worksheet-12

October 25, 2023

1 Worksheet 12

Name: Ivanna Morales UID: U69469924

1.0.1 Topics

- Introduction to Classification
- K Nearest Neighbors

1.0.2 Introduction to Classification

a) For the following examples, say whether they are or aren't an example of classification.

1. Predicting whether a student will be offered a job after graduating given their GPA.
2. Predicting how long it will take (in number of months) for a student to be offered a job after graduating, given their GPA.
3. Predicting the number of stars (1-5) a person will assign in their yelp review given the description they wrote in the review.
4. Predicting the number of births occuring in a specified minute.

- Yes, there is a binary output based on students GPA
- No, this is a regression problem
- Yes, we are classifying into different classes
- No, this is a regression problem

b) Given a dataset, how would you set things up such that you can both learn a model and get an idea of how this model might perform on data it has never seen?

- split the dataset for testing, training and validation
- handle outliers
- create features
- apply the correct model
- evaluate its performance on the validation set

c) In your own words, briefly explain:

- underfitting
- overfitting

and what signs to look out for for each.

- underfiting is when a model is too simplistic or does not have enough parameters to represent the complex patterns

- overfitting is when a model is too complex such that it detects small details. this makes it difficut to apply to other datasets.

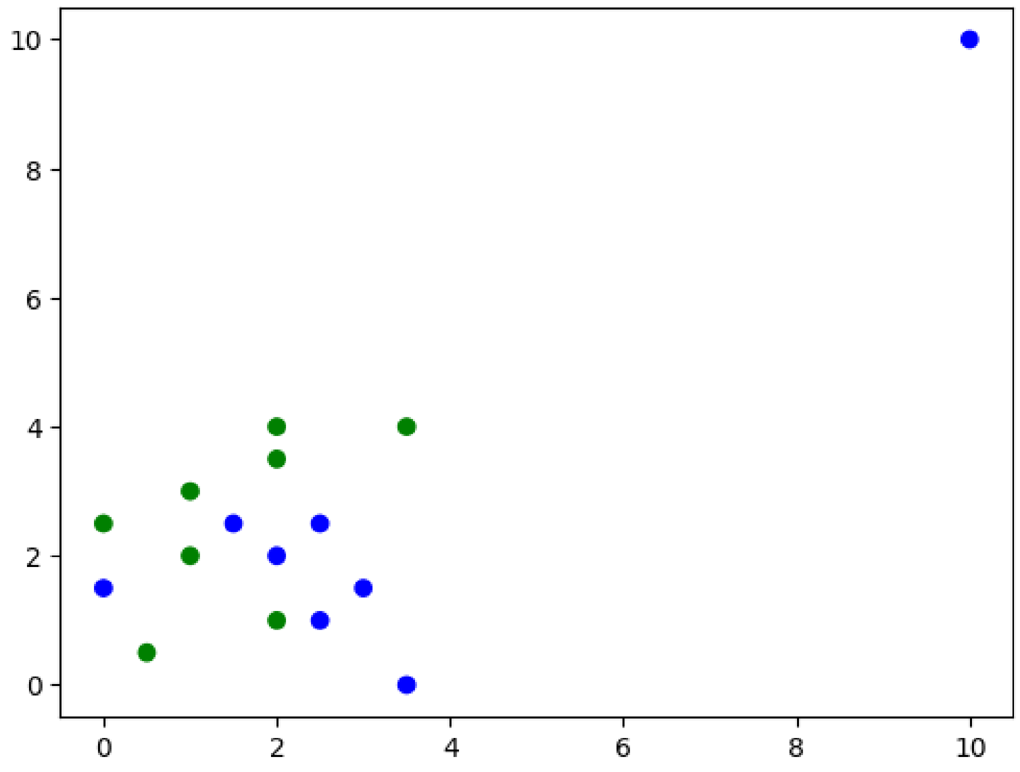
1.0.3 K Nearest Neighbors

[5]:
```
import numpy as np
import matplotlib.pyplot as plt

data = {
    "Attribute A" : [3.5, 0, 1, 2.5, 2, 1.5, 2, 3.5, 1, 3, 2, 2, 2.5, 0.5, 0.,␣
 ↪10],
    "Attribute B" : [4, 1.5, 2, 1, 3.5, 2.5, 1, 0, 3, 1.5, 4, 2, 2.5, 0.5, 2.5,␣
 ↪10],
    "Class" : [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0],
}
```

a) Plot the data in a 2D plot coloring each scatter point one of two colors depending on its corresponding class.

