Problem Set 5: Modeling Global Warming

Out: November 23, 2016 11:30pm

Due: December 7, 2016 11:59pm

Introduction

President-elect Donald Trump said in 2012 that he believed climate change was a hoax created by China. In this pset, we will attempt to prove him wrong. We will use regression analysis to model the climate of different areas in the United States in order to find evidence of global warming.

- First, you will create models to analyze and visualize climate change in terms of temperature, and then consider ways to make the data less noisy and obtain clearer temperature change trends.
- You will then test your models to see how well historical data can predict future temperatures.
- Lastly, you will investigate a way to model the extremity of temperature, rather than just the increasing temperature.

Getting Started

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

Also, please consult the Style Guide, as we will be taking point deductions for specific violations. Furthermore, there will be deductions for non-descriptive variable names and uncommented code.

Besides that, you may want to check out the pylab documentation (see the numpy and scipy sections here), as pylab includes a number of functions that will make parts of this pset much easier.

You should submit

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

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

The Write Up

For this problem set, you will be submitting a write up based on the plots generated from the different problems. You *must* put plots created by your code in this writeup. Even if you believe your plots are wrong, answer the question according to your plots/code, and you can include an explanation of what you think should have happened.
As this problem set is based on exploring trends in data, we will be grading the write up more strictly based on your quality of analysis. Answer the questions in full sentences.

The Climate Class

To model the change in climate of a particular area, you will need some data. For this problem set, we will use temperature data obtained from the National Centers for Environmental Information (NCEI). The data, stored in **`data.csv`**, contains the daily maximum and minimum temperatures observed in 21 U.S. cities from 1961 to 2015. Open the file, and take a look at the raw data.
In order to parse the raw data, in **`ps5.py`** we have provided a helper class, `Climate`. You can initialize an instance of the `Climate` class by providing the name of the raw data file. Look over this class and read its docstrings to figure out how to get data for the following problems.

Part A: Creating Models

Problem 1: Curve Fitting

Implement the **`generate_models`** function. This function takes in a set of (x,y) coordinates and degrees (1 = linear, 2 = quadratic, 3 = cubic, etc), and fits polynomials of the specified degrees to the data points. It then returns the coefficients for each of the best-fit polynomials.

Hint: see the documentation for pylab.polyfit.

Function inputs:
`x` and `y` are two pylab one-dimensional arrays (NOT Python lists) corresponding to the x-coordinates and y-coordinates of the data samples. For example, if you have N data points, then

$$\texttt{x} = [x_1, x_2, \ldots, x_N]$$
and
$$\texttt{y} = [y_1, y_2, \ldots, y_N]$$

where x_i and y_i are the x and y coordinates of the i^{th} data points.

In this problem set, each x-coordinate is an **integer** and corresponds to the year of a sample (e.g., `1997`). Each corresponding y-coordinate is a **float**, and represents the temperature

observation of that year in Celsius (we will describe how exactly these observations are calculated later on in the problem set -- you don't need to worry about it for this function.) This two-array representation of data points will be used throughout the entire problem set.

`degs` is a list of integers, indicating the degrees for each regression model that we want to create. For each model, this function should fit the data `(x,y)` to a polynomial curve of that degree.

Function output:
The function should return a list of models. A ***model*** is a 1-d pylab array of the coefficients used for the polynomial. (So your final output is a list made up of pylab arrays.) The models should be in the same order as their corresponding integers in `degs`.

Example:
```
print generate_models(pylab.array([1961, 1962, 1963]),
pylab.array([-4.4, -5.5, -6.6]), [1, 2])
```

Should print something close to
```
[array([ -1.10000000e+00,  2.15270000e+03]),
 array([  6.83828238e-14, -1.10000000e+00,   2.15270000e+03])]
```

The above example generates linear and quadratic curves (degrees 1 and 2) on data samples (x_i, y_i) = (1961, -4.4), (1962, -5.5), and (1963, -6.6). The resulting models are in the same order as specified in `degs`. Note that it is fine if you did not get the exact number because of numerical errors.

After implementing this, your code should pass the unit test **`test_generate_models`**.

Problem 2: R^2

After we create some regression models, we want to be able to evaluate our models to figure out how well they represent our data and choose the best ones.

One way to evaluate how well a model performs is by computing the model's R^2 value, also known as its coefficient of determination. R^2 provides a measure of how well the total variation of samples is explained by the model. Implement the function **`r_squared`**.

Input:
`y`: a 1-dimensional pylab array, which contains the y-coordinates of the actual data samples
`estimated`: a 1-dimensional pylab array, which contains the estimated y-coordinates from a regression model

Output:
This function should return the computed R^2 value.

You can compute R^2 as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - e_i)^2}{\sum_{i=1}^{n}(y_i - \text{mean})^2}$$

Where

- e_i is the estimated (predicted by the regression) y-value for the i^{th} data point
- y_i is the actual (from the raw data) y-value for the i^{th} data point
- mean is the mean of the original data samples (i.e. y_1, y_2, … y_n)

If you are still confused about R^2, the Wikipedia page has a good explanation about its use and how to calculate it.

Some Python packages include functions that directly calculate R^2; you may not use these, and should do the calculations from scratch.

Hint: with pylab arrays, you can easily perform many operations without using for-loops. For example:

```
>>> a = pylab.array([1,2,3])
>>> b = pylab.array([1,0,1])
>>> a + b
array([2, 2, 4])
>>> a - 1
array([0, 1, 2])
>>> a * 2
array([2, 4, 6])
```

Your code should now pass **`test_r_squared`**.

Problem 3: Visualizing the Models

We've learned how to obtain a numerical metric for evaluation. Visualizing our data samples along with fitting curves can also help us figure out the goodness of obtained models. In this problem, we will integrate the numerical metrics and visualization for a comprehensive evaluation.

Implement the function **`evaluate_models_on_training`**. This function will be used to evaluate models *on the same data that we use to create them* -- aka the *training* data.

This function takes as input several data samples (`x` and `y`) and a list of `models` (which are the arrays of coefficients we obtain from **`generate_models`**) that you want to apply to your data. The function should generate a figure for **each** model. In the figure, you should plot the data along with the curve specified by the model, and report the goodness of fit with the R^2 value.

Your graph needs to match the following format:

- Plot the data points **as individual blue dots**
- Plot your model as **a red solid line**
- Include a **title** and **label your axes** (you can assume this function will only be used in the case where the x-axis is years and the y-axis is degrees Celsius)
    - Your title should include the R^2 value of your model, along with the degree of this model. Your title may be longer than your graph, and get cut off when you copy and paste the graph to your write-up. To fix that you can add the newline character `"\n"`, which adds a line break to your string, in the title (e.g., `title = string_a + "\n" + string_b`).

If the model is a linear curve (i.e., its degree is one), the title of your plot should also include the ratio of the standard error of this fitted curve's slope to the slope. (***see `se_over_slope` helper function***)
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 about it 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).

Problem 4: Investigating the trend

Now we have all the components we need. We can start generating data samples from the raw temperature records and investigate the trend.

Problem 4.I January 10th

A simple first method for sampling: we randomly pick a day from a year (i.e., Jan 10th in this case), and see whether we can find any trends in the temperatures over the years. We surmise, due to global warming, that the temperature of this specific date should increase over time.

Write your code for parts 4.I and 4.II under `if __name__ == '__main__':` after the comment `# Part A.4.`

First, generate your data samples. Each sample (data point) should be a year from 1961 to 2009 (i.e., the years in `TRAINING_INTERVAL`) and the temperature on January 10th for New York City in that year (look at the Climate class for a function to help with you this!)
Next, fit your data to a degree-one polynomial with **`generate_models`**, and plot the regression results using **`evaluate_models_on_training`**.

You will need to include the figure you generate in your write-up.

Problem 4.II Annual Temperature

Let's try another way to get data points. We surmise that due to global warming, the average temperature for each year should increase over time. Thus, we will going to plot the results of a linear regression on the average annual temperature of New York City.

Write your code for this part under `if __name__ == '__main__':` after the comment `# Part A.4.`

First, generate your data samples. Each sample (data point) should be a year from 1961 to 2009 (i.e., the years in `TRAINING_INTERVAL`) and the average temperature in New York City for that year (again, look at the Climate class for a function to help with you this.)

Hint: make sure you properly account for leap years!

Next, fit your data to a degree-one polynomial with **`generate_models`** and plot the regression results with **`evaluate_models_on_training`**.

You will need to include the figure you generate in your write-up.

Include the plots for A4.I and A4.II in a document called **ps5_writeup.pdf**. Make sure each plot has appropriately labeled axes, and is titled according to the type of model (e.g. linear, quadratic, etc.), the R^2 value, and the standard error-to-slope ratio.

