Machine Learning Capstone Project

Kaushik Prakash

December 2017

Contents

1 Definition

1.1 Project Overview

The weather is a very influential part of any person's life. It may affect the decisions they make on a daily basis. These decisions may be about whether or not to go to work or if it is worth it go shop that day. Thus, being able to accurately predict weather is essential for many people. Machine learning has been growing in popularity over the past decade. It can be applied to a myriad of things, weather being one of them. Currently, the weather is predicted on a real time basis based on the settings of any given day. These settings are measured using sources of weather data such as satellites, dropsondes, weather stations, and ships. Weather stations are used to measure different aspects of the weather at a fixed area. Ships on the other hand measure weather on the move.

1.2 Problem Statement

The goal of this project is to be able to accurately predict the temperature based off of past weather data. The solution to predicting temperatures that will be explored in this project is training a neural network on the past five years of weather data for a certain area, in this case Morganville. The type of neural network I will be using in this project is a multilayer perceptron. Experimenting with different epoch sizes and batch sizes will also be used to come up with the best model. The input features for the data will be characteristics on a given day such as barometric pressure and humidity. The output will be the predicted temperature on that day. The temperatures can also be analyzed over an extended period time in order to find how global warming has affected the change in temperatures. The weather prediction is a multivariate regression sort of problem. This is because there are five input features, all being combined to predict one output feature.

1.3 Metrics

In order to properly measure how well the model has learned the data, I will be using two of the metrics available in Keras that are geared towards Regression. These are mean absolute error and mean squared error. The

formula for mean absolute error is

$$\frac{1}{n}\sum_{i=1}^{n}|x_i - x|$$

Mean absolute error helps gauge how far off the predictions are in context of the actual labels, in this case the temperatures. I will also be using mean squared error as a metric for the model. The formula for mean squared error is

$$\frac{1}{n}\sum_{i=1}^{n}(|x_i - x|)^2$$

An important thing to notice is that mean squared error is the same basic formula mean absolute error, except that the error part of the formula is squared. Although this is a minor difference between the two, it has an impact on interpretation of the data. Due to the square, any large errors will have a greater influence. Mean absolute error is more resistant to outliers. Both MSE and MAE are both helpful in evaluating how well the model is doing.

2 Analysis

2.1 Data Exploration

This project requires data with a couple of features about the weather along with its corresponding temperature. Due to the nature of this problem, multivariate regression, a large amount of samples is required. Open weather map has a history bulk option available in their API. This allows me to access the past five years worth of hourly weather data. In total there are 30490 rows of data. As will be explained in the future sections, this dataset proved to be hard to work with in many ways. Unfortunately, there was no other relatively "clean" option for learning. The history bulk in Open Weather Map allows me to download a .csv file of the data for a specified area. It costed $10 to obtain the dataset for Morganville. The dataset is split into multiple columns, each with its own feature. There are two columns for the time of recording the weather characterstics. One column represents the date in UNIX format, while the other is in UTC format. The latter is what I used to understand the structure of the dataset. Columns after include the temperature, both min and max, pressure, humidity, wind

speed, wind direction, % cloudiness, weather id, weather main, weather description, and weather icon. The last four are non-numerical forms of describing the data. Below is a screenshot of the first forty data points

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	dt	dt_iso	temp	temp_min	temp_max	pressure	humidity	wind_speed	wind_deg	clouds_all	weather_id	weather_main	weather_description	weather_icon
2	1349096400	2012-10-01 13:00:00 +0000 UTC	291.2	286	291.2	980	50	7	356	69	800	Clear	sky is clear	02d
3	1349100000	2012-10-01 14:00:00 +0000 UTC	291.2	286	291.2	980	50	7	356	69	800	Clear	sky is clear	02d
4	1349190000	2012-10-02 15:00:00 +0000 UTC	291.2	286	291.2	980	50	7	356	69	800	Clear	sky is clear	02d
5	1349193600	2012-10-02 16:00:00 +0000 UTC	291.2	286	291.2	980	50	7	356	69	800	Clear	sky is clear	02d
6	1349197200	2012-10-02 17:00:00 +0000 UTC	291.2	286	291.2	980	50	7	356	69	800	Clear	sky is clear	02d
7	1349200800	2012-10-02 18:00:00 +0000 UTC	291.2	286	291.2	980	50	7	356	69	800	Clear	sky is clear	02d
8	1349204400	2012-10-02 19:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02d
9	1349208000	2012-10-02 20:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02d
10	1349211600	2012-10-02 21:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02d
11	1349215200	2012-10-02 22:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02d
12	1349218800	2012-10-02 23:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02d
13	1349222400	2012-10-03 00:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02n
14	1349226000	2012-10-03 01:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02n
15	1349229600	2012-10-03 02:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02n
16	1349233200	2012-10-03 03:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02n
17	1349236800	2012-10-03 04:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02n
18	1349240400	2012-10-03 05:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02n
19	1349244000	2012-10-03 06:00:00 +0000 UTC	289	289	292.6	980	67	8	349	99	800	Clear	sky is clear	02n
20	1349247600	2012-10-03 07:00:00 +0000 UTC	282.24	282.26	284.37	981	61	3	346	0	800	Clear	sky is clear	02n
21	1349251200	2012-10-03 08:00:00 +0000 UTC	282.24	282.26	284.37	981	61	3	346	0	800	Clear	sky is clear	02n
22	1349254800	2012-10-03 09:00:00 +0000 UTC	282.24	282.26	284.37	981	61	3	346	0	800	Clear	sky is clear	02n
23	1349258400	2012-10-03 10:00:00 +0000 UTC	282.24	282.26	284.37	981	61	3	346	0	800	Clear	sky is clear	02n
24	1349262000	2012-10-03 11:00:00 +0000 UTC	282.24	282.26	284.37	981	61	3	346	0	800	Clear	sky is clear	02n
25	1349265600	2012-10-03 12:00:00 +0000 UTC	282.24	282.26	284.37	981	61	3	346	0	800	Clear	sky is clear	02n
26	1349269200	2012-10-03 13:00:00 +0000 UTC	286.9	280.8	286.9	981	42	2	332	0	800	Clear	sky is clear	02d
27	1349272800	2012-10-03 14:00:00 +0000 UTC	286.9	280.8	286.9	981	42	2	332	0	800	Clear	sky is clear	02d
28	1349283600	2012-10-03 17:00:00 +0000 UTC	286.9	280.8	286.9	981	42	2	332	0	800	Clear	sky is clear	02d
29	1349287200	2012-10-03 18:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02d
30	1349290800	2012-10-03 19:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02d
31	1349294400	2012-10-03 20:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02d
32	1349298000	2012-10-03 21:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02d
33	1349305200	2012-10-03 23:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02d
34	1349308800	2012-10-04 00:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02n
35	1349312400	2012-10-04 01:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02n
36	1349316000	2012-10-04 02:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02n
37	1349319600	2012-10-04 03:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02n
38	1349323200	2012-10-04 04:00:00 +0000 UTC	286.6	280.5	286.5	981	37	2	330	0	800	Clear	sky is clear	02n
39	1349326800	2012-10-04 05:00:00 +0000 UTC	286.9	280.8	286.9	981	42	2	332	0	800	Clear	sky is clear	02n
40	1349330400	2012-10-04 06:00:00 +0000 UTC	286.9	280.8	286.9	981	42	2	332	0	800	Clear	sky is clear	02n
41	1349334000	2012-10-04 07:00:00 +0000 UTC	284.26	284.25	285.68	977	50	3	191	0	800	Clear	sky is clear	02n

