Problem Set 5: Modeling Temperature Change

Out: November 21st, 2018

Due: **December 3rd, 2018 9:00 pm**
Checkoff Due: **N/A**

Introduction

In this pset, we will evaluate the controversial change in temperature over time in the U.S. Using data from the National Centers for Environmental Information (NCEI), you will use regression analysis to model the temperature of different cities in the United States.

Getting Started

Download **ps5.zip** from Stellar. This folder contains the following files:

- **data.csv**: *comma separated value* file containing the NCEI temperature data
- **ps5.py**: skeleton code
- **ps5_test.py**: tester code

Please do not rename these files, change any of the provided helper functions, change function/method names, or delete docstrings. You will need to keep **data.csv** in the same folder as **ps5.py**.

You should submit...

1. Code for your solutions in a file named **ps5.py**
2. Write-up for your discussions in a file named **ps5_writeup.pdf**

Please ***do not*** submit **data.csv**, since you won't make any changes to it.

Libraries:
In this pset we will be using the `numpy` library to store and manipulate data and `matplotlib.pyplot` to plot data.

Write-up:
In addition to writing code, you will also be turning in a write-up that will include plots you generate your answers to a few questions. If you believe your plots are wrong, you can include an additional explanation of what you think *should* have happened. Since there is no checkoff, the writeup will demonstrate your analysis of your work. Please write in **full sentences** - each question shouldn't require more than a 2 - 3 sentence answer and please don't be unnecessarily verbose.

Problem 0: The Dataset Class

In **ps5.py** we have provided you with the `Dataset` class. You will be using this class to access the data in **data.csv** for use throughout this pset. Take some time to read through the docstrings and make sure you understand what each function does (you do not need to understand how they work).

You should also open up **data.csv** and take a look at the raw data. Each row specifies the city, the average daily temperature in Celsius, and the date for the years 1961-2016.

Problem 1: Linear Regression

In this part we will write code to fit a simple linear regression to a data set. For the purposes of this problem set, you may assume that each x-coordinate is an `int` that corresponds to the year of a sample (e.g., 1997) and each y-coordinate is a `float` that corresponds to the temperature observation of that year.

*don't use imports!!

Our goal is find a line of the format $\boldsymbol{y = mx + b}$ that best fits the dataset given. Specifically, we want to minimize the **squared error for all points.** For a given point, x, with regression estimate y' and actual value y, the squared error = $(y - y')^2$.

We want to find a function that minimizes the squared error for all points in our data set - that is, Minimize $\sum_{i=1}^{n} SE(y_i, y_i')$ for *n* points. We can do this by taking the derivative and solving, which gives us the following equations:

$$\mathbf{m} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{\bar{x}})^2}$$

$$\mathbf{b} = \bar{y} - (m * \bar{x})$$

where $\bar{x}$ and $\bar{y}$ are average of the *x* and *y* values, respectively.

Implement the `linear_regression` function according to its docstring.

We want to evaluate the line we solved for above by calculating the squared error. Given the *m* and *b* parameters, and the true point values, we can evaluate this.

Implement the `evaluate_squared_error` function according to its docstring.

Problem 2: Curve Fitting

In this section you will write code to fit a polynomial regression model to a data set with samples of the form (x, y). The samples will be of the same form as in the previous problem.

Implement the `generate_models` **function according to its docstrings**. This function should return a list of best-fit polynomial models for a given data set. A model is defined here as a 1-d numpy array containing the coefficients of the polynomial; therefore, your final output is a list made up of numpy arrays. Note that the models should be in the same order as their corresponding integers in `degs`.

Example:
```
>>> print(generate_models(np.array([1961, 1962, 1963]),
                          np.array([-4.4, -5.5, -6.6]),  [1, 2]))
[array([ -1.10000000e+00,   2.15270000e+03]),
 array([  8.86320195e-14,  -1.10000000e+00,   2.15270000e+03])]
```

** Note that due to numerical errors, it is fine if you do not get this exact output, but it should be very close.*

Some helpful hints:

- Check out the documentation for **`numpy.polyfit`**.
- Note that the inputs `x` and `y` are numpy one-dimensional arrays, **NOT** Python lists. Although they are similar, *do not expect them to always behave like a Python list*. The documentation for numpy N-dimensional arrays can be found here.

Your code should pass `test_generate_models`.

