for setosa sepal length (Black line): 0.9533333333333334
Accuracy score for setosa sepal length (Red line): 0.9933333333333333
Accuracy score for setosa sepal length (Purple line): 0.6666666666666666
```

In [7]:
```
#2 a-b
versicolor = pd.read_csv("versicolor_virginica_data.csv")
versicolor.head()
```

Out[7]:

	label	sepal_length (cm)	sepal_width (cm)	petal_length (cm)	petal_width (cm)
0	versicolor	7.0	3.2	4.7	1.4
1	versicolor	6.4	3.2	4.5	1.5
2	versicolor	6.9	3.1	4.9	1.5
3	versicolor	5.5	2.3	4.0	1.3
4	versicolor	6.5	2.8	4.6	1.5

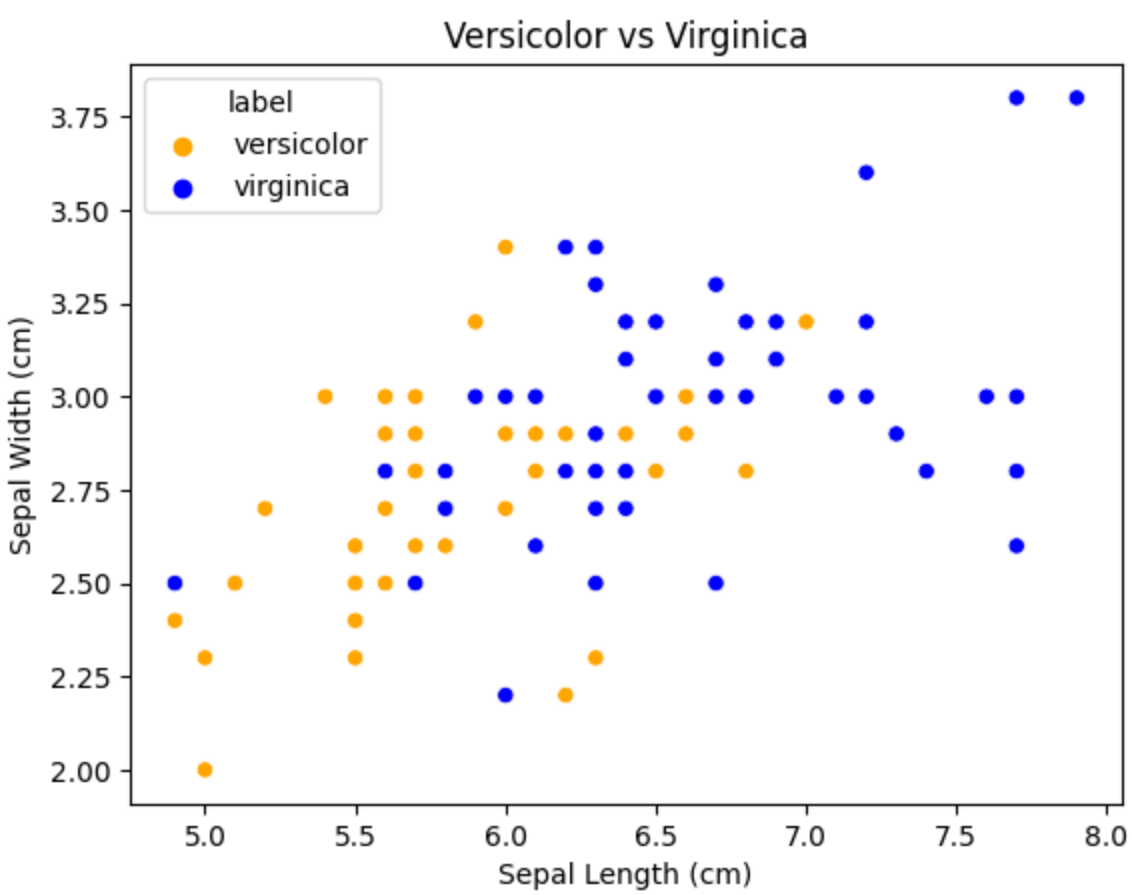
In [8]:
```
versicolor.groupby("label").count()
```

Out[8]:

label	sepal_length (cm)	sepal_width (cm)	petal_length (cm)	petal_width (cm)
versicolor	50	50	50	50
virginica	50	50	50	50

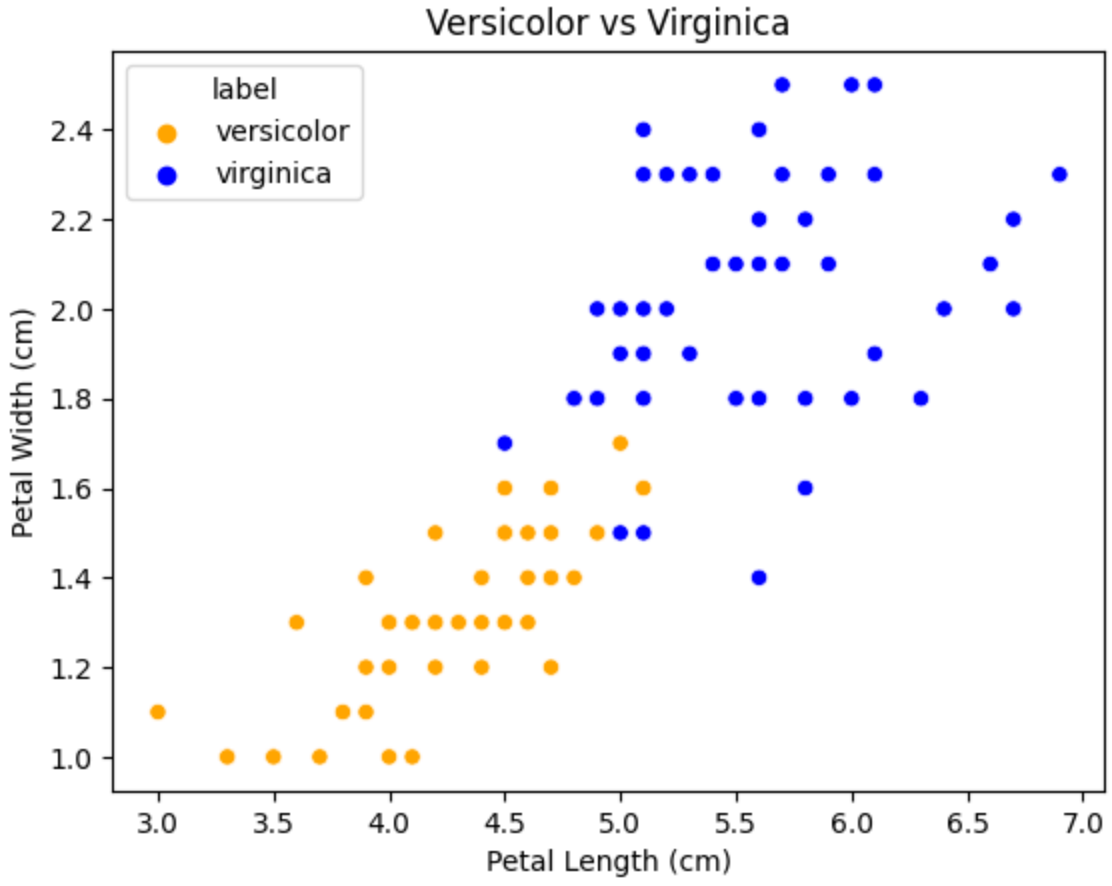
In [25]:
```
#2c
palette = {
    "versicolor":"orange",
    "virginica": "blue"
}