You also need to answer the following questions with a short paragraph in **ps5_writeup.pdf**.

- What difference does choosing a specific day to plot the data for versus calculating the yearly average have on our graphs (i.e., in terms of the R^2 values and the fit of the resulting curves)? Interpret the results.
- Why do you think these graphs are so noisy? Which one is more noisy?
- How do these graphs support or contradict the claim that global warming is leading to an increase in temperature? The slope and the standard error-to-slope ratio could be helpful in thinking about this.

Part B: Incorporating More Data

Let's see whether we can get a better picture of climate change by using data from more than just one city.

Implement the function **`gen_cities_avg`** to get the yearly average temperature over multiple cities. Use this function to compute national yearly temperature (i.e., average the yearly averaged temperature over the 21 cities listed in `CITIES`) between 1961-2009 as your data samples. As in previous parts, the x-coordinate of a sample is an integer for the year; this time, the y-coordinate is a float for the national yearly temperature in that year.

Your code should now pass the test **`test_gen_cities_avg.`**

Again, you should fit your data to a degree-one polynomial with **`generate_models`** and plot the regression results with **`evaluate_models_on_training.`** Leave your code for this part after the comment `# Part B`, which is under `if __name__ == '__main__':`.

You will need to include the figure you generate in your write-up.

Plot the result and include it in **ps5_writeup.pdf**.

Answer the following questions with a short paragraph in **ps5_writeup.pdf**.

- How does this graph compare to the graphs from part A (i.e., in terms of the R^2 values, the fit of the resulting curves, and whether the graph supports/contradicts our claim about global warming)? Interpret the results.
- Why do you think this is the case?
- How would we expect the results to differ if we used 3 different cities? What about 100 different cities?
- How would the results have changed if all 21 cities were in the same region of the United States (for ex., New England)?

Part C: 5-year Moving Average

We are now going to generate the temperatures for samples by taking a moving average over 5 years of data. The moving average allows us to emphasize the general/global trend over local/yearly fluctuation.

Implement the function **`moving_average`**. This function takes in an 1d pylab array, `y`, and the `window_length`, which is the length of window for moving average. It returns another 1d pylab array containing the moving average results.
Here, we define moving average of `y[i]` as the average of `y[i-window_length+1]` to `y[i]`. For example, if `y = [10, 20, 30, 40, 50]` and `window_length = 3`, the array of moving averages is:

$$\left[\frac{10}{1}, \frac{10+20}{2}, \frac{10+20+30}{3}, \frac{20+30+40}{3}, \frac{30+40+50}{3}\right] = [10, 15, 20, 30, 40]$$

Note that in some cases we have less than `window_length-1` previous values to use (as in the first two calculations in the array); in those cases you should just average over the current value along with the prior values that do exist. (So for the second array value, for example, we just average 10 and 20.)

You should now pass the test **`test_moving_avg`**.

Use this function on the national yearly temperatures from 1961-2009 in order to generate the moving average temperatures with a window size of 5. Then, fit the (year, moving average) samples a to a degree-one polynomial with **`generate_models`**, and plot the regression results with **`evaluate_models_on_training`**.

Leave your code for this part after the comment `# Part C`, which is under `if __name__ == '__main__':`.

Plot the result and include it in **ps5_writeup.pdf**.

Answer the following questions with a short paragraph in **ps5_writeup.pdf**.

- How does this graph compare to the graphs from part A and B (i.e., in terms of the R^2 values, the fit of the resulting curves, and whether the graph supports/contradicts our claim about global warming)? Interpret the results.
- Why do you think this is the case?

Part D: 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.

Problem 1: RMSE

Before we use our models to predict future data points (i.e., data from later than 1961-2009), we should think about how to evaluate our models' performance.
We can't use R^2 here, since R^2 does not have a clear meaning on testing data -- R^2 measures how closely a model matches the data used to generate the model, but we are generating the model with 1961-2009 and testing on 2010-2015.

One way to evaluate a model's performance on test data is with Root Mean Square Error (RMSE), which measures the deviation of predicted values from true values.

Implement the function **`rmse`** to return the RMSE of a model, where `y` represents the actual y-values from the data set and `estimated` is the y-values estimated by the model.

RMSE can be found as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - e_i)^2}{n}}$$

- e_i is the estimated (predicted by the regression) y-value for the i[th] data point
- y_i is the actual (from the raw data) y-value for the i[th] data point
- n is the number of data points

If you are still confused about RMSE, its Wikipedia page has a good explanation about its use/how to calculate it. https://en.wikipedia.org/wiki/Root-mean-square_deviation

You should now pass the test **`test_rmse`**.