Problem 3: Evaluating the Models

After we create some regression models, we want to evaluate how well they represent our data and choose the best ones. In this problem we will write a function to help us evaluate the regression models made with `generate_models` using both numerical and graphical metrics. More specifically, we will use the following:

- The model's R^2 value (aka its coefficient of determination). The function `r2_score` is provided for you in order to calculate this value.
- The plot of the data samples along with the polynomial curve fits

Implement the function `evaluate_models_on_training` **according to its docstrings**.

You should make a separate plot for each model. The format of your figure should adhere to the following:

- **Plot the data points** as individual blue dots.
- **Plot the model** with a red solid line color.
- **Include a title**. Your title should include the R^2 value of the model and the degree. If the model is a linear curve (i.e. its degree is one), the title should also include the ratio of the standard error of this fitted curve's slope to the slope (see below).
- **Label the axes.** You may assume this function will only be used in the case where the x-axis is years and the y-axis is degrees Celsius.

> *Standard Error Over Slope*
>
> This ratio measures how likely it is that you'd see the trend in your data (upward/downward) and fitting curve just by chance. The larger the absolute value of this ratio is, the more likely it is that the trend is by chance. We won't cover this evaluation method in class, so if you are interested in learning more check out: Hypothesis Test for Regression Slope.
>
> In our case, if the absolute value of the ratio is less than 0.5, the trend is significant (i.e., not by chance). We have provided you with a helper function `se_over_slope` that calculates this value for you.

Problem 4: Sampling the Data

Now that we have all the components we need, we can generate data samples from the raw temperature records and begin investigating the trend. In this problem we will try out two different methods of sampling data.

Write your code for Problem 4A and 4B under `if __name__ == '__main__'`.

Part 4A: Daily Temperature

For our first sampling method we will pick an arbitrary day and see whether we can find any trends in the temperature on this day over the years. Use the `Dataset` class to generate a data set where the x coordinates are the years from 1961 to 2016 and the y coordinates are the temperature measurements in **Boston** on **February 12th** for each year. Next, fit the data to a **degree-one polynomial** with `generate_models` and plot the regression results using `evaluate_models_on_training`.

Part 4B: Annual Temperature

Let's try another way to sample data points. Instead of looking at the change in temperature for a single day, we will look at the change in the average annual temperature. Repeat the steps in 3A but now let the y coordinates be the **average annual temperature** in **Boston**.

Implement the function `gen_cities_avg` **according to its docstrings** and use it to generate the y coordinates. **Your code should pass** `test_gen_cities_avg`. (Hint: make sure you properly account for leap years!)

For this problem in **ps5_writeup.pdf**…

- **Plot** from 4A
- **Plot** from 4B
- **4.1** What difference does choosing a specific day to plot the data versus calculating the yearly average have on the goodness of fit of the model? Interpret the results.
- **4.2** Why do you think these graphs are so noisy?

Problem 5: Long-Term vs. Short-Term Trends

Looking at trends within smaller time intervals can sometimes lead us to make different conclusions from the data. In this problem, you will make use of linear regression models to construct conflicting narratives for the change in the average annual temperature in **Los Angeles**.

Part 5A: Most Extreme Slope

Implement the function `find_interval` **according to its docstrings**. This function takes as arguments the x and y samples, an interval length, and a specified trend. It returns the indices of the start and end years of the interval with the most positive or negative slope (as specified by the trend parameter). **Your code should pass** `test_find_interval`.

Note: Due to floating point precision errors, use a tolerance of `1e-8` to compare slope values (i.e. a float `x` and a float `y` are considered equal if `abs(x - y) <= 1e-8`).

Part 5B: Increasing or Decreasing?

Write your code for Problem 5B under `if __name__ == '__main__':`.

Suppose you are the mayor of Los Angeles, and you are taking a data-driven approach to policy. Use your implementation of `find_interval` to call your citizens to action against rising temperatures! Find a window of **30 years** that shows that the **average annual** temperature in **Los Angeles** is **rising.** Plot the corresponding model with `evaluate_models_on_training`.

For this problem in **ps5_writeup.pdf**…

- **Increasing Plot**
- **5.1** What was the start and end year for your window? What was the slope?

Congratulations! The political group "Turn Down the AC" has donated 1 trillion dollars to your campaign--as long as you amend your previous statements about temperature change to a more agnostic opinion. Find a convincing window of **30 years** to show that the **average annual** temperature in **Los Angeles** is **decreasing**. Plot the corresponding model with `evaluate_models_on_training`.