plot = sns.scatterplot(data=versicolor, x="sepal_length (cm)", y="sepal_width (cm)", hue="label"
plot.set(title="Versicolor vs Virginica", xlabel="Sepal Length (cm)", ylabel="Sepal Width (cm)")
```

Out[25]:
```
[Text(0.5, 1.0, 'Versicolor vs Virginica'),
 Text(0.5, 0, 'Sepal Length (cm)'),
 Text(0, 0.5, 'Sepal Width (cm)')]
```

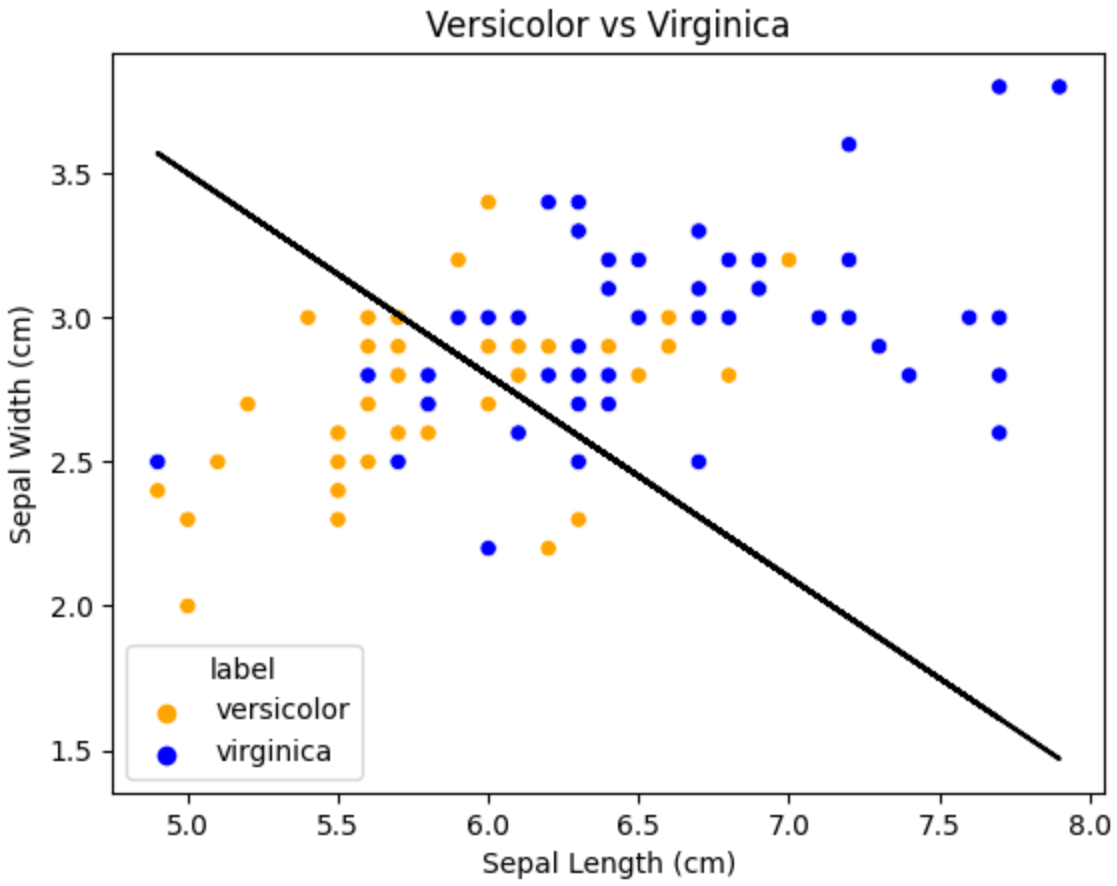


In [26]:
```
#2d
plot = sns.scatterplot(data=versicolor, x="petal_length (cm)", y="petal_width (cm)", hue="label"