For the evaluation, you also need to implement the function **`evaluate_models_on_testing`**. This function works very similarly to **`evaluate_models_on_training`**, except that you should use **`rmse`** rather than **`r_squared`** to evaluate the prediction of your model. You should report the RMSE value in the figure's title.
You *don't* need to compute the ratio of the standard error of slope to the slope.

Problem 2: 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". We need to compare our models' predictions to the real data in order to evaluate their performance, so we will use data from 2010-2015 (i.e. the `TESTING_INTERVAL`) to simulate the future. We will call data from 1961-2009 the "training"

data set that we create the model on, and data from 2010-2015 the "test" data set which we predict the values for.

Leave your code for this part after the comment `# Part D`, which is under `if __name__ == '__main__':`.

Problem 2.I Generate more models

First, we want to generate more models for prediction. Complete the following steps:

1. Compute 5-year moving averages of the national yearly temperature from 1961-2009 as your **training** data samples.
2. Fit the samples to polynomials of degree 1, 2 and 20.
3. Use **`evaluate_models_on_training`** to plot your fitting results.

Plot the results and include them in **ps5_writeup.pdf**.

Answer the following questions with a short paragraph in **ps5_writeup.pdf**.

- How do these models compare to each other?
- Which one has the best R^2? Why?
- Which model best fits the data? Why?

Problem 2.II Predict the results

Now, let's do some predictions and compare our predictions to the real average temperatures from 2010-2015. Complete the following steps:

1. Compute 5-year moving averages of the national yearly temperature from 2010-2015 as your test data samples.
2. For each model obtained in the previous problem (i.e., the curves fit to 5-year moving averages of the national yearly temperature from 1961-2009 with degree 1, 2, and 20), apply the model to your test data (defined above), and graph the predicted and the real observations (i.e., 5-year moving average of test data). You should use **`evaluate_models_on_testing`** for applying the model and plotting the results.

Plot the the resulting graphs for the degree =1,2,20 models and include them in **ps5_writeup.pdf**.

Answer the following questions with a short paragraph in **ps5_writeup.pdf**.

- How did the different models perform? How did their RMSEs compare?
- Which model performed the best? Which model performed the worst? Are they the same as those in part D.2.I? Why?

- If we had generated the models using the A.4.II data (i.e. average annual temperature of New York City) instead of the 5-year moving average over 22 cities, how would the prediction results 2010-2015 have changed?

Part E: Modeling Extreme Temperatures

In addition to raising temperature, global warming also makes temperatures more extreme (e.g., very hot or very cold). We surmise that we can model this effect by measuring the standard deviation in our data. A small standard deviation would suggest that the data is very close together around the mean. A larger standard deviation, however, would suggest that the data varies a lot (i.e., more extreme weather). Therefore, we expect that over time, the standard deviation should increase.

In order to test our prediction, implement the function **`gen_std_devs`**. This should be very similar to **`gen_cities_avg`**.

This function returns a pylab array containing one value for each specified year. For each year in years, your function will need to:

1. Calculate a temperature for each day in that year, by averaging the temperatures for that day across the specified cities.
2. Take the standard deviation of the daily averages for the whole year.

You should now pass the test **`test_gen_std_devs`**.

Next, let's try out the function. Leave your code for this part after the comment `# Part E`, which is under `if __name__ == '__main__':`.

1. Use **`gen_std_devs`** to compute the standard deviations using all 21 cities over the years in the training interval, 1961-2009.
2. Compute 5-year moving averages on the yearly standard deviations.
3. Finally, fit your data to a degree-one polynomial with **`generate_models`** and plot the regression results with **`evaluate_models_on_training`**.

Plot the the resulting graph and include it in **ps5_writeup.pdf**.

Answer the following questions with a short paragraph in **ps5_writeup.pdf**.

- Does the result match our claim (i.e., temperature variation is getting larger over these years)?
- Can you think of ways to improve our analysis?

Hand-In Procedure

1. Save

Save your solutions as **ps5.py** and **ps5_writeup.pdf**.
For ps5_writeup.pdf , please 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 one graph from A.4.I, one graph from A.4.II, one from B, one from part C, three graphs from D.4.I, three graphs from D.4.II, and one graph from part E.

6.0002 Introduction to Computational Thinking and Data Science

Fall 2016