Nashville, TN Weather Forecasting with Machine Learning

1. **Introduction and Motivation**

Accurate weather forecasts are often taken for granted, but they help guide business decisions and help people plan every day. One thing that sparked my interest in this topic was my morning running routine in Nashville. Having an estimate of the temperature and the precipitation was a big help on deciding how to dress. I was curious to see how difficult making weather predictions would be just by using the previous few days of weather to make educated predictions via machine learning models.

1. **Data Acquisition**

The data was requested from the National Oceanic and Atmospheric Administration website. I requested as much weather data that the site would allow which included hourly reports from early 2011 to early 2020. The initial data was sent out as a large csv file that contains a lot of extra features that were mostly empty or irrelevant. The following are the primary features that I used from the initial data set.

Predictors:

Dewpoint(degrees F), Air temperature(degrees F), Precipitation(daily total in inches), Relative Humidity, Visibility(a number 1-10, 10 being clear ), wind direction (number 0-360 degrees clockwise from due north), wind gust speed (mi/h), wind speed (mi/h)

1. **Data Preparation**

After obtaining hourly weather reports from the National Oceanic and Atmospheric Administration (NOAA), the collected data had to undergo lots of processing before it was fit to run analysis on. The first step was to reformat the date and time index. Secondly, the data needed to be converted into type float. This couldn’t immediately be done because almost every data column had a few entries with random letters thrown in. I identified what those pesky strings were and then wrote lines of code to remove them. Next, all of the hourly data needed to be reformatted into daily weather data. The reason for this is that predicting one hour of data given the previous few hours would be too easy. Predicting tomorrow’s weather given today’s and yesterday’s is a more challenging and useful problem. To accomplish this, I made 4 new data frames from the old hourly one. The 4 data frames represented the minimum, maximum, average, and sum of the hourly data grouped by day. Concatenating these data frames together results in one larger data frame that is now in terms of days and gives the min, max, and average for the daily weather parameters. Of course, it’s basically cheating to predict a parameter from today’s weather using today’s weather data. To remedy this, I created more features that would detail what the weather parameters were 1, 2, and 3 days in the past. This was accomplished by creating 3 new data frames that were shifted 1 row, 2 rows, and 3 rows respectively. After concatenating these data frames and renaming the columns, what is left is a data frame that gives a day’s weather parameters and also gives the weather parameters for the days 1, 2, and 3 days in the past. An extra step that was done next was to create dummy/binary variables for the columns that were not adequately described by their number values. An example of this is the wind direction in degrees. A wind direction value of 1 degree and of 359 degrees will look very far apart to a statistical model. However, both of these numbers represent wind coming almost straight out of the north. In this case, the degree measurement was replaced with the parameters out of east, out of south, and out of west. When doing dummy variables, you need one less parameters than there are groups to describe, hence north is not included (if the wind isn’t east, west, or south, then it has to be north by default). In the final stages of the pre-processing stage, a target variable and the predictor variables are chosen. In the beginning I chose to use all the features about the weather in the past in order to predict the binary parameter “did it rain?” on a given day. I also made a separate notebook that contains the target variable for minimum temperature. After the target variable is chosen, the data is then spit into a training group and a testing group. The training group will be used to fit models, whereas the test group will be used to quiz finished models on unseen data to see how well it performs. After splitting the data, the test group was feature scaled. The reason for doing this is because some parameters naturally have higher numbers than others and will therefore have much higher weights in the models. To level the playing field between the parameters, each parameter (besides the binary ones) was normalized by substituting in its corresponding z score. After all this, the data is finally ready start fitting models.

