Discussion 5

DSC 20, Spring 2023

map/filter/lambda, Machine Learning

Since we just had the first midterm, we'll have an interlude of content

Midterm 1 Grades

Don't ask us about this! The grades will be out when they're out!

Agenda

- Map/Filter/Lambda
- Machine Learning Preface
- Vocabulary
- Classifiers vs Regressors
- KNN

Lambda Functions

- known as anonymous functions (their functions are so simple, they don't need a name)
- syntax: lambda (input): (some operation)
- ex: lambda x: x + 2 (adds 2 to every element encountered)
- within the scope of this course, lambda is used in conjunction with map and filter

note: lambda functions can't include statements (ex. return, assert)

Map

Syntax: map(function, iterable)

- Map allows you to apply a function to all elements to an iterable input
- very common to use a lambda function as the function to apply
- returns an iterator through the iterable object, applying the function as it traverses

In [3]:
```
test = ['dsc20', 'midterm', 'one']
sample_map = map(lambda x: x[:3], test)
list(sample_map)
```

Out[3]:
```
['dsc', 'mid', 'one']
```

Filter

Syntax: filter(function, iterable)

- Filter takes in a function that returns a boolean and only keeps elements that satisfy the function (i.e. return True).
- Very common to use a lambda function as the function to apply, but keep in mind the function **must return a boolean**.
- Returns an iterator through the iterable object that only yields values that pass the function.

note: filters are a unique subset of maps. You **can** theoretically write filters as maps, but that's unnecessary complication.

In [4]:
```
test = ['dsc20', 'midterm', 'one']
sample_filter = filter(lambda x: any([c in 'aeiou' for c in x]), test)
list(sample_filter)
```

Out[4]:
```
['midterm', 'one']
```

Checkpoint

Given a list of integers, use only map/filter/lambda to create a new list containing only even numbers from the original list, then multiply the remaining elements by 3.

In [6]:

```
def even_by_3(int_list):
    """
    Function that utilizes only map/filter/lambda to
    remove odd numbers from a given list and multiply
    the remaining elements by 3.

    Args:
        int_list (list): list of ints to be considered
    Returns:
        a list of even numbers from the original list,
        multiplied by 3.

    >>> lst = [1,2,3,4,5,6]
    >>> even_by_3(lst)
    [6,12,18]
    """
    # Write your implementation here
    return
```

Checkpoint Solution

In [7]:
```
def even_by_3(int_list):
    """
    Function that utilizes only map/filter/lambda to
    remove odd numbers from a given list and multiply
    the remaining elements by 3.

    Args:
        int_list (list): list of ints to be considered
    Returns:
        a list of even numbers from the original list,
        multiplied by 3.

    >>> lst = [1,2,3,4,5,6]
    >>> even_by_3(lst)
    [6,12,18]
    """
    only_evens = filter(lambda x: x%2==0, int_list)
    mult_by_3 = map(lambda x: x*3, only_evens)
    return list(mult_by_3)
```

In [8]:
```
even_by_3([1,2,3,4,5,6])
```

Out[8]:
```
[6, 12, 18]
```

What happens if I don't cast the output as a list?

In [9]:
```
def even_by_3(int_list):
    only_evens = filter(lambda x: x%2==0, int_list)
    mult_by_3 = map(lambda x: x*3, only_evens)
    return mult_by_3
```

In [10]:
```
even_by_3([1,2,3,4,5,6])
```

Out[10]:
```
<map at 0x7f93fa54b550>
```

map and filter are part of a class of objects called Iterators in Python. Without being explicitly called, they will perform no operations. In order to get the desired output, we had to cast the map instance into a list.

Machine Learning

Preface

Recall from discussion 1:

"Why bother learning how to code? So that we can do cool things with the tools that people have invented!"

The start of those cool things is always foundational machine learning - solving problems and finding answers through code that can "learn".

Learning always starts with data. Without having some "ground truth" to base your learning on, whatever you learn is ultimately meaningless. For today's discussion, we'll take a look at a hallmark dataset for data science, penguins:

To simplify the problem, we will not be using all features. Instead, we will only consider **'body_mass_g', 'flipper_length', and 'species'**.

	flipper_length_mm	body_mass_g	species
0	181.0	3750.0	Adelie
1	186.0	3800.0	Adelie
2	195.0	3250.0	Adelie
4	193.0	3450.0	Adelie
5	190.0	3650.0	Adelie

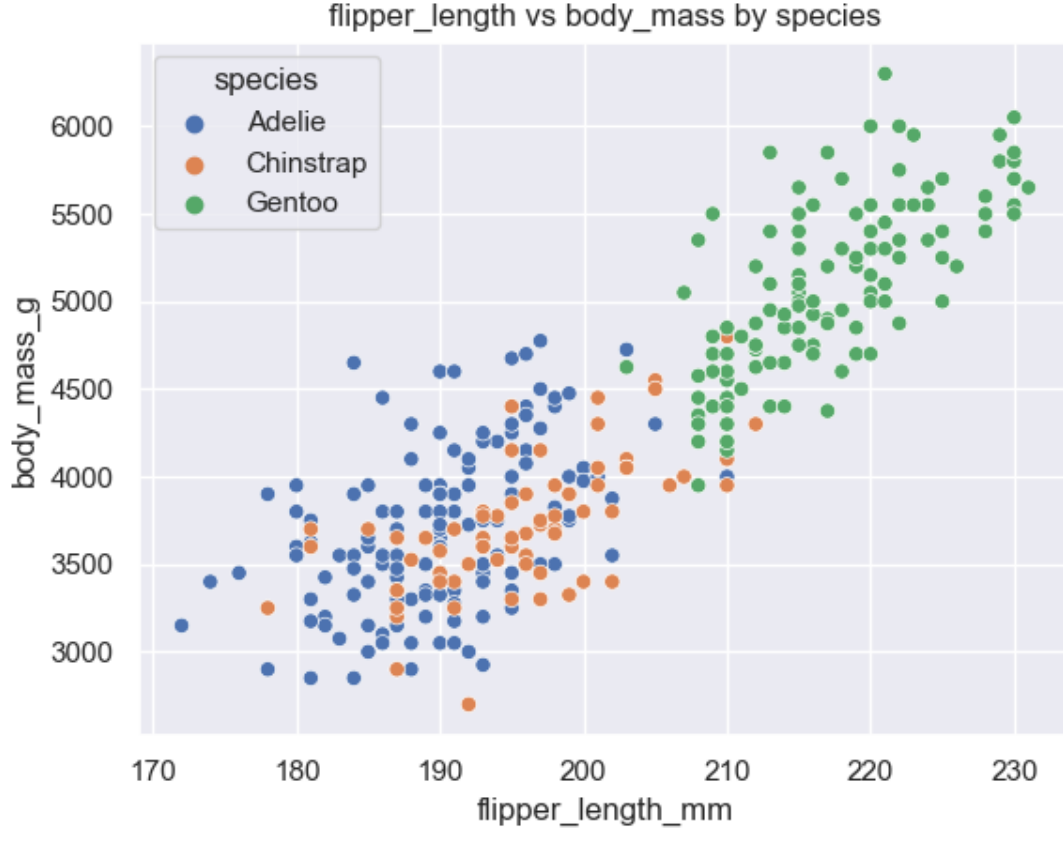
ML Vocabulary

- **fit** - attach (training) data to the model
- **predict** - According to the implementation of the model, predict the outcome of new data (based on the training data)
- **error/accuracy** - Metric to measure the accuracy of your model.

The goal of machine learning is to be able to model some real life phenomenon so that a reasonable prediction can be derived based on past trends. A very popular example within HDSI is predicting Data Science Salaries based on education and years of experience. In this example, we would **fit** our model with past data about salaries based on education and experience, measure its **error/accuracy** by running it on past unseen data, then once we achieve an acceptable accuracy, we can begin utilizing this model to **predict** future salaries!

Classifiers

Classification is one of the major tasks in machine learning. Its goal (as its name implies) is to ingest data associated with a label and be able to predict the label of future unseen data by learning some sort of pattern. Within our penguins dataset, this would involve predicting the penguin species ('species') based on its 'flipper_length_mm' and 'body_mass_g'.



In [33]:

```
clf.fit(X_train, y_train);
training_accuracy = (clf.predict(X_train) == y_train).mean()
testing_accuracy = (clf.predict(X_test) == y_test).mean()
```

training accuracy: 0.745

testing accuracy: 0.6940298507462687

Regressors

Regression is one of the other major tasks in machine learning. Its goal is to ingest (continuous) data and be able to predict a representative value for unseen data by learning some sort of pattern. Within our penguins dataset, this would be like predicting 'body_mass_g' based off of 'flipper_length_mm'.

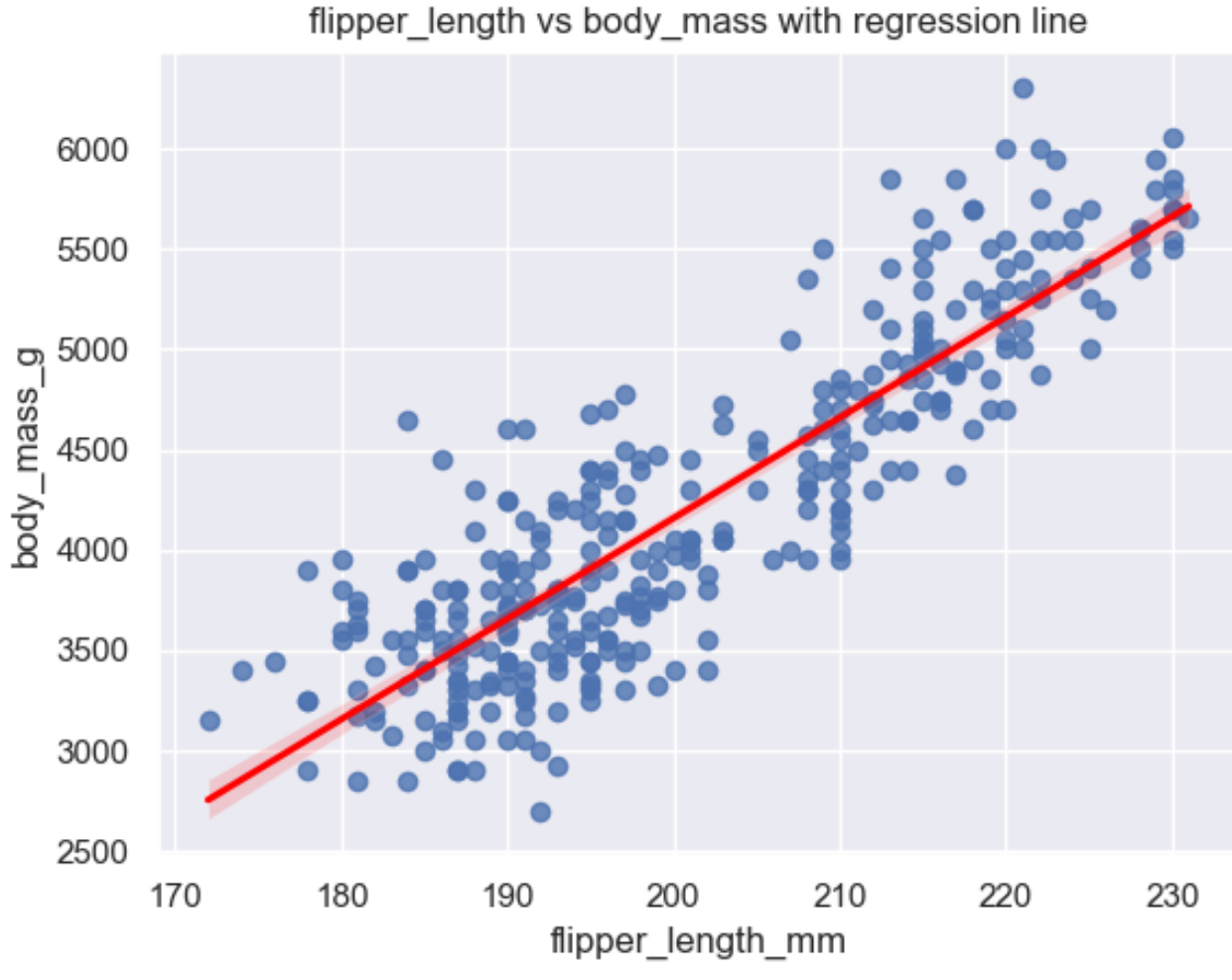
In [39]:

```
clf.fit(X_train, y_train);
MSE(clf.predict(X_train), y_train)
MSE(clf.predict(X_test), y_test)
```

training MSE: 148670.31768681708

testing MSE: 162810.850300213

Result


flipper_length vs body_mass with regression line
body_mass_g
6000
5500
5000
4500
4000
3500
3000
2500
170
180
190
200
210
220
230
flipper_length_mm

Why are we talking about this?

You'll have to implement a very basic machine learning algorithm later in the quarter for your project. The model itself is very simple, but it's generally effective and in some cases it's actually the optimal solution.

K Nearest Neighbors (KNN) for Classification

idea: Given an unseen value of flipper_length and body_mass, how can we determine what species of penguins it belongs to?

The KNN approach is simple - if I look at the k nearest points to this new value, and I take the most common species among them, then this point is **most likely** the same species as the most common species among the neighbors.


flipper_length vs body_mass by species
species
Adelie
Chinstrap
Gentoo
body_mass_g
6000
5500
5000
4500
4000
3500
3000
170
180
190
200
210
220
230
flipper_length_mm

Procedure

Step 1: Given a new point *X*, quantify the distance between all points and *X*

Step 2: Take the **k-nearest** points to *X* into consideration

Step 3: Classify *X* as the most common label among the neighbors

note: As the name implies, the key parts of this algorithm are **k** and **nearest**.
k is often referred to as a **hyper-parameter**, or a value that you choose independently. There are often procedures to select a good k, but this varies depending on the context.
Nearest refers to quantifying distance - one such way is to use euclidian distance (distance formula), but there are many other "distances" that can be used (ex. Manhattan Distance).

Checkpoint

Given the new data point (201, 4750), what would a KNN classifier classify the new point as for **k=1**? What about for **k=4**? What about for **k=10**?

Checkpoint Solution

for k=1: Seems like the orange point is closest according to an eye test, so Chinstrap

for k=4: The 3 closest points seem to all be from unique colors, the fourth point is hard to determine by eye. Could easily be Adelie or Chinstrap

for k=10: The bulk of the points nearby are blue, so Adelie

Thanks for coming!

There's a discussion quiz on canvas!