Living the (Zack Chauvin) Dream:
An Investigation in Meteorological Prediction

Molly Cinnamon, Grant Hoechst, Frances Shapiro, and Gil Wassermann

Our Demo!

Overview

We created a weather predictor! We designed a series of programs that implement a sliding window algorithm from NOAA (found here) to predict tomorrow's local weather (high temp, low temp, and precipitation) for Cambridge, MA. Essentially, the program uses JSON to pull data from NOAA corresponding to the last week, and to two weeks this time last year. We work that data into two matrices, perform sliding window matrix math, euclidean distance computation, and variation vector math in order to compute the high, low, and precipitation for tomorrow! Finally, we taught ourselves to combine python and HTML5 UP to create a clean-looking interface that opens locally in-browser.

Planning

We planned out different levels of our project, the most basic level was to get our weather predictor working for a hard-coded set of data for one location, and output that in the terminal. Our next goal was to have it work for one location but with the data scraped from the internet, so that it is actually accurate per day, and still in the terminal. Then we hoped to have a prettier interface and finally our reach goal was to have it work for many locations. We accomplished everything except the last goal, so our beautiful weather predictor only works for Cambridge, MA. Our original planning was well thought out because we set goals that we could accomplish, then built up from there. To see our goals and steps more in depth, our draft and final specs are below, with post-project commentary in **red**.

Design and Implementation

Our code is written largely in Python. For three out of the four members of our group (Gil had some prior experience), that involved teaching ourselves the basics of Python in order to complete the various different steps of the project. Python is an incredibly useful language, especially for manipulating matrices and doing math with the addition of the NumPy Library. We definitely had some difficulties adjusting to Python's interesting indentation rules, and once we even ran into an error that completely shut down our code until our wonderful TF Zack helped us locate a spot in which we had used four spaces instead of a tab indent.

Before our project even got too far under way, Frances and Molly took on the mammoth task of manipulating the CS51 system — primarily by pestering the TFs — until we got our section TF Zack Chauvin as our Final Project TF. This was by far the most stress-inducing part of the project, but also the part with the biggest payoff.

Splitting up the work from then on was relatively easy, and for the most part ended up being pretty fair. To start, we all read and understood the algorithm we were using, including referencing diagrams and brushing up on our matrix math (euclidean distance and variation vector, for example). We then split up the initial work:

Molly worked to generate a test data set that we could work with until we had the scraper working,

Grant implemented the matrices, including the eight sliding windows, and wrote the methods to calculate the euclidean distances and select the resulting 7-day matrix from last year with the closest euclidean distance to that of the last week,

Gil wrote the code that calculated the variation vectors for each of the two matrices "CD" (current data, i.e. last week) and "PD" (past data, i.e. last year),

Frances implemented the methods that would do additional math with those two matrices, including calculating the mean and the Predicted Variation.

After Gil and Grant combined these initial steps and cleaned up the code, we were left with an algorithm that worked effectively in the terminal for any data set we fed it, but we did only have the one sample data set. Gil and Molly, then, completed the JSON request process to access time-sensitive data and convert it into usable format, while Grant and Frances used HTML5 UP and some additional help from the web to combine Python and HTML into a clean interface to present the data. Finally, Gil and Molly created the demo video, and Frances and Grant completed this write-up.

Our final folder, then, has a few key files, while everything else is just related to our testing process or to implementing the browser-based presentation (css, for example). “api.py” handles all of the data-gathering from the web. “main.py” is the only file that actually needs to be run, and calls all the other files as needed. “main_algorithm.py” handles the sliding windows, euclidean distances, and variation vectors. “stat_calc.py” handles the means and predicted variations.

As listed in “README.txt,” simply run `python main.py` in the terminal, and your default web browser will bring up a temporary file containing tomorrow’s predicted weather! And it’s quite accurate too (assuming you don’t have a problem converting from Celsius and centimeters!).

Reflection

We were happy to find that python was very easy to learn and use for the algorithm we were implementing. Because of the NumPy Library many of our methods were written in a small amount of code and still accomplished the same job. We were also lucky to find a great description of our algorithm online, as well as access to the wunderground API to get our weather data after registering for a key.

We were faced with an unfortunate roadblock when there was an error we couldn’t figure out. Thankfully our TF helped us and also made us realize that python can be very annoying sometimes, especially when it wants a tab instead of 4 spaces. Other than that, our implementation of the algorithm, scraping the data from the API and creating the web front-end went by smoothly although it took some time.