plot.set(title="Versicolor vs Virginica", xlabel="Petal Length (cm)", ylabel="Petal Width (cm)")
```

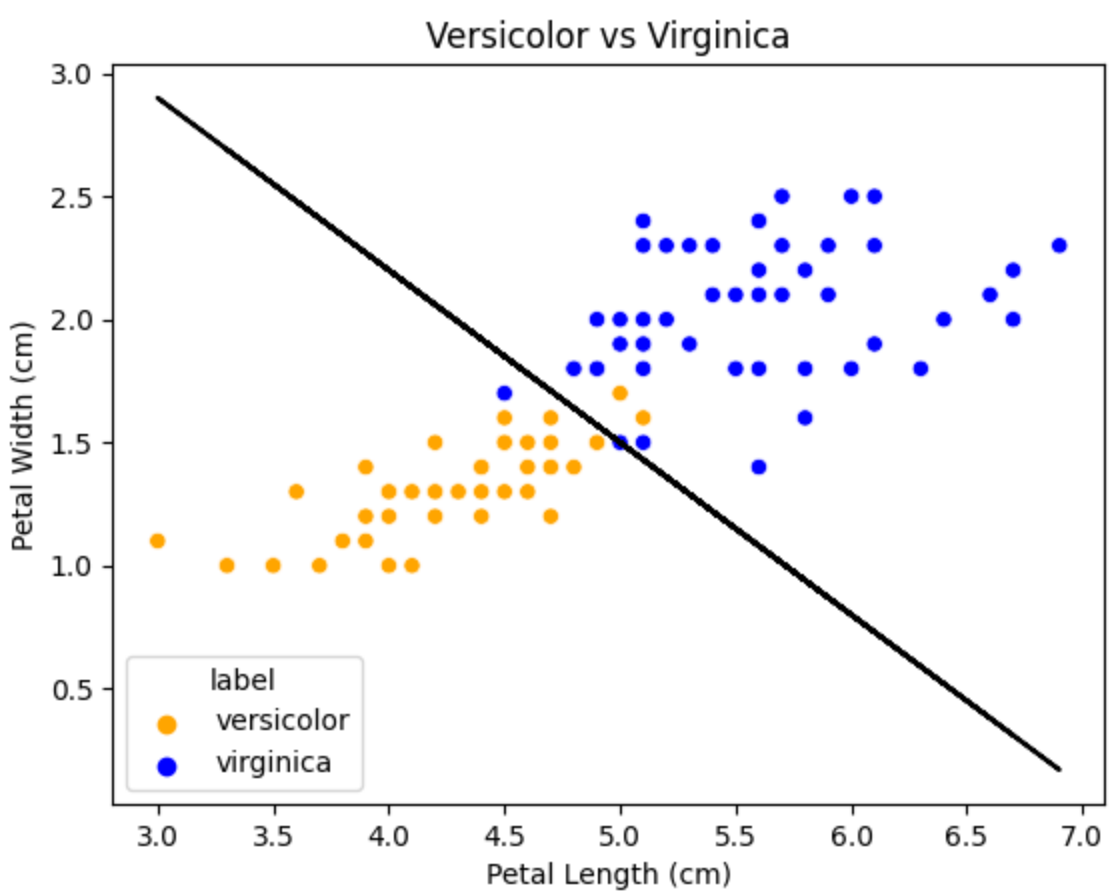
Out[26]:
```
[Text(0.5, 1.0, 'Versicolor vs Virginica'),
 Text(0.5, 0, 'Petal Length (cm)'),
 Text(0, 0.5, 'Petal Width (cm)')]
```



In [28]:
```
#2e
line_equation_sepal = lambda x : -0.7 * x + 7
resultsSepal = line_equation_sepal(versicolor["sepal_length (cm)"])

