

# HW 11 Autograder

● Graded

Student

Sangwon Ji

Total Points

79 / 80 pts

Autograder Score

74.0 / 80.0

Passed Tests

Public Tests

Question 2

Early Submission Bonus

5 / 0 pts

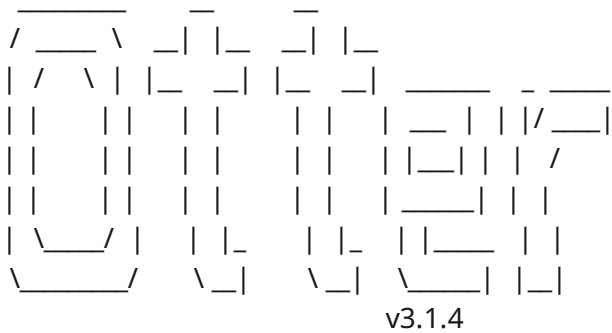
✓ + 5 pts Early Submission Bonus

+ 0 pts No bonus

Autograder Results

Autograder Output

Matplotlib is building the font cache; this may take a moment.  
/root/miniconda3/envs/otter-env/lib/python3.7/site-packages/datascience/maps.py:800: FutureWarning: T  
warnings.warn("The 'radius' argument is deprecated. Please use 'area' instead.", FutureWarning)



<IPython.core.display.HTML object>  
Training set: 75 examples  
Test set: 25 examples  
<IPython.core.display.HTML object>  
<IPython.core.display.HTML object>

----- GRADING SUMMARY -----

Error encountered while trying to verify scores with log:  
'TestCaseResult' object has no attribute 'hidden'

Successfully uploaded submissions for: sangwon@berkeley.edu

Total Score: 74.000 / 80.000 (92.500%)

	name	score	max_score
0	Public Tests	NaN	NaN
1	q1_1	10.0	10.0
2	q1_2	10.0	10.0
3	q1_3	10.0	10.0
4	q1_4	10.0	10.0
5	q1_5	4.0	10.0
6	q1_6	10.0	10.0
7	q1_9_1	10.0	10.0
8	q1_9_2	10.0	10.0

## Public Tests

q1\_1 results: All test cases passed!

q1\_2 results: All test cases passed!

q1\_3 results: All test cases passed!

q1\_4 results: All test cases passed!

q1\_5 results: All test cases passed!

q1\_6 results: All test cases passed!

q1\_9\_1 results: All test cases passed!

q1\_9\_2 results: All test cases passed!

## Submitted Files

In [1]:

```
# Initialize Otter
import otter
grader = otter.Notebook("hw11.ipynb")
```

## Homework 11: Classification

### Helpful Resource:

- [Python Reference](#): Cheat sheet of helpful array & table methods used in Data 8!

### Recommended Reading:

- [Classification](#)

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to setup the notebook by importing some helpful libraries. Each time you start your server, you will need to execute this cell again.

For all problems that you must write explanations and sentences for, you **must** provide your answer in the designated space. **Moreover, throughout this homework, please be sure to not re-assign variables throughout the notebook!** For example, if you use `max_temperature` in your answer to one question, do not reassign it later on. Otherwise, you will fail tests that you thought you were passing previously!

### Deadline:

This assignment is due **Friday, 8/5 at 11:59pm PT**. Turn it in by Thursday, 8/4 at 11:59pm PT for 5 extra credit points. Late work will not be accepted as per the [policies](#) page.

**Note: This homework has hidden tests on it. That means even though tests may say 100% passed, it doesn't mean your final grade will be 100%. We will be running more tests for correctness once everyone turns in the homework.**

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged. Refer to the policies page to learn more about how to learn cooperatively.

You should start early so that you have time to get help if you're stuck. The OH schedule appears on <http://data8.org/su22/office-hours.html>.

In [2]:

```
# Don't change this cell; just run it.

import numpy as np
from datascience import *
import d8error

# These lines do some fancy plotting magic
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore')
from datetime import datetime
```

## 1. Bay Area School Coordinates with Classification

Welcome to Homework 11! This homework is about k-Nearest Neighbors classification (k-NN). Since this topic is covered in depth in Project 2, the purpose of this homework is to reinforce the basics of this method. You can and should reuse a lot of code that you wrote for Project 2 for this homework, or use code from this homework on Project 2!

### Our Dearest Neighbors

Carol is trying to classify students as either attendees of UC Berkeley or as attendees of Stanford University. To classify the students, Carol has access to the coordinates of the location they live during the school year. First, load in the `coordinates` table.

In [3]:

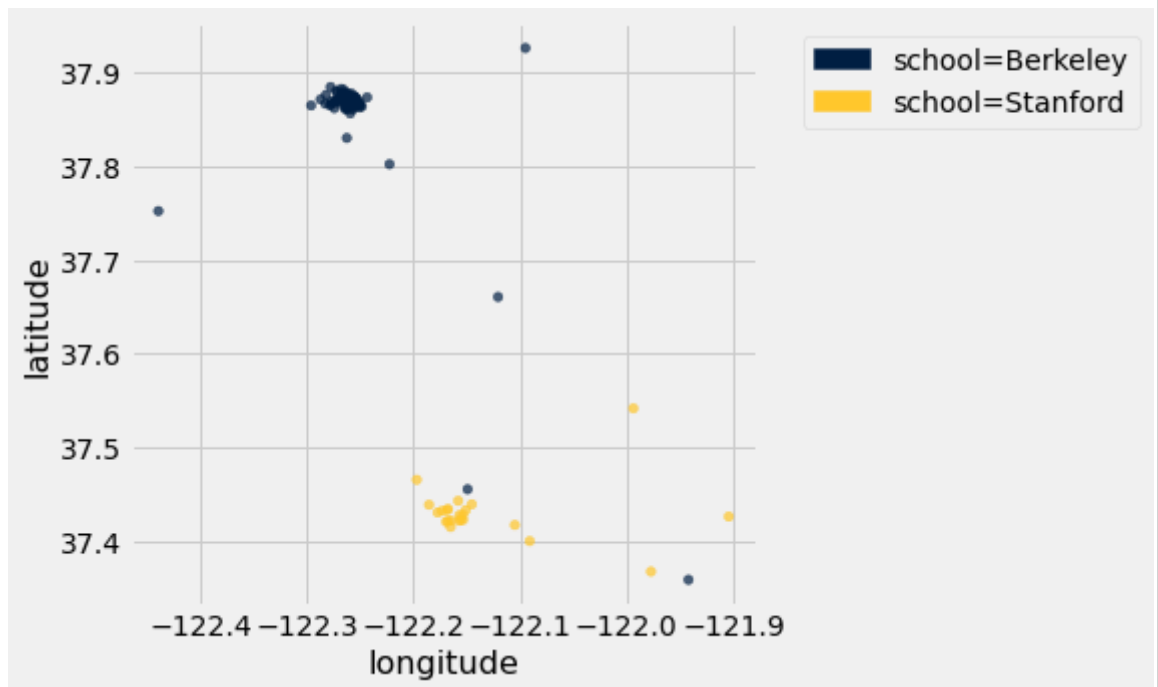
```
# Just run this cell!
coordinates = Table.read_table('coordinates.csv')
coordinates.show(5)
```

<IPython.core.display.HTML object>

As usual, let's investigate our data visually before performing any kind of numerical analysis.

In [4]:

```
# Just run this cell!
coordinates.scatter("longitude", "latitude", group="school")
```



The locations of the points on this scatter plot might be familiar - run the following cell to see what they correspond to.