[7]:
```
colors = np.array([x for x in 'bgrcmyk'])
plt.scatter(data["Attribute A"], data["Attribute B"],␣
 ↪color=colors[data["Class"]].tolist())
plt.show()
```



Outliers are points that lie far from the rest of the data. They are not necessarily invalid points however. Imagine sampling from a Normal Distribution with mean 10 and variance 1. You would expect most points you sample to be in the range [7, 13] but it's entirely possible to see 20 which, on average, should be very far from the rest of the points in the sample (unless we're VERY (un)lucky). These outliers can inhibit our ability to learn general patterns in the data since they are not representative of likely outcomes. They can still be useful in of themselves and can be analyzed in great depth depending on the problem at hand.

b) Are there any points in the dataset that could be outliers? If so, please remove them from the dataset.

(10,10)

[8]:
```python
for x in data:
    data[x].pop()
    print(data[x])
```

```
[3.5, 0, 1, 2.5, 2, 1.5, 2, 3.5, 1, 3, 2, 2, 2.5, 0.5, 0.0]
[4, 1.5, 2, 1, 3.5, 2.5, 1, 0, 3, 1.5, 4, 2, 2.5, 0.5, 2.5]
[1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1]
```

Noise points are points that could be considered invalid under the general trend in the data. These could be the result of actual errors in the data or randomness that we could attribute to oversimplification (for example if missing some information / feature about each point). Considering noise points in our model can often lead to overfitting.

c) Are there any points in the dataset that could be noise points?

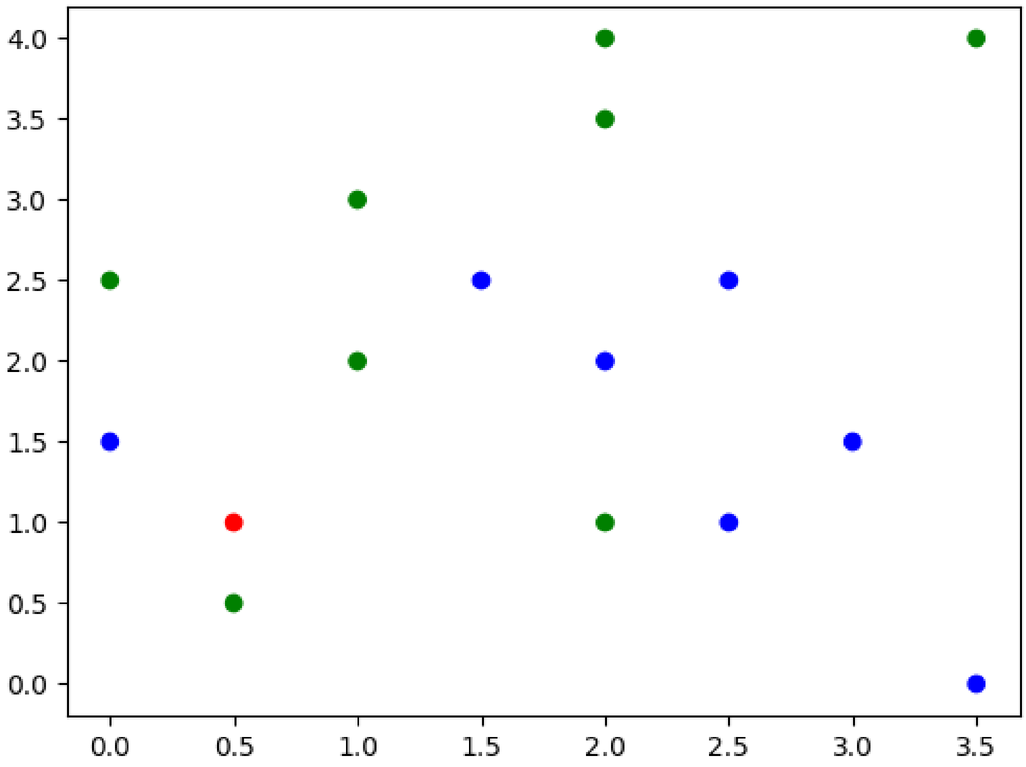
(0, 1.5) & (3.5, 0)

For the following point

A	B
0.5	1

d) Plot it in a different color along with the rest of the points in the dataset.