Below is a screenshot of what the data looks like as a pandas dataframe.

```
           dt                          dt_iso   temp  temp_min  temp_max  \
0  1349096400  2012-10-01 13:00:00 +0000 UTC  291.2     286.0     291.2
1  1349100000  2012-10-01 14:00:00 +0000 UTC  291.2     286.0     291.2
2  1349190000  2012-10-02 15:00:00 +0000 UTC  291.2     286.0     291.2
3  1349193600  2012-10-02 16:00:00 +0000 UTC  291.2     286.0     291.2
4  1349197200  2012-10-02 17:00:00 +0000 UTC  291.2     286.0     291.2

   pressure  humidity  wind_speed  wind_deg  clouds_all  weather_id  \
0       980        50           7       356          69         800
1       980        50           7       356          69         800
2       980        50           7       356          69         800
3       980        50           7       356          69         800
4       980        50           7       356          69         800

  weather_main weather_description weather_icon
0        Clear        sky is clear          02d
1        Clear        sky is clear          02d
2        Clear        sky is clear          02d
3        Clear        sky is clear          02d
4        Clear        sky is clear          02d
```

2.1.1 Understanding the data

These are some statistics on the data. This gives me a quick summary of each feature.
$\mu_{temperature} = 286.4834216$
$\mu_{pressure} = 1010.48939$
$\mu_{humidity} = 67.22995179$
$\mu_{windspeed} = 3.513168684$
$\mu_{winddeg} = 183.3618354$
$\mu_{cloudiness} = 31.70818984$

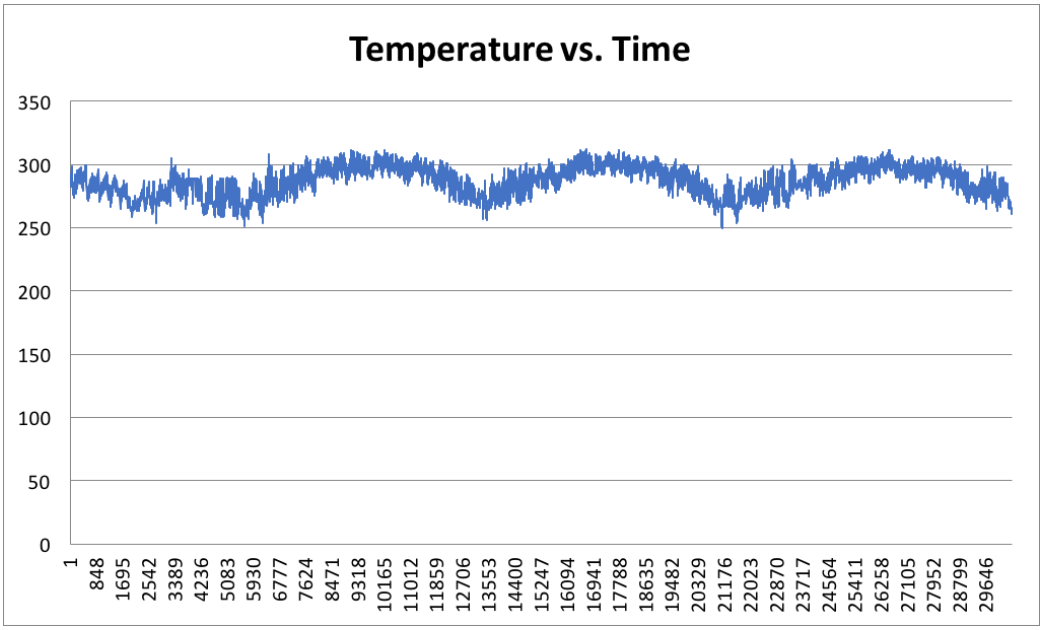
$\sigma_{temperature} = 11.31718255$
$\sigma_{pressure} = 15.49472237$
$\sigma_{humidity} = 20.94290762$
$\sigma_{windspeed} = 2.213118395$
$\sigma_{winddeg} = 101.7087166$
$\sigma_{cloudiness} = 37.77201213$

2.2 Exploratory Visualization

There are features that will be crucial to creating a relationship with the temperatures. These are presented above. Unfortuantely the data from Open Weather Map is on an hourly basis. This is a major problem for doing anything with the data. Data being recorded on a hourly basis will, no matter what, tend to repeat. There will points in the data are simply mirror images of each other with only the time being changed. An example of this is provided below