In [5]:

```
# Just run this cell!
colors = {"Berkeley": "blue", "Stanford": "red"}
t = Table().with_columns("lat", coordinates.column(0),
                        "lon", coordinates.column(1),
                        "color", coordinates.apply(colors.get, 2)
                        )
Circle.map_table(t, radius=5, fill_opacity=1)
```

Out [5]: <datascience.maps.Map at 0x7f3ab60fe8e0>

**Question 1.1.** Let's begin implementing the k-Nearest Neighbors algorithm. Define the `distance` function, which takes in two arguments: an array of numerical features (`arr1`), and a different array of numerical features (`arr2`). The function should return the [Euclidean distance](#) between the two arrays. Euclidean distance is often referred to as the straight-line distance formula that you may have learned previously. **(10 points)**

In [6]:

```
def distance(arr1, arr2):
    return np.sqrt(sum((arr2 - arr1) ** 2))

# Don't change/delete the code below in this cell
```

```
distance_example = distance(make_array(1, 2, 3), make_array(4, 5, 6))
distance_example
```

Out [6]: 5.196152422706632

```
In [7]: grader.check("q1_1")
```

Out [7]: q1\_1 results: All test cases passed!

## Splitting the Dataset

We'll do two different kinds of things with the `coordinates` dataset:

1. We'll build a classifier using `coordinates` for which we know the associated label; this will teach it to recognize labels of similar coordinate values. This process is known as *training*.
2. We'll evaluate or *test* the accuracy of the classifier we build on data we haven't seen before.

As discussed in [Section 17.2](#), we want to use separate datasets for training and testing. As such, we split up our one dataset into two.

**Question 1.2.** Next, let's split our dataset into a training set and a test set. Since `coordinates` has 100 rows, let's create a training set with the first 75 rows and a test set with the remaining 25 rows. Remember that assignment to each group should be random, so we should shuffle the table first. **(10 points)**

*Hint:* As a first step we can shuffle all the rows, then use the `tbl.take` function to split up the rows for each table.

```
In [8]: shuffled_table = coordinates.sample(100, with_replacement=False)
train = shuffled_table.take(np.arange(75))
test = shuffled_table.take(np.arange(75, 100))

print("Training set:\t", train.num_rows, "examples")
print("Test set:\t", test.num_rows, "examples")
train.show(5), test.show(5);
```

Training set: 75 examples

Test set: 25 examples

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

In [9]: `grader.check("q1_2")`

Out [9]: q1\_2 results: All test cases passed!

**Question 1.3.** Assign `features` to an array of column names (strings) of the features from the `coordinates` table. **(10 points)**

*Hint:* Which of the column names in the `coordinates` table are the features, and which of the column names correspond to the class we're trying to predict?

*Hint:* No need to modify any tables, just manually create an array of the feature names!

In [10]: `features = make_array("latitude", "longitude")`  
`features`

Out [10]: `array(['latitude', 'longitude'], dtype='<U9')`

In [11]: `grader.check("q1_3")`

Out [11]: q1\_3 results: All test cases passed!

**Question 1.4.** Now define the `classify` function. This function should take in a `test_row` from a table like `test` and classify in using the k-Nearest Neighbors based on the correct `features` and the data in `train`. A refresher on k-Nearest Neighbors can be found [here](#). **(10 points)**

*Hint 1:* The `distance` function we defined earlier takes in arrays as input, so use the `row_to_array` function we defined for you to convert rows to arrays of features.

*Hint 2:* The skeleton code we provided iterates through each row in the training set.

In [12]: 

```
def row_to_array(row, features):  
    """Converts a row to an array of its features."""  
    arr = make_array()  
    for feature in features:  
        arr = np.append(arr, row.item(feature))
```



```

return arr

def classify(test_row, k, train):
    test_row_features_array = row_to_array(test_row, features)
    distances = make_array()
    for train_row in train.rows:
        train_row_features_array = row_to_array(train_row, features)
        row_distance = distance(test_row_features_array, train_row_features_array)
        distances = np.append(distances, row_distance)
    train_with_distances = train.with_column("distance", distances)
    nearest_neighbors = train_with_distances.sort("distance").take(np.arange(0,k))
    most_common_label =
make_array(np.count_nonzero(nearest_neighbors.column("school") ==
"Berkeley"), np.count_nonzero(nearest_neighbors.column("school") ==
"Standord"))
    if most_common_label.item(0) > most_common_label.item(1):
        return "Berkeley"
    else:
        return "Stanford"

# Don't modify/delete the code below
first_test = classify(test.row(0), 5, train)
first_test

```

Out [12]: 'Berkeley'

In [13]: grader.check("q1\_4")

Out [13]: q1\_4 results: All test cases passed!