Our choice to use python was great, suggested by our TF, especially because of NumPy. Also, gaining access to the API was a huge help so we’re lucky to have obtained the key. It also allowed us to try out a new coding language and see how we would react in the real world when given a project that requires us to code in a new language. We also love our front-end and think it is a much better choice compared to just receiving the output of the algorithm in the terminal, it makes it a lot more fun! We kept the output in the terminal as well, though, so that it’s easy to see exactly what is going on under the hood, step-by-step.

If we had more time we would surely get our predictor working for anywhere in the U.S., and next, the world! Other than that, I don't know if we would do anything different next time.

Advice for Future Students

Definitely set reachable goals before trying to tackle the coolest or most difficult project. Because we had our different levels of goals we were able to have a working project at each step, and then we could build off of what we had to improve. However, even with these goals you can't procrastinate! Next, make sure you pick the most effective language, even if you've never used it before. A language you don't know could be easier than one you do, depending on the algorithm. Lastly, make use of how smart and awesome your TF is. Our couple of meetings with Zack helped so much and we would be lost in a world where tabs are spaces without his guidance.

Our Previous Submissions

Weather Predictor Draft Specification

Molly Cinnamon, Grant Hoechst, Frances Shapiro, and Gil Wassermann

Project Overview

Our final project will consist of creating a weather predictor. We'll draw recent weather data from the web, and we'll plug that data into algorithms that will allow us to predict the weather over the next 24 hours. A number of reference algorithms are available online from resources like NOAA; we'll pull these down, analyze them so that we understand what they are doing with the data, and then modify them to suit our needs.

Our project will have two major parts. On the back end, we will use matrix mathematics and elements of machine learning to incorporate the meteorological algorithms and the recent weather data into a cohesive 24 hour weather predictor with solid under-the-hood functionality. This will involve object-oriented work in either Java, OCaml, or **Python**, along with SQL if we decide to use a database to store our data entries (depending on our conversations with our advising TF). Then, on the front end, we will develop a clean, engaging user interface to present our meteorological forecast, using a combination of **CSS, HTML, JavaScript,** and PHP.

We'll start by focusing our efforts on **Cambridge,** with a stretch goal to implement search functionality so as to be able to generate a weather forecast anywhere in the US **(Did not complete, but that's why it was a stretch goal!)**.

Feature List

Core Features

- We will scrape data from the internet about the past 30 **(14, and 28 last year)** days of weather from a meteorology site.
- We will then put that data into an excel sheet (or mySQL database) that allows for CSV manipulation **(We decided to use JSON instead of CSV)**.
- We will adapt NOAA algorithms in order to be suited to the language we are using (whether that is OCaml, Java, or **Python**).
    - this is specifically described in the following research paper: http://www.hindawi.com/journals/isrn/2013/156540/
- We will allow for a user to see the predicted weather for the next 24 hours in Cambridge, MA **Check!**.
- Users can access this predicted weather through the *terminal*.

Rockin' Extensions

Once these described base features are completed, we will move into our extended goals. These are ranked in order from easiest to accomplish to most difficult to accomplish.

- Creation of a front-end for the Cambridge, MA weather predictor, allowing for greater user-accessibility **Check!**.
    - Using **HTML, CSS,** PHP and **JavaScript**
- Host this front-end so that it web accessible from any computer **(Opted to temporarily generate locally, still accessible from any computer)**.
- Create a search feature so that the service is expanded beyond Cambridge, MA.
    - Users can enter any zip code and algorithm will scrape data from the web from that city.
    - It will be presented in an equally elegant, user-friendly way as the single city function. **X**

Technical Specification

Back-end

The back-end of this project can be broadly split into three major parts.

1) The extraction of data from the internet
2) The organization of this data into a useable format
3) The implementation of the algorithm

Extraction

The NOAA database that we will be using allows download from the internet in the form of a CSV file. One member of the team feels comfortable with doing these steps in VBA, but we will experiment with doing this step in another programming language (perhaps Java) as we are aiming to keep as many processes centralised as possible.

Organisation

Once we have the CSV downloaded (perhaps to a MySQL database that can be accessed through the website), we will need to group the data so that the algorithm will work. Although we will experiment with different algorithms, the majority of those which deal with meteorological predictions center on the use of matrices. Java, Ocaml and Python all have libraries which make matrices easy to manipulate and process. After this stage, we can run the algorithm and (hopefully) predict the weather.