	A	B	C	D	E	F	G	H	I	J
1	dt	dt_iso	temp	temp_min	temp_max	pressure	humidity	wind_speed	wind_deg	clouds_all
2	1349096400	2012-10-01 13:00:00 +0000 UTC	291.2	286	291.2		50	7	356	69
3	1349100000	2012-10-01 14:00:00 +0000 UTC	291.2	286	291.2		50	7	356	69
4	1349190000	2012-10-02 15:00:00 +0000 UTC	291.2	286	291.2		50	7	356	69
5	1349193600	2012-10-02 16:00:00 +0000 UTC	291.2	286	291.2		50	7	356	69
6	1349197200	2012-10-02 17:00:00 +0000 UTC	291.2	286	291.2		50	7	356	69
7	1349200800	2012-10-02 18:00:00 +0000 UTC	291.2	286	291.2		50	7	356	69

As one can see all of the features and labels are the same in this example. The only difference is the time stamp. This problem can even be seen when graphing only the labels, the temperature, against time.



At first it may look like there is just a lot of noise in the data. Upon further inspection though, it is clear that some data simply repeats. If this raw data were use for machine learning, there would be many problem with over fitting and such.

2.3 Algorithms and Techniques

The first step in this project was to thoroughly preprocess the data. To do this, I created two functions that sorted through my entire list of features, each time disregarding the repeats.

This first function was created to obtain a list of all the time stamps. It was done to make manipulation of the data easier as the time stamps were now a native python list.

```
1     # Parameter is a pandas dataframe
2     # In this case it is my .csv file
3     def get_list_of_years(dataframe):
4         list_of_years = []
5         for i in dataframe['dt_iso']:
6             list_of_years.append(i)
7         return list_of_years
```

The next function that I created was where all of the calculations and conversions were made. Here the hourly data was all compiled and averaged into daily data

```
    # Get daily values for specified features
    def get_daily(dataframe, feature):
        # Get all the time stamps in a list
        list_of_years = get_list_of_years(dataframe)

        # Keep track of the same day
        match_counter = 0

        # List of combined values per day
        daily_feature = []

        # List containing every feature for the each day
        sum_feature = []

        # Loop through the entire list of days
        for i in range(len(list_of_years)):
            # Check to see if at last data point
            if i != 30488:
                current_data_point = list_of_years[i]
                next_data_point = list_of_years[i + 1]
                sum_feature.append(dataframe[feature][i])
                # If the days match
                if current_data_point[:10] == next_data_point[:10]:
                    match_counter += 1
                # If they don't match take the average value of the feature
                # and append it as one value to the list
                else:
                    daily_feature.append(sum(sum_feature) / (match_counter + 1))
                    match_counter = 0
                    sum_feature = []
            # I know that the last data point does not match
```

```
            # so I add it to the list manually
            else:
                daily_feature.append(dataframe[feature][i])
    return daily_feature
```

For the learning aspect of the project, I will be using Keras to create a Neural Network that trains on the newly split training data. A neural network is a model that has a set of neurons in different layers. A neuron itself is a mathematical function in a network of neurons, similar to the brain. Each neuron is interconnected to another neuron. Depending on the activation and optimizer functions specified, different neurons will fire. The activation function defines the output of the neuron and maps it to certain values based on the function. Optimization functions are used to minimize error. Each optimizer has its own algorithm for minimization. The optimization function is dependent upon the models learnable paramters. There will be no validation set because this is a regression task, not a classification task. A validation split is usually created to test the accuracy of classification data. Since this a regression problem and the predictions will never be perfect, creating a validation set is not the best use of the data.

2.4 Benchmark

Comparing the weather channels predictions with my own predictions was one of the benchmarks that I used in this project. To begin with, I created a relatively standard neural network architecture. I used the rectified linear unit for both of my hidden layers, considering that I knew the Kelvin value for temperature would never be negative. The output layer, quite obviously, was had 1 neuron for predicting one temperature. Comparing my models prediction with the weather channel's prediction, I noticed that mine had a deviation of about 5. It made sense to me that the weather channel had more accurate predictions. This was because the weather channel had data that was much more recent than mine. Furthermore, they were analyzing and compiling on many more features than I had access to. If I had access to more than five years worth of data and more features to train on, I will believe that the deviation would have been much lower. As you will see below I have implement a simple Ridge Regression model as a benchmark to compare results to.