**Question 1.5.** Define the function `three_classify` that takes a `row` from `test` as an argument and classifies the row based on using 3-Nearest Neighbors. Use this function to find the `accuracy` of a 3-NN classifier on the `test` set. `accuracy` should be a proportion (not a percentage) of the schools that were correctly predicted. **(10 points)**

*Hint:* You should be using a function you just created!

*Note:* Usually before using a classifier on a test set, we'd classify first on a "validation" set, which we then can modify our training set again if need be, before actually testing on the test set. You don't need to do that for this question, but you will learn about this more in Data 100.

In [14]: 

```
def three_classify(row):
    return classify(row, 3, train)
```

```
test_with_prediction = test.with_column("prediction", test.apply(three_classify))
labels_correct =
np.count_nonzero(test_with_prediction.column("prediction")==test_with_prediction.co

accuracy = labels_correct/test.num_rows
accuracy
```

Out [14]: 1.0

```
In [15]: grader.check("q1_5")
```

Out [15]: q1\_5 results: All test cases passed!

**Question 1.6.** There are 77 rows of Berkeley students and 23 rows of Stanford students in the `coordinates` table. If we used the entire `coordinates` table as the training set, what is the smallest value of  $k$  that would ensure that a  $k$ -Nearest Neighbor classifier would *always* predict Berkeley as the class? Assign the value to `k`. **(10 points)**

```
In [16]: k = 47
k
```

Out [16]: 47

```
In [18]: grader.check("q1_6")
```

Out [18]: q1\_6 results: All test cases passed!

**Question 1.7.** Why do we divide our data into a training and test set? What is the point of a test set, and why do we only want to use the test set once? Explain your answer in 3 sentences or less. **(10 points)**

*Hint:* Check out this [section](#) in the textbook.

The reason we divide our data into a training and test set is because we want our algorithm to be able to generalize, and having the test set is testing whether our algorithm can do that. The role and the point of a test set, is a way to understand how well our classifier would perform in a real world scenario with the data it has not seen before. We only want to use the test set once since it represent the data that is out of sample, that provides the best possible estimate, that can be used obejctively.

**Question 1.8.** Why do we choose `k` to be an odd number in k-NN? Explain. (10 points)

If we choose an even number, there is chance that there might be ties when evaluating the data. We will not be able to classify if the results come out tie. For example having 6 for k, we might get the same results for 3 and 3 for both stanford and berkeley.

### Question 1.9.0. Setup

Thomas has devised a scheme for splitting up the test and training set. For each row from `coordinates`:

- Rows for Stanford students have a 50% chance of being placed in the training set and 50% chance of being placed in the test set.
- Rows for Berkeley students have a 80% chance of being placed in the training set and 20% chance of being placed in the test set.

*Hint 1:* Remember that there are 77 Berkeley students and 23 Stanford students in `coordinates`.

*Hint 2:* Thomas' last name is Bayes. (So [18.1](#) from the textbook may be helpful here!)

*Hint 3:* The following tree diagram may be helpful in Questions 1.9.1 and 1.9.2!