plot = sns.scatterplot(data=versicolor, x="sepal_length (cm)", y="sepal_width (cm)", hue="label"
plt.plot(versicolor["sepal_length (cm)"], resultsSepal, color = "black")
plot.set(title="Versicolor vs Virginica", xlabel="Sepal Length (cm)", ylabel="Sepal Width (cm)")
plt.show()
```



In [29]:
```
#2e continued...
line_equation_petal = lambda x :  -0.7 * x + 5
resultsPetal = line_equation_petal(versicolor["petal_length (cm)"])

plot = sns.scatterplot(data=versicolor, x="petal_length (cm)", y="petal_width (cm)", hue="label"
plt.plot(versicolor["petal_length (cm)"], resultsPetal, color = "black")
plot.set(title="Versicolor vs Virginica", xlabel="Petal Length (cm)", ylabel="Petal Width (cm)")
plt.show()
```



2f

Rewrite each decision boundary equation in a modified standard form: Ax + By + c = 0

y = -0.7x + 7

- 0 = -0.7x -By + 7

y = -0.7x + 5

- 0 = -0.7x -By + 5

In [13]:
```
#2g SEPAL

#constants used for parameters that won't change
features = versicolor[["sepal_length (cm)", "sepal_width (cm)"]].values
true_labels = versicolor["label"].values

#Different decision boundaries using equations given earlier (2f)
dec_bound_vec = np.array([-0.7, -1.0, 7])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, features, true_labels, features
print("Accuracy score for versicolor sepal: ", accuracy_score(versicolor["label"], pred_labels))
```

```
Accuracy score for versicolor sepal:  0.7
```

In [14]:
```
#2g PETAL

#constants used for parameters that won't change
features = versicolor[["petal_length (cm)", "petal_width (cm)"]].values
true_labels = versicolor["label"].values
```

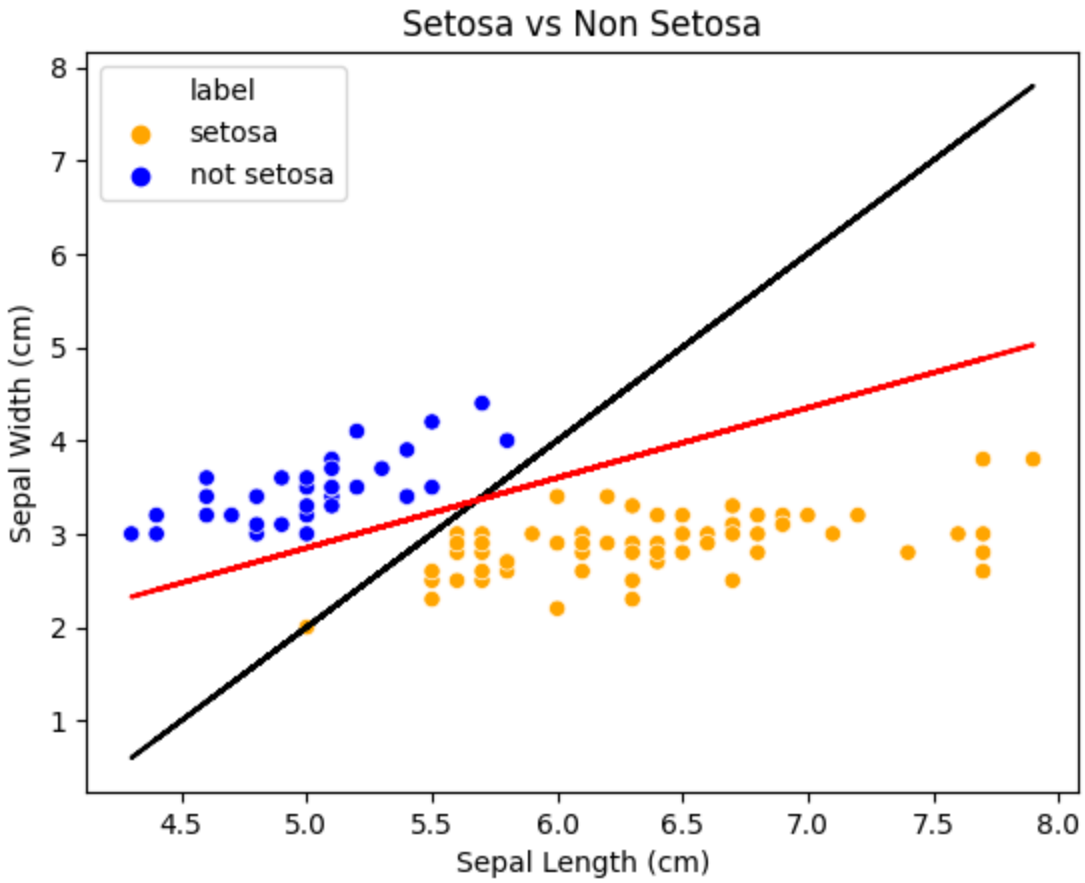
```
#Different decision boundaries using equations given earlier (2f)
dec_bound_vec = np.array([-0.7, -1.0, 5])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, features, true_labels, features
print("Accuracy score for versicolor petal: ", accuracy_score(versicolor["label"], pred_labels))
```