3 Methodology

3.1 Data Preprocessing

The first step in preprocessing the data was to convert each feature and label to daily features and daily labels. The get_daily function has been defined above.

```
# Daily Input Features: Pressure, Humidity, Wind speed, % Cloudiness
daily_pressure = get_daily(df, 'pressure')
daily_humidity = get_daily(df, 'humidity')
daily_cloud_percent = get_daily(df, 'clouds_all')
daily_wind_speed = get_daily(df, 'wind_speed')
daily_wind_deg = get_daily(df, 'wind_deg')

# Daily Output Feature: Temperature
daily_temperature = get_daily(df, 'temp')
```

Each feature is now a python list of daily features. The next step was to combine all of these features into one list

```
features_list = []
labels_list = []

for i in range(1363):
    features_list.append([daily_pressure[i],
                          daily_humidity[i],
                          daily_cloud_percent[i],
                          daily_wind_speed,
                          daily_wind_deg])
    labels_list.append(daily_temperature[i])
```

3.2 Implementation

Below is a brief description of regression and all the different forms of regression as described in Regression Analysis on Wikipedia (https://en.wikipedia.org/wiki/Regression_analysis)

3.2.1 Basics of Regression Models

"Regression models involve the following parameters and variables:

- *"The unknown parameters, denoted as $\boldsymbol{\beta}$, which may represent a scalar or a vector.*
- *The independent variables, $\mathbf{X}$.*
- *The dependent variable, Y*

A regression model relates Y to a function of $\mathbf{X}$ and $\boldsymbol{\beta}$"

$$Y = F(\mathbf{X}, \boldsymbol{\beta})$$

3.2.2 Interpolation and Extrapolation

"Regression models predict a value of the Y variable given known values of the X variables. Prediction within the range of values in the dataset used for model-fitting is known informally as interpolation. Prediction outside this range of the data is known as extrapolation. Performing extrapolation relies strongly on the regression assumptions. The further the extrapolation goes outside the data, the more room there is for the model to fail due to differences between the assumptions and the sample data or the true values. It is generally advised[citation needed] that when performing extrapolation, one should accompany the estimated value of the dependent variable with a prediction interval that represents the uncertainty. Such intervals tend to expand rapidly as the values of the independent variable(s) moved outside the range covered by the observed data."

For such reasons and others, some tend to say that it might be unwise to undertake extrapolation.[26] However, this does not cover the full set of modeling errors that may be made: in particular, the assumption of a particular form for the relation between Y and X. A properly conducted regression analysis will include an assessment of how well the assumed form is matched by the observed data, but it can only do so within the range of values of the independent variables actually available. This means that any extrapolation is particularly reliant on the assumptions being made about the structural form of the regression relationship. Best-practice advice here[citation needed] is

that a linear-in-variables and linear-in-parameters relationship should not be chosen simply for computational convenience, but that all available knowledge should be deployed in constructing a regression model. If this knowledge includes the fact that the dependent variable cannot go outside a certain range of values, this can be made use of in selecting the model – even if the observed dataset has no values particularly near such bounds. The implications of this step of choosing an appropriate functional form for the regression can be great when extrapolation is considered. At a minimum, it can ensure that any extrapolation arising from a fitted model is "realistic" (or in accord with what is known)."