[9]:
```python
plt.scatter(data["Attribute A"] + [0.5], data["Attribute B"] + [1],␣
  ↪color=colors[data["Class"] + [2]].tolist())
plt.show()
```



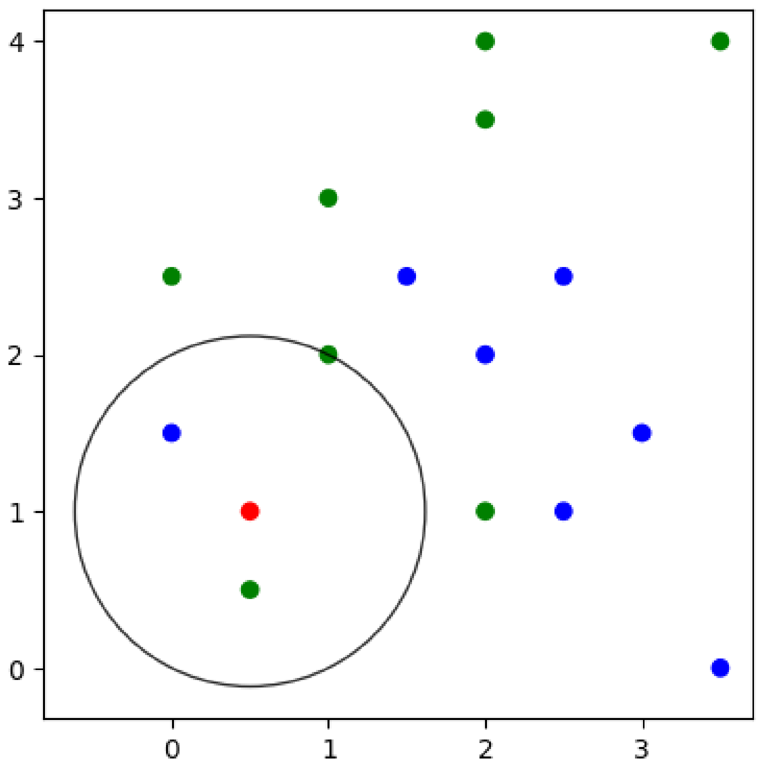
e) Write a function to compute the Euclidean distance from it to all points in the dataset and pick the 3 closest points to it. In a scatter plot, draw a circle centered around the point with radius the distance of the farthest of the three points.

[10]:
```
def n_closest_to(example, n):
    distances = []
    for i in range(len(data["Attribute A"])):
        distances.append(np.linalg.norm(np.array([data["Attribute A"][i],␣
 ↪data["Attribute B"][i]]) - np.array(location)))
    return np.argsort(distances)[:n]

location = ( 0.5 , 1 )
radius = np.linalg.norm(np.array([data["Attribute A"][n_closest_to(location,␣
 ↪3)[-1]], data["Attribute B"][n_closest_to(location, 3)[-1]]]) - np.
 ↪array(location))

_, axes = plt.subplots()
axes.scatter(data["Attribute A"] + [0.5], data["Attribute B"] + [1],␣
 ↪color=colors[data["Class"] + [2]].tolist())
cir = plt.Circle(location, radius, fill = False, alpha=0.8)
axes.add_patch(cir)
axes.set_aspect('equal') # necessary so that the circle is not oval
```

```
plt.show()
```



f) Write a function that takes the three points returned by your function in e) and returns the class that the majority of points have (break ties with a deterministic default class of your choosing). Print the class assigned to this new point by your function.

[29]:
```
def majority(points):
    one = 0
    zero = 0
    for point in points:
        if data["Class"][point]==1:
            one+=1
        else:
            zero+=1
    if(one>zero):
        return 1
    else:
        return 0
print("Majority Class: ", majority(n_closest_to(location, 3)))
```

```
Majority Class:  1
```

g) Re-using the functions from e) and f), you should be able to assign a class to any new point.

In this exercise we will implement Leave-one-out cross validiation in order to evaluate the performance of our model.

For each point in the dataset:

- consider that point as your test set and the rest of the data as your training set
- classify that point using the training set
- keep track of whether you were correct with the use of a counter

Once you've iterated through the entire dataset, divide the counter by the number of points in the dataset to report an overall testing accuracy.

[36]:
```
count = 0
for i in range(len(data["Attribute A"])):
    actual_class = data["Class"][i]
    training_set = data["Class"][:i] + data["Class"][i+1:]
    prediction = majority(n_closest_to((data["Attribute A"][i], data["Attribute␣
 ↪B"][i]), 3))

    if prediction == actual_class:
        count += 1

accuracy = count/len(data["Attribute A"])
print("overall accuracy = ", accuracy)
```

```
overall accuracy =  0.5333333333333333
```