```
Accuracy score for versicolor petal:  0.96
```

In [31]:
```
#3a
training = pd.read_csv("setosa_training.csv")
testing = pd.read_csv("setosa_testing.csv")
validation = pd.read_csv("setosa_validation.csv")

palette = {
    "setosa":"orange",
    "not setosa": "blue"
}
```

Training Data

In [32]:
```
#base scatter plot
plot = sns.scatterplot(data=training, x="sepal_length (cm)", y="sepal_width (cm)", hue="label", p

#decision boundary   y = 2x - 8
line_equation = lambda x :  2 * x - 8
results = line_equation(training["sepal_length (cm)"])
plt.plot(training["sepal_length (cm)"], results, color = "black")

#decision boundary   y = 0.75x - .9
line_equation = lambda x : 0.75 * x - .9
results = line_equation(training["sepal_length (cm)"])
plt.plot(training["sepal_length (cm)"], results, color = "red")

plot.set(title="Setosa vs Non Setosa", xlabel="Sepal Length (cm)", ylabel="Sepal Width (cm)")

plt.show()
```



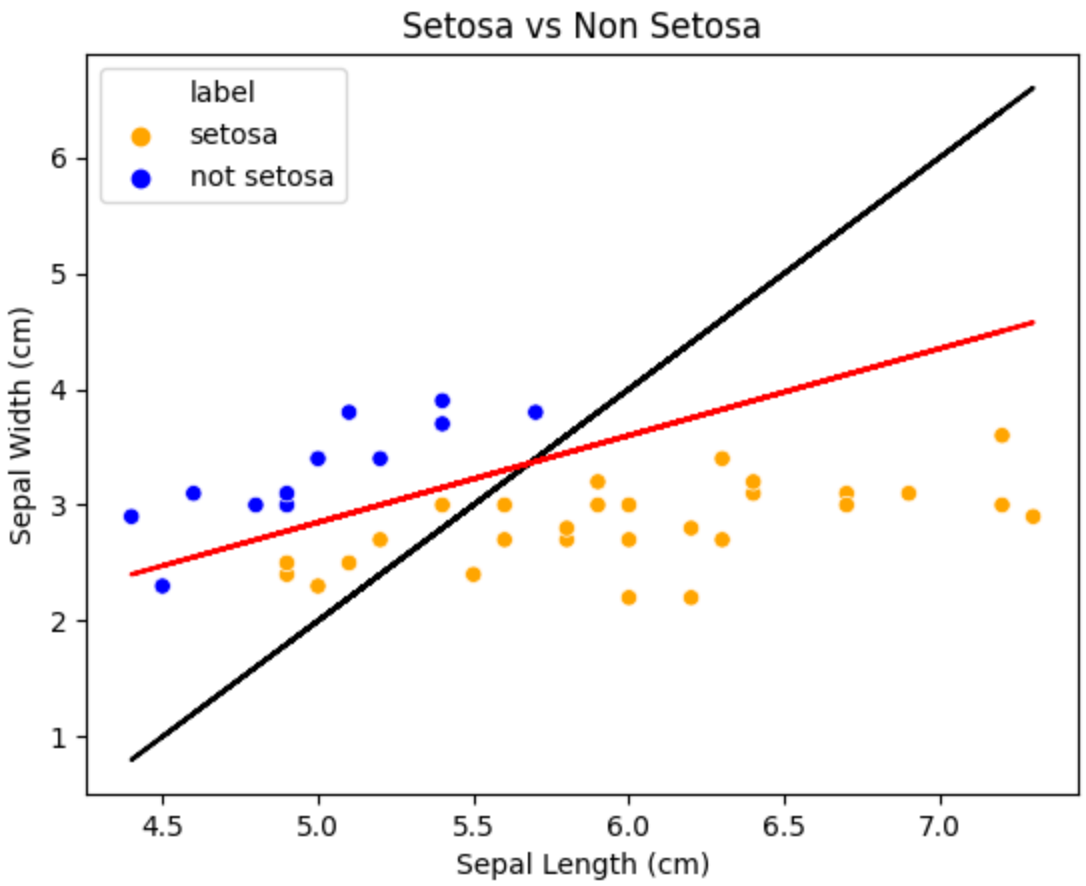
In [17]:
```
#constants used for parameters that won't change
training_points = training[["sepal_length (cm)", "sepal_width (cm)"]].values
prediction_points = training[["sepal_length (cm)", "sepal_width (cm)"]].values