3.2.3 Making the model

These were the steps carried out in order to come up with a mode that could predict relatively accurate weather. First was to get all the basics setup. I had to make sure the network could even take in my data to train on. This was done by specifying the input shape for the first Dense layer. This only needs to be done once as Keras can predict the input shape for all other layers. The input shape for the first layer was the same as my data, (5,). The second value was left blank because Keras interprets this as None. This means that whatever dimension the other value is Keras will automatically fill it in. I didn't hardcode that second dimension for the data because it threw errors that the input shape did not match the input data. I had a two Dense hidden layers each with 32 neurons. For the activation functions, I decided to use the relu or rectified linear unit activation function for my hidden layers. The explanation for this can be found in the Benchmark section. Once the basic structure was set, I had to set up the rest of the steps. The model I was using in Keras was the Sequential model. Thus, the compile and fit functions had to be configured. In the compile function I had to specify the loss function, the metrics, and the optimizer. The optimizer function that I had decided to use was Adam. Adam is a form of a gradient descent algorithm. The logic behind adam is that in addition to storing the exponentially decaying average of past squared gradients, it stores such a average of the regular gradients as well. As discussed in the metrics section the, loss functions that I will be using are mean squared error and mean absolute error. The most difficult part of making this model was finding the correct optimizer function and determining how many neurons to have per layer. This is because the optimizers gave different output based on the

number of neurons. I solved this problem by keep a set number of neurons and using different optimizers. The 32, 32 setup with Adam ended up being the best model.

3.3 Refinement

To fine tune my model, I started experimenting with how the model reacted to changes in shape of itself. For example, making neural network longer with less neurons per hidden layer, shorter with more neurons per hidden layer, or any combination of the two. I found that relatively short and slightly wider models do better. As such I kept my model with 32 neurons per hidden layer. The next step was to see how batch_size during training affected my loss functions. During the raw creation of the model I was using batch sizes of 16. I changed the batch_size to 4, 32, 8, 10, etc. It seemed that a batch_size of 32 was the best for both achieving low loss and quick train times. All the work for this model was done using paperspace. The CPU was an Octacore Xeon and the GPU was a Quadro P4000. I had to use paperspace because my computer was not powerful enough for me to be able to train and retrain quickly, changing the parameters each time.

4 Results

4.1 Model Evaluation and Validation

In the end my model was relatively good at predicting temperatures that were somewhat on par with the actual temperatures. Obviously, with more time fine tuning the parameters and using more advanced techniques the loss could decrease even more that it has. Having access to more data and features is crucial to any model. Especially with such a complex problem as weather prediction, being able to analyze a variety of patterns is crucial for success. After loading the best model weights and testing on the test set the mae was 4.79824.

Below is my implemented Ridge Regression model where I use it as a benchmark for my results. The resulting score was a mere 15.28%. Comparing this to my models very low mae of 4.79824, it is obvious that my model learned. In addition, calculating the mean absolute error of the Ridge Regression model resulted in a mean absolute error of 21.42958. This is again

much higher than my models mean absolute error.

```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_absolute_error
clf = Ridge(alpha = 1.0)
clf.fit(X_train, y_train)
print(clf.score(X_test, y_test))
pred = clf.predict(X_test)
print(mean_absolute_error(y_test, pred))
```