Implementation

Now we will implement the algorithm. There are many algorithms that we may choose in the end, but by making the algorithm a separate stage we make the overall project easier to abstract. Maybe, we will even perform several algorithms and average and aggregate the data to make a more robust prediction. We will aim to build up the algorithm functionally.

Front-end

The front-end will take the form of a website which will display the results of the algorithm in an aesthetically pleasing manner. This will ideally be written in HTML/CSS, PHP and JavaScript. We will attempt to make the interface as user-friendly and robust to change as possible.

Next Steps/Expectations

Before we write up our final technical specification we have a few decisions ahead of us. First of all, we have to figure out our goals for the project. Our preliminary set of levels of goals is as follows: First we would like to be able to make a weather predictor that works for Cambridge, MA and outputs the data in the terminal. Second, if that is successful we would like to be able to have the predictions for the weather in Cambridge, MA (a.k.a. the Harvard area) to be seen on a front end, so basically a website that allows the user to view the prediction online, preferably in an aesthetically pleasing way. Third, if that is successful then we would like to (and this is a much bigger step) be able to make a weather prediction for anywhere in the U.S. in addition to the front end stated above. That way, the user could use the website interface to input a location and our project would output the weather prediction in that area.

Our next steps will require answers to these questions:

1. What language should we use for each part?
2. Is OCaml, java, python or something else better for matrix data math?
3. Should the information scraped from the internet be stored in mySQL or excel or another form?

Knowing these, our next step is to pick a language for these parts. Our thoughts on which languages to use are stated above. The next step would be to make a game plan- figure out who should do which part. Additionally, we’ll need to set up our environments such that we are fully ready to begin working on our project.

CS51 Project: Technical Specification

Molly Cinnamon, Grant Hoechst, Frances Shapiro and Gil Wassermann

Signatures and Interfaces

main_algorithm.py: In this class, we deal with the functionality of the “Sliding Window” algorithm itself. This will be as hidden as possible. This class takes in two arguments:

CD: A 7x14 matrix of current year data

PD: A 14x14 matrix of last years data

All functionality in this class will be dealt with in the class itself apart from the calculation of the **Variation**.

The class will expose the **Variation Vectors** of CD and PD through “get” methods (**VC** and **VP** respectively).

weather_data.py: This class looks at the available weather data and exposes the parts necessary for the algorithm to calculate the **Variation**. It will scan through the necessary .csv file and expose **CD** and **PD**. It will also expose the previous day’s weather conditions.

stat_calc.py: This is the statistical package we will create to calculate the means of the **Variation Vectors** of VC and VP. We will also implement some other classes to calculate variance, unbiased variance and maybe even Pearson’s Product Moment Correlation Coefficient if we decide to store and track the output of previous runs in order to notice trends.

This class will expose the **Variation**.

main.py: This finishes off the beast. It combines the necessary exposed parts of the project in order to output the day’s predictions.

We added api.py for data extraction, as well as other files for the front-end

<u>**Modules and Actual Code**</u>

Timeline

Week 1:

Get familiar with Python as a language

Finish the Algorithm class so that it works on a specific dataset

Tweak and improve ideas with respect to abstraction

Know which variables should and should not be exposed

Begin work on an automated download from the data website

Week 2:

Create a way for the Algorithm class to interact with the downloaded dataset

Start building a GUI, taking the form of a simple Python Applet to provide a human interface for the project

Improve speed and readability of code

Ensure functionality make robust to change and errors in the database

Progress Report

At the moment we are currently still getting to grips with the nuances of Python as a language. Although the “class” system is relatively user-friendly and approachable, we are struggling to grasp the idea of indentation as a part of the language. However, one of our goals this week is to become proficient in this language.

As a result of this, we are a little thin on code, however we have written pseudocode and general ideas in our Algorithm class in preparation for the actual code that will come later. To gauge progress, I would look at **main_algorithm.py** and **stat_calc.py**.

Version Control

For this project we have set up a GitHub repository from which we can all push and pull changes to the final project. The repository is known as **CS51FinalProjectWeather** and is hosted on Gil's GitHub account (@thewassermann).

Please feel free to give us your GitHub account so we can let you read our project as it is updated.