true_labels = training["label"].values

#Different decision boundaries using equations given earlier (1e)
dec_bound_vec = np.array([2, -1.00, -8])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, training_points, true_labels, pr
print("Accuracy score for validation data (Black line):", accuracy_score(training["label"], pred_

dec_bound_vec = np.array([0.75, -1.00, -0.90])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, training_points, true_labels, pr
print("Accuracy score for validation data (Red line):", accuracy_score(training["label"], pred_la
```

```
Accuracy score for validation data (Black line): 0.9907407407407407
Accuracy score for validation data (Red line): 1.0
```

Testing Data

In [34]:
```
#base scatter plot
plot = sns.scatterplot(data=testing, x="sepal_length (cm)", y="sepal_width (cm)", hue="label", pa

#decision boundary   y = 2x - 8
line_equation = lambda x :  2 * x - 8
results = line_equation(testing["sepal_length (cm)"])
plt.plot(testing["sepal_length (cm)"], results, color = "black")

#decision boundary   y = 0.75x - .9
line_equation = lambda x : 0.75 * x - .9
results = line_equation(testing["sepal_length (cm)"])
plt.plot(testing["sepal_length (cm)"], results, color = "red")
```

```
plot.set(title="Setosa vs Non Setosa", xlabel="Sepal Length (cm)", ylabel="Sepal Width (cm)")

