Using Machine Learning to Predict Tornadoes for use in Sub-Seasonal to Seasonal Forecasting

by

Scott Alan Burgholzer

Dr. Cynthia Howard, Advisor

Masters project submitted in partial fulfillment of the

requirements for the Master of Science in Data Science degree

in the College of Arts and Sciences of

Lewis University

Using Machine Learning to Predict Tornadoes for use in Sub-seasonal to Seasonal Forecasting

Scott A. Burgholzer[1]

[1] Lewis University, Romeoville IL 60446, USA
scottaburgholzer@lewisu.edu

Abstract. Machine learning has become an interesting topic within the meteorology community. There are many existing examples where machine learning has been implemented and shown to have improvement over some existing meteorology methods. This study attempts to find the optimal machine learning algorithm, specifically using neural networks, to predict if a day will have tornadoes based on meteorological parameters. The purpose of this study is to find the optimal values for these parameters that can be used in Sub-Seasonal to Seasonal forecasting.

Keywords: Machine Learning, Meteorology, Sub-Seasonal to Seasonal Forecasting.

1 Introduction

This study attempts to find the optimal machine learning (ML) algorithm that can predict if there will be five or more tornadoes on a given day. The eventual goal is to use this model to find the optimal meteorological parameters for tornadic activity so that these values can be used to improve Sub-Seasonal to Seasonal (S2S) forecasting.

There are some early indications in this study that indicate this could be possible. One algorithm had acceptable results for 12Z conditions. This same algorithm did not have acceptable results for 00Z but a second algorithm did have acceptable results for 00Z conditions. There were concessions made in this study that may have a role in the effectiveness of these algorithms. Future work will be needed to see if machine learning algorithms do have potential in improving tornado forecasts.

2 Sub-seasonal to Seasonal Forecasting

Sub-seasonal to seasonal forecasting (S2S) generally is predicting weather two weeks to a full year in advance. One big difference between weather forecasts and S2S forecasts is that weather forecasts rely heavily on initial atmospheric conditions as initial input, while S2S uses the initial conditions in addition to effects of other slowly evolving boundary conditions. These boundaries can include sea surface temperature, sea ice, and soil moisture [1].

Another major difference is weather forecasts give specific values to meteorological conditions, while S2S uses probabilistic values to state if meteorological conditions will be above, below or near normal [1].

3 Existing Machine Learning Research in Meteorology

3.1 Using Machine Learning in S2S Forecasting

When seasonal forecasting began, simple statistical techniques were being used to predict the forecast. In today's world, however, there are complex forecast models that are used for S2S prediction and are being favored over the statistical methods previously used [2]. There are a few studies discovered that combine machine learning and sub-seasonal to seasonal forecasting.

Cohen et al. specifically used clustering methods and multilinear regression and compared the results to canonical correlation analysis (CAA) and the North American Multi-Model Ensemble (NMME). In two different studies they proved that these ML methods had vast improvement over CAA and NMME [2].

The first study used hierarchical clustering to create winter precipitation hindcasts in Europe. The results showed that the hierarchical clustering skill was much higher than CAA and NMME. The second study looked at winter surface temperature hindcasts. In addition to the previously used hierarchical clustering they also used the Ward method. In this case, they used permutations of meteorological inputs into the models to create 127 models. This study found the skill score of these models were generally higher than the skill score of the NMME [2].

Hwang et al. used ML to participate in a contest put on by the U.S. Bureau of Reclamation and the National Oceanic and Atmospheric Administration. In this study, an ensemble of two types of regression models were used. This study showed that the models on their own and combined in the ensemble performed better than the Climate Forecasting System (CFSv2) [3].

Homstrom et al. used linear regression to predict high and low temperatures. This study also used a variation of functional regression that predicts the weather based on searching for similar conditions in the past. This study showed that professional weather forecasting continually performed better than the ML models. This study did note, however, that while the differences between the ML models and professional forecast were larger early on, it diminished as time went further in the future. This study noted that with the decreasing error, ML might have potential in a longer-range forecast than professionals [4]. This study shows the need for continued research in S2S timeframes with ML.

3.2 Using Machine Learning in Other Meteorological Applications

While sub-seasonal to seasonal forecasting and machine learning is a growing field within meteorology, ML has already been applied in many other meteorological applications. One such application is to identify Midlatitude Mesoscale Convective Systems (MCS) in Radar Mosaics. This study was performed by Alex Haberlie and Walker

Ashley. This study specifically used three ML algorithms; random forest, gradient boosting and "XGBoost". The purpose stated in the study of using these algorithms was to reduce false positive identifications of MCS [5].

Liu et al. used Deep Neural Networks (DNN) to investigate using it for forecasting different meteorological conditions. A 4-layer DNN was used to predict temperature and dew point for the next time point. In this case, the study showed the DNN was able to predict temperature with very little error. This study also looked at predicting Mean Sea Level Pressure (MSLP). This DNN had obvious errors between actual results versus predicted results [6].

McGovern et al. study provided multiple examples of ML in meteorology. One example was to predict the duration of a specific storm. This example used gradient boosted regression trees, random forests and elastic nets. The study showed that random forests and gradient boosted regression trees had the best results with 100 different trees with each having a max depth of 5 [7].

Ghosh and Krishnamurti used a generalized regression neural network (GRNN) to improve hurricane intensity forecasts. This study's results used different hurricane seasons as inputs. The first season was the 2012 season. The results of GRNN for this season showed that there was a major reduction of errors in the forecasts. The absolute error between forecast hours of 24 and 120 were between 7 and 9 knots. The 12-hour forecast was close to 6 knots. The neural network showed a 10% reduction in error for forecast hours between 48 and 120 hours compared to other models [8].

Wimmers et al. used ML to estimate the strength of a hurricane based on data from weather satellites. This studied produced a model called DeepMicroNet. This model had results that were 16 miles per hour different from historical data of human forecasts. This study also looked at using aircraft data for inclusion into the model, which then produced differences of 11.5 MPH [9].

Liu et al. used a Convolutional Neural Network to detect extreme weather by using climate datasets. The CNN in this model had 4 layers, 2 of which are convolutional layers and the other two are fully connected layers. The convolutional layers also had a max pooling layer behind them. The activation function used in the convolutional layers and first fully connected layer was the Rectified Linear Unit (ReLU). The second fully connected layer used a logistic activation function. The results of this CNN had an accuracy between 89% and 99% for classification [10].

There are other studies that used machine learning in meteorology, but these give examples of ML being used in areas of interest to this author. These studies also give hope for continued success of incorporating ML into meteorology, and into S2S.

4 Ethical Considerations

There are no ethical considerations for this study. The reason is there is no personally identifiable information anywhere, even in the Storm Prediction Center's tornado dataset. That dataset does include number of injuries and number of fatalities. This study does not use either of those fields, and in addition, those fields are simply the number of people affected. If this study did use those fields, there would still be no ethical

considerations due the fact it is only a numeric value and we never had access to direct personal information.

5 Methodology

5.1 Expected Limitations

There are many factors that are expected to limit the results of the study. The first being the short time frame allowed for this study. This study was part of a capstone course that was 8 weeks long. The first couple of weeks were used mostly for fