HOMEWORK 8

Due Thursday, March 16, at 11pm

Please enter your answers into the starter code Jupyter notebook and submit by the deadline via canvas.

Intersection of Line Segments. Here is a class for line segments that stores line segments by the coordinates of their end-points.

```
class LineSegment():

    def __init__(self, x1, y1, x2, y2):
        self.x1 = x1
        self.x2 = x2
        self.y1 = y1
        self.y2 = y2

    def intersect(self, other):
        # ...
        # returns the coordinates of the intersection point of self and other
        # returns None if there is no intersection point
        # can assume line segments are not parallel
```

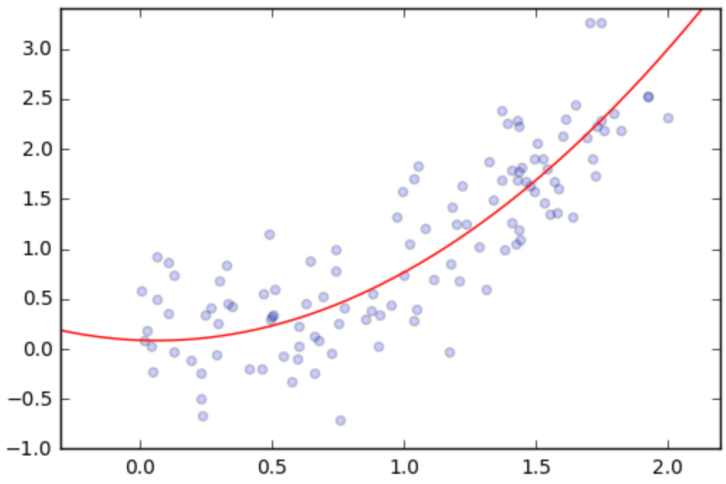
Implement the intersect function for the line segment class. You can assume that the line segments are not parallel. The function should return `None` if the line segments do not intersect.

Polynomial Regression and Overfitting. The starter code for this problem contains:

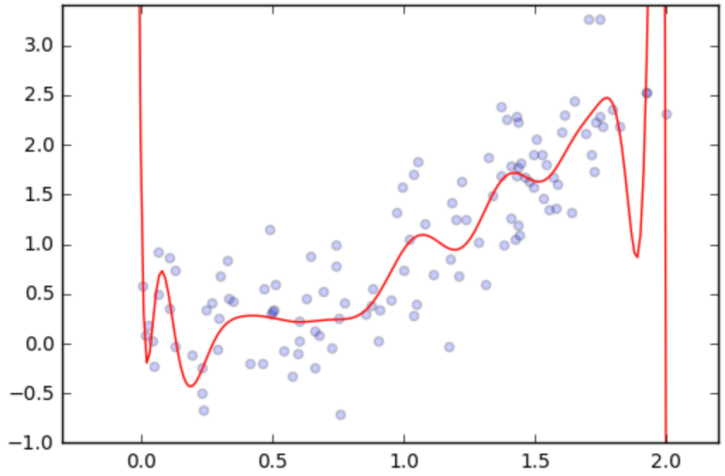
- A data-set called `data` that is provided to you as a list.
- A class called `PolyFitter(degree)` that uses the built-in linear regression model in `sklearn` to fit not just a linear function but a polynomial function of any degree (like $f(x) = ax^{10} + bx^9 + \cdots + k$) to the data. (you are not responsible for this but how does it do that?! easy: give x^2, x^3,... to the model as if they were extra information about the data points, read the code you are provided with if you are curious).

  The way PolyFitter works is as follows:

```
model = PolyFitter(2)
model.train(X, Y)
model.plot(X, Y)
```




```
model = PolyFitter(20)
model.train(X, Y)
model.plot(X, Y)
```

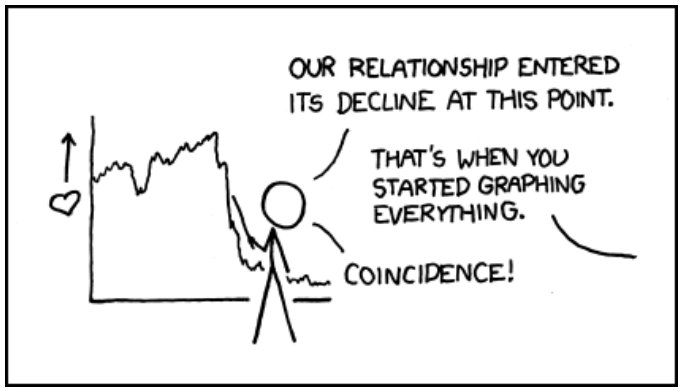


As you can see, picking a high degree polynomial fits the data much more precisely at certain points. But clearly, this degree 20 aporoximation is not really capturing the essence of this data-set. Your job is:

- Write the `mean_squared_error(X, Y)` function in the class. It should return the average of the square of the difference between the model's predictions (obtained using `self.predict(X)`) and the given answers Y. This will allow us to see how our model is doing.
- Split the data into 75% training examples and 25% test examples (You don't need to shuffe the data, just split it). More precisely, from the list `data` given to you,

make four numpy arrays. `X_train`, `Y_train`, `X_test` and `Y_test`. The training arrays should contain the x and y coordinates of 75% of data-points while the test arrays should contain the remaining 25%. We will use the training pair to train the model, and the test pair to see how our model is doing on data that it has not seen before. This is a standard method for avoiding the mistake of making models that work really well during development, but then work very badly when they see data they have never seen before.

- For each degree $d = 1, 2, 3, 4, \ldots, 20$, train a `model = PolyFitter(d)` with the data `X_train, Y_train`, plot what we have using `model.plot()` then compute its mean squared error on the training set `X_train, Y_train`, also compute its mean squared error on the test data-set `X_test, Y_test`. Plot this 'training error' and 'test error' over $d = 1, \ldots, 20$.



Source: XKCD

You should notice that as the degree increases, the training error goes down a lot, which means that our model is becoming very good at fitting to the training data. But, for the test data, it gets worse and worse. What's the best degree for this data-set?

(Optional) Linear Regression by Gradient Descent. In the lectures, we learned about linear regression, which fits a linear function to a data-set. Download the starter code for this homework from the same page where you downloaded the pdf file. The starter code should come with a file called `nba_data.csv` where we have some data about NBA players. That file should be in the same folder as the starter code notebook.

The first part of the starter code contains code for doing linear regression (using gradient descent) that makes the best possible linear function to guess the weight of an NBA player from the height (weight = a * height + b). A few parts of the code are missing: the part that fills the numpy arrays, part that computes the gradient, some of the gradient descent code, ...). Complete the relevant parts of the code to make it work. Once it works,

you should see pictures of our linear approximation at various stages during this 'training'. There are more instructions on the starter code notebook.

(Definitely email me your solution if you do this) Modify the code so that your model approximates the weight as `weight = a * height + b * age + c`. You will need three partial derivatives and update all three variables during gradient descent. Does taking age into account give you a better fit (less total error)?