plt.show()
```



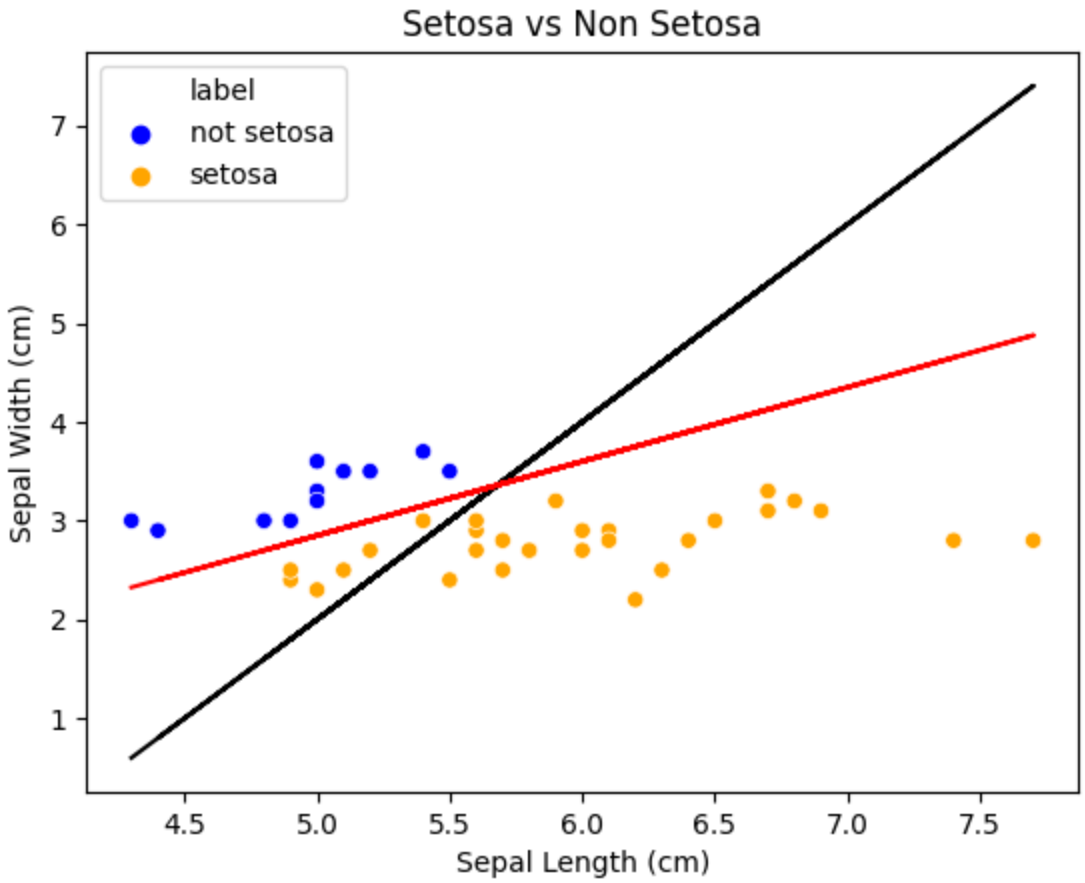
In [19]:
```
#constants used for parameters that won't change
training_points = training[["sepal_length (cm)", "sepal_width (cm)"]].values
prediction_points = testing[["sepal_length (cm)", "sepal_width (cm)"]].values

true_labels = training["label"].values

#Different decision boundaries using equations given earlier (1e)
dec_bound_vec = np.array([2, -1.00, -8])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, training_points, true_labels, pr
print("Accuracy score for validation data (Black line):", accuracy_score(testing["label"], pred_

dec_bound_vec = np.array([0.75, -1.00, -0.90])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, training_points, true_labels, pr
print("Accuracy score for validation data (Red line):", accuracy_score(testing["label"], pred_lab
```

```
Accuracy score for validation data (Black line): 0.8571428571428571
Accuracy score for validation data (Red line): 0.9761904761904762
```

Validation Data

In [35]:
```
#base scatter plot
plot = sns.scatterplot(data=validation, x="sepal_length (cm)", y="sepal_width (cm)", hue="label"

#decision boundary   y = 2x - 8
line_equation = lambda x :  2 * x - 8
results = line_equation(validation["sepal_length (cm)"])
plt.plot(validation["sepal_length (cm)"], results, color = "black")
```

```
#decision boundary   y = 0.75x - .9
line_equation = lambda x : 0.75 * x - .9
results = line_equation(validation["sepal_length (cm)"])
plt.plot(validation["sepal_length (cm)"], results, color = "red")
plot.set(title="Setosa vs Non Setosa", xlabel="Sepal Length (cm)", ylabel="Sepal Width (cm)")

plt.show()
```



In [21]:
```
#3b

#constants used for parameters that won't change
training_points = training[["sepal_length (cm)", "sepal_width (cm)"]].values
prediction_points = validation[["sepal_length (cm)", "sepal_width (cm)"]].values

true_labels = training["label"].values

#Different decision boundaries using equations given earlier (1e)
dec_bound_vec = np.array([2, -1.00, -8])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, training_points, true_labels, pr
print("Accuracy score for validation data (Black line):", accuracy_score(validation["label"], pre

dec_bound_vec = np.array([0.75, -1.00, -0.90])
pred_labels = linear_decision_boundary_classifier(dec_bound_vec, training_points, true_labels, pr
print("Accuracy score for validation data (Red line):", accuracy_score(validation["label"], pred_
```

```
Accuracy score for validation data (Black line): 0.8636363636363636
Accuracy score for validation data (Red line): 1.0
```