4.2 Justification

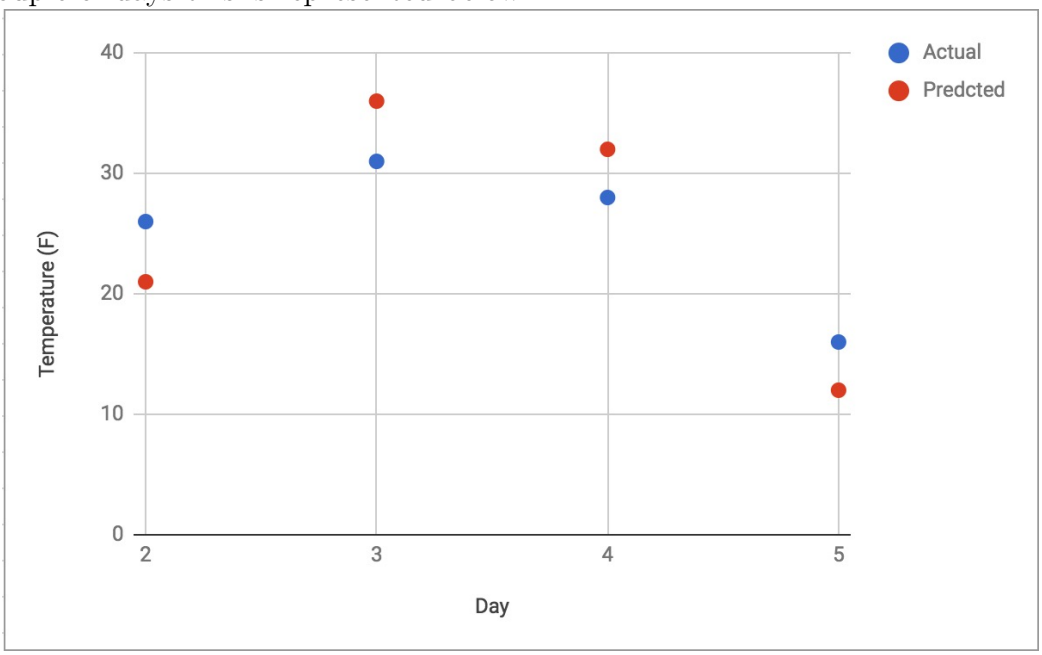
As noted above, the selected model "felt" like it was doing the right thing. I say "felt" like because I was able to compare my model's prediction with the real temperature that I was experiencing. I was able to compare the temperature to my model's predicted temperature by walking outside and using a thermometer to measure the temperature. I also checked the weather channel and compared it to my prediction. Additionally, it was clear the resulting numbers represented a much lower accuracy compared to my own model. This is proof enough that my model has learned. In addition to this, I used cross validation on my data to determine if my model have overfitted or if it had generalized well. The mae for cross validated set were similar within the range of 4.65 to 4.9. This means that my model was able to actually create a function that models the data as opposed to "memorizing the answers."

5 Conclusion

5.1 Free-Form Visualization

I was able to visualize how well my model was doing by comparing it to the real temperatures outdoors. The Open Weather Map API allows me to make calls on a daily basis to access the necessary features for my model. To test how much of an impact a difference in data made I temporarily deleted a couple thousand rows of data. I noticed that the change in loss was immense. This is clearly telling me that my model "wants" more data. Unfortunately,

I could not find a dataset that had the features that I was looking for and the necessary quantity I needed. The 5 year hourly history bulk from Open Weather Map was the closest I could get to the total dataset. I was able to compare my models predicted temperature with the actual temperature for a couple of days this is represented below.



5.2 Reflection

The steps and process I took to develop the model is outlined as follow

1. The problem is defined and researched
2. A dataset is searched and found on Open Weather Map. The history bulk section of Open Weather map costs $10. After the money is paid the dataset is downloaded
3. Functions are created to preprocess and separate the data. The data is then preprocessed, making it easier for manipulation
4. An initial raw model was defined and designed
5. In order to further improve the loss metric, research was done on many forms of regression including: linear, nonlinear, and multivariate

6. Choosing activation functions and experimenting with which ones made the most sense and resulted in the lowest metric score

7. Fine tuning the model by changing batch and epoch size

Personally I found that the last couple steps were most challenging. This is because changing the activation function can either make the model do better or worse. Not only was choosing the proper activation function difficult, but figuring out where exactly to place each one took up a lot of time. I noticed that the improvement in the loss directly proportional to the epoch size. At one point though, no matter the epoch size, the limited number of training parameters and data was a bottleneck for the model.

5.3 Improvement

There are many things that can be done to improve this model. First and foremost, becoming more familiar with the deeper, more intricate function of Keras. This would allow me to mix and match many combinations and possibly result in a very accurate model. I would also start looking into other forms of neural networks other than the MLP or Multi-Layer Perceptron. Although Convolutional Neural networks aren't renown for their accuracy in regression, maybe I would be able to create a model that does just that. Possibly even using Machine Learning ensemble methods that combine different models into one very accurate model. One major improvement for this would be to have access to more data. I believe that I am even close to point where adding more data would simply lead to overfitting. Thus, being able to analyze more features and learning on that data would be a great way to predict the weather.