1. **Problem #1 : Rain Prediction, a Classification Problem**

After running a few models that tried to predict whether or not it rained on a given day, it became apparent that things were not going well. At first, I used a logistic regression model, because it is probably the most common linear model for classification problems. I also used cross validation with my logistic regression model to reduce the variance of the score. I split the data into 10 groups and had each group take a turn being the testing set. The 10 different testing scores were averaged to a final accuracy score of 70 percent. At first, I thought that this might not be too bad, but after looking the confusion matrix from one of the fitted models, I saw that there was a big problem. The confusion matrix pointed out to me that the model had predicted that it would not rain every single day. Even by doing this, it was still able to obtain nearly 70% accuracy because in almost 70% of the days in my data set, it doesn’t rain. This somewhat accurate model is not useful to me at all, because it can never predict any of the times it actually does rain. I then proceeded try out different classification models (some linear and some not) including a support vector machine, random forest, and K-nearest neighbors. After cross validation, the scores for these models turned out to be slightly worse than the accuracy score for logistic regression (they were all high 60 %). I then did a similar thing and checked the confusion matrices for these new models. While these different models were able to correctly identify a few cases where it rained, it was nowhere near satisfactory (the most correctly identified rainy cases was 15 out of over 300 rainy days).

|  |  |
| --- | --- |
| 610 | 28 |
| 333 | **15** |

This is the best model’s confusion matrix; it was only able to correctly classify 15 out of the 348 rainy days in this testing batch.

Conclusion:

Predicting somewhat rare occurrences like this did not seem too promising with the models I was working with, so I transitioned to choosing a new target variable: minimum temperature.

1. **Problem #2 : Morning Temperature Prediction, a Regression Problem**

I chose to focus on predicting the minimum temperature for a given day next. The minimum daily temperature best resembles the temperature that I would experience while outside early in the morning.

* 1. **Linear Models**

The model I chose to initially explore this regression problem was an OLS linear regression model. I started with a full model and observed the p-values from the model. I considered all p- values less than 0.5 to be significant. I also made a correlation table to see which variables had a strong linear relationship with the target variable of min temperature. After omitting parameters that had high p-values and low correlations, I would run a new linear regression model and again observe the parameters p-values and correlations. It is important to note that A close up of text on a white background

Description automatically generatedResidual Plot 


Description automatically generateda parameter having a low correlation value just means that it doesn’t have a strong linear relationship with the target; it could very well have some other strong nonlinear relationship. However, linear regression assumes that the parameters have a linear relationship with the target variable. A few models later after continuously tossing parameters out, I was left with only 8 parameters. I decided to run a slightly more detailed analysis of this model. The fitted linear regression model using 8 parameters received a r^2 score of 0.889 and a √MSE of 5.61. I also cross validated the r^2 score by splitting the data into 10 folds and got r^2 = 0.886 which was very close to the original model. On the left is the residual plot for the training set. From this, we can gather that the model is a lot better at predicting minimum temperatures that are between 60 - 80 degrees F and struggles more at predicting the colder temperatures. It also seems like on average the model overestimates cold temperatures and overestimates the warm ones (hence the slightly positive slope). I then used this same linear regression model to try and predict a test set that it had never seen before. The residual plot for the test set is the bottom one. This residual plot tells a very similar story to the one above it. The √MSE for the test set was 5.53. This is lower than the error for the training set, so it is clear that we are probably not overfitting the data. Of course, a plain old OLS linear regression model is not really ever going to overfit data, but it doesn’t hurt to check.

A screenshot of a cell phone

Description automatically generated A linear regression model also assumes that its parameters are independent of each other. I knew that this was probably not the case for my model, but I decided to check with another correlation plot for the 8 parameters. The correlation values were atrocious; there was definitely multicollinearity at work in my model (in other words, there were probably a lot of redundant parameters). To remedy this, I decided to move down to a smaller model. First, I ran a simple (1 parameter) linear regression model using each of the old 8 parameters in order to predict the target. From these results, I selected 2 parameters that appeared to have the strongest linear relationship with the target base on their r^2 values. These two parameters happened to be the dew point from 1 day ago and the average air temperature from 1 day ago. From the correlation table, it is clear that these 2 parameters are also strongly correlated. To account for this in my new 2 parameter model, I included a term in the model where those 2 parameters were multiplied together (the goal of that is to account for how each of the variables affect each other). With this 2-parameter model, I repeated the exact same process as I did with the 8-parameter model, so I am not going to explain every detail, but just simply provide the results. The r^2 score of the model was 0.877 and a cross validated r^2 score of 0.872. The √MSE for the training set model was 5.90 and the √MSE for the unseen testing data set was 5.77. The residual plots are included below and are very similar to the ones from the 8-parameter model.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Summary:

The analysis indicates that the 2-parameter regression model does almost just as well accuracy wise as any of the other higher parameter regression models. However, this amount of error is not really low enough to be practically useful. The √MSE indicates that, on average, the model is off by about 5.8 degrees F or so. Most of the time, it is wrong by less that (most of the data in the residual plot is clustered in between + and – 5 degrees) The fewer times that it is wrong by a large amount increase the error by a lot. Treating the √MSE like you would treat standard deviation, a 95% confidence for the errors/residuals would come out as plus or minus 11.6 degrees from the estimate. This confidence interval seems too wide to be of much practical use, but it isn’t too bad.

* 1. **Non-linear Models**

The 3 non-linear regression models that I ran on my model include random forest regressors, support vector regressors, and k-nearest neighbors regressors.

I started off this section of the project by deciding on the features that I would use with my model. After running a couple of models, I decided to use a consistent set of 8 predictors for the models. A set of 8 provided better r^2 and RMSE scores than the much smaller sets I tried. Also, the set of 8 wasn’t so big that it carried as much redundant information from multicollinearity as a set of 20-50 features would have.

Random Forest Regressor

The first model I tried was the random forest regressor. I used a package called GridSearchCV in order to find an ideal combination of parameters for the model. For this model, I chose various different numbers for the max depth of the trees and for the minimum samples per leaf. The max depth represents how many total splitting stages/levels a tree can have. The minimum sample per leaf won’t allow a leaf unless it has the minimum number of samples in it. After running a 10-fold cross validation over these tuning parameters, the most effective set of parameters were a max depth of 10 and a minimum sample per leaf of 10. I then constructed a new model that had these tuned parameters in place. I fit the model with the training data and then ran a 10-fold cross validation with the testing data. The average r^2 score from the test data ended up being 0.875 and the RMSE was 5.68.



It is hard to show a picture of a random forest, but I made a decision tree that used the same optimal parameters as the forest. This is a picture of it. You have to zoom in a lot to read at what values it splits at. For the random forest model, I found that the more trees it had, the better the score. At a certain tree count, the model will take too long to load, so I settled on 100 trees for each of the random forest models.

Support Vector Regressor

Similar to the last model, I used the exact same process with GridSearchCV in order to tune the parameters for the model. I tried 2 different types of kernels (rbf and polynomial) and I also tried various values for C. The greater C is, the higher the penalty the model uses for having errors in the training set. A larger C value will make the model fit the training set better (lower bias but higher variance). For my model, the best combination of parameters had a C value of 100 and a polynomial kernel. I then made a new model using these specific parameters. After training the model and running another 10-fold cross validation with the test set, the r^2 value came out as 0.887 and the RMSE was 5.45.

KNN Regressor Model

GridSearchCV came in handy one final time here. In this model, I was trying to tune the n neighbors and also the weights. The optimal pairing came out to be 20 neighbors and distance related weight. The parameter 20 means that the nearest 20 data point will be taken into account when trying to decide a value for the unknown data. The distance weight means that data points closer to the unknown data point will now have a higher weight in the decision than datapoints further away. A new model was created with these parameters and cross validation was used with the testing set to come up with a r^2 score of 0.874 and a RMSE of 5.60.

It is hard to find any visual way of representing regression problems that aren’t 2 dimensional, which is kind of sad. I did graph the residual plots for each of the 3 models though. All 3 residual plots look very similar to each other and also to the residual plots of the linear models. Just like with the linear models, these models have a harder time estimating the minimum temperature when it happens to be low. They are much more accurate when it comes to estimating a low temperature that is around 60-80 degrees F. I thought maybe this was simply a flaw of the linear models, but it turns out that for some reason it is just inherently harder to estimate the minimum temperature in the winter months (this quite possibly might A close up of a map

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedbe due to the increased variability of these temperatures).