**Question 1.9.1.** Given that a row is in the test set, what is the probability that it corresponds to a Stanford student? Assign that probability to `prob_furd`. (10 points)

```
In [19]: prob_furd = (23 * 0.5) / ((77 * 0.2) + (23 * 0.5))
         prob_furd
```

```
Out [19]: 0.42750929368029744
```

```
In [20]: grader.check("q1_9_1")
```

```
Out [20]: q1_9_1 results: All test cases passed!
```

**Question 1.9.2.** Given that a row is Stanford, what is the probability that the student is in the test set? Assign that probability to `prob_test`. (10 points)

```
In [21]: prob_test = (0.23 * 0.5)/ 0.23
        prob_test
```

Out [21]: 0.5

```
In [22]: grader.check("q1_9_2")
```

Out [22]: q1\_9\_2 results: All test cases passed!

## (OPTIONAL, NOT IN SCOPE): k-NN for Non-Binary Classification

**THIS IS NOT IN SCOPE.** There are no autograder tests for this or code for you to write. It just relies on the function `classify` in Question 1.4. Go ahead and read through this section and run the following cells!

In this class, we have taught you how to use the k-NN algorithm to classify data as one of two classes. However, much of the data you will encounter in the real world will not fall nicely into one of two categories.

**How can we classify data with non-binary classes?** It turns out we can still use k-NN! That is, we find the distance between a point and all its neighbors, find the nearest neighbors, and take a majority vote among the neighbors to determine this point's class.

The only difference is that now the neighboring points have more than two possible classes. This does introduce difficulty because now we have no way of guaranteeing that we will not encounter ties between classes. In the case that we do encounter a tie, we can just arbitrarily choose one of the classes.

In fact, you don't even have to modify the code you wrote before at all to enable multi-class classification!

Let's add some more data to our train table, this time for another class of students, students at San Jose Community College (SJCC).

```
In [23]: coordinates_multi = coordinates.with_rows([
        [37.304346, -121.915401, "SJCC"],
        [37.316275, -121.913879, "SJCC"],
```

```
[37.409435, -121.951379, "SJCC"],  
[37.349387, -121.960771, "SJCC"],  
[37.329083, -121.928479, "SJCC"],  
[37.313017, -121.866730, "SJCC"],  
[37.346525, -121.894767, "SJCC"],  
[37.364157, -121.955717, "SJCC"],  
[37.383362, -121.925776, "SJCC"],  
[37.329545, -121.880639, "SJCC"]  
])
```

In [24]: `classify(coordinates_multi.row(0), 5, coordinates_multi)`

Out [24]: 'Berkeley'

In [25]: `classify(coordinates_multi.row(91), 5, coordinates_multi)`

Out [25]: 'Stanford'

In [26]: `classify(coordinates_multi.row(105), 5, coordinates_multi)`

Out [26]: 'Stanford'

Our classifier can classify rows as belonging to one of three classes!

Classification is one of the most important fields in statistics, data science, and machine learning. There are thousands of different classification algorithms and modifications of algorithms! There are many that you'll learn if you continue down the path of becoming a data scientist!

You're all done with Homework 11!

**Important submission steps:** 1. Run the tests and verify that they all pass. 2. Choose **Save Notebook** from the **File** menu, then **run the final cell**. 3. Click the link to download the zip file. 4. Go to [Gradescope](#) and submit the zip file to the corresponding assignment. The name of this assignment is "HW 11 Autograder".

**It is your responsibility to make sure your work is saved before running the last cell.**

---

To double-check your work, the cell below will rerun all of the autograder tests.

In [27]: `grader.check_all()`

Out [27]: q1\_1 results: All test cases passed!  
q1\_2 results: All test cases passed!  
q1\_3 results: All test cases passed!  
q1\_4 results: All test cases passed!  
q1\_5 results: All test cases passed!  
q1\_6 results: All test cases passed!  
q1\_9\_1 results: All test cases passed!  
q1\_9\_2 results: All test cases passed!

## Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**


In [ ]: `# Save your notebook first, then run this cell to export your submission.  
grader.export(pdf=False)`

▼ .OTTER\_LOG

 Download

1 Binary file hidden. You can download it using the button above.

▼ \_\_zip\_filename\_\_

 Download

1 [hw11\\_2022\\_08\\_04T17\\_02\\_33\\_064509.zip](#)