For this problem in **ps5_writeup.pdf**…

- **Decreasing Plot**
- **5.2** What was the start and end year for your window? What was the slope?
- **5.3** Considering *both* plots, what conclusions might you make with respect to how temperature is changing over time?

Problem 6: Predicting the Future

It looks like we have indeed discovered some trends. Now, we are curious whether we can predict future temperatures based on what we learn from historical data. We will call data from 1961-2010 the *training* data (the data used to create models) and data from 2011-2016 the *test* data (that data used for prediction). Use the provided variables `TRAINING_INTERVAL` and `TESTING_INTERVAL` to represent these ranges in your code.

Part 6A: RMSE

Before we use our models to predict 'future' data points (i.e. temperatures for years later than 2010), we should think about how to evaluate a model's performance on this task. We can't use R^2 here, since R^2 does not have a clear meaning on testing data. Recall that R^2 measures how closely a model matches the *training* data. What we want to do, however, is to provide the model with data points that it has not seen before, i.e. the *test* data. One way to evaluate a model's performance on test data is with the Root Mean Square Error (RMSE). This measures the deviation between a model's predicted values and the true values across all the data samples.

Implement the function `rmse` **according to its docstrings.**

RMSE can be found as follows:

$$RMSE = \sqrt{\frac{\sum_{i=n}^{n}\left(X_{obs,i} - X_{pred,i}\right)^2}{n}}$$

- $X_{pred,i}$ is the estimated (predicted by the regression) y-value for the i^{th} data point
- $X_{obs,i}$ is the actual (from the raw data) y-value for the i^{th} data point
- n is the number of data points

Your code should pass `test_rmse`.

Implement the function `evaluate_models_on_testing` **according to its docstrings.** This function is very similar to `evaluate_models_on_training`, except that you should report the RMSE in the title rather than the R^2 value (Note: you *do not* need to compute the Standard Error Over Slope for this function).

Part 6B: Predicting

Now that we have a method for evaluating models' performance on test data, we are going to use our models to "predict" the future.

Write your code for Problem 6B under `if __name__ == '__main__':`.

(i) Generate more models

First, we want to generate more models for prediction:

- Use the `Dataset` class to generate a **training set** of the **national annual average** temperature for the years in `TRAINING_INTERVAL`.
- Fit the training set to polynomials of **degree 2 and 15** with `generate_models` and plot the results with `evaluate_models_on_training`.

In **ps5_writeup.pdf…**

- **Training Data, Degree 2 Plot**
- **Training Data, Degree 15 Plot**
- **6.1** How do these models compare to each other in terms of R^2 and fitting the data?

(ii) Predict the results

Now, let's do some predictions and compare our predictions to the real average temperatures from 2011-2016:

- Use the `Dataset` class to generate a **test set** of the **national annual average** temperature for the years in `TESTING_INTERVAL`.
- Evaluate the predictions of each model obtained in the previous problem and plot the results with `evaluate_models_on_testing`.

In **ps5_writeup.pdf**…

- **Test Data, Degree 2 Plot**
- **Test Data, Degree 15 Plot**
- **6.4** Which model performed the best? Which model performed the worst? Is this different from the training performance in the previous section? Why?
- **6.5** If we had generated the models using the data from Problem 4B (i.e. the average annual temperature of Boston) instead of the national annual average over the 22 cities, how would the prediction results on the national data have changed?

Hand-In Procedure

1. Save

Save your solutions as **ps5.py** and **ps5_writeup.pdf**. DO NOT submit a .doc, .odt, .docx, etc.

2. Time and Collaboration Info

At the start of each file, in a comment, write down the number of hours (roughly) you spent on the problems in that part, and the names of the people you collaborated with. For example:

```
# Problem Set 5
# Name: Jane Lee
# Collaborators: John Doe
# Time:
... your code goes here ...
```

3. Sanity checks

After you are done with the problem set, do sanity checks. Run the code and make sure it can be run without errors. You should never submit code that immediately generates an error when run!

Make sure that your write up contains everything we've asked for. In particular, your write up needs to contain two graphs from Problem 4, two graphs from Problem 5, and four graphs from Problem 6.

As always, please consult the Style Guide on Stellar since we will deduct points for specific violations.

4. Submit

Upload all your files to the 6.00 submission site before the deadline specified at the top of this document.