* 1. **Model Comparison**

There were 5 main regression models used to try and predict the daily low temperature in this project. The first 2 models were multiple feature linear regression models. One model used 8 features and the 2nd model only used the most effective 2 features out of the 8. Three non-linear models were used: a random forest regressor, a support vector regressor, and a KNN regressor. For a regression problem, the main 2 score metrics that can be used to evaluate a model are the RMSE and the R^2 value. The RMSE (root mean squared error) measures the average error for the model predictions. This error will be in degrees F, since that is what we are predicting. The R^2 metric is a measure of the proportion of variation in the Temperature that is explained by the variance in the predictor variables in the model. In order to make accurate comparisons between the models, I used the same training and testing sets for all of these models. (The 2-feature linear model only uses 2 columns from the training and testing sets instead of the full 8). Below are some charts that summarize the scores of the regression models:

Do notice that I included two R^2 charts. I just wanted to show that sometimes visuals can be misleading depending on how they are scaled. All the R^2 values were within 0.013 of each other, and this is hardly enough of a change to be significant. The best performing model based on these metrics was the SVR model. It had a RMSE that was almost a whole degree below the other models. Even if some of this just happens to just because of a lucky test set draw, the difference is still significant. This one model score aside, most of the models performed very similar to each other. Even the simple 2 feature linear regression model did surprisingly well. Despite the fact that some of the other models slightly edged out the linear models, the linear models might be the best way to model this problem because they are by far the simplest to implement. The simplest linear model scored just barely behind the other more complex non-linear models; this might mean that the complex models don’t really capture that many more real trends/patterns in the data.

1. **Conclusion**

In summary, the best models for this regression problem seem to be linear regression due to the fact that it scores fairly well considering how simple it is. You could also argue that SVR is a good choice because of its significantly lower error score. However, this model is more complicated and takes more processing power and more time to run. A major takeaway from this is to use simpler models when they work. Considering that a simple 2 feature linear regression model performs similar to the more complicated models, these more complicated models might be fairly redundant and not fully utilized. However, some problems including the classification of Nashville’s rainy days might require more sophisticated models or techniques in order to get better predictions. My findings from this project also match up with what is found in real weather reports: rain is often hard to predict (this is messed up commonly), while the temperature predictions are fairly accurate.

1. **Further Exploration**

This project could be continued by implementing more advanced models and techniques in order to predict the rainy days in Nashville. A logical next step would be to use a neural network for this classification problem. Another possible tactic that might help is to try and gather or engineer extra data of rainy days in order to better balance out the ratio of rainy to non-rainy days.

The set of features that I chose were determined through the linear model process that I did at the beginning of the project. This means that I know the chosen 8 features were the best features for predicting the target variable using linear models, but it also means that some of the variables that I eliminated might have certain non-linear relationships that I didn’t care about at that point. It might be possible to get higher test scores if I would reevaluate the features that I would use before each new model.

**References**

**8.1 Source Code**

<https://github.com/ihollars/weatherProject.git>

**8.2 Works Cited:**

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. New York :Springer, 2013.

McQuistan, Adam. “Using Machine Learning to Predict the Weather.” Stack Abuse, Stack Abuse, 2018,stackabuse.com/using-machine-learning-to-predict-the-weather-part2/#disqus\_thread.

“National Oceanic and Atmospheric Administration.” National Oceanic and Atmospheric Administration, NOAA, 2020, www.noaa.gov/.

scikit-learn developers. “User Guide¶.” Scikit-Learn.org, 2019, scikit-learn.org/stable/user\_guide.html.

the pandas development team. “Getting Started¶.” Getting Started - Pandas 1.0.3 Documentation,2014,pandas.pydata.org/pandasdocs/stable/getting\_started/index.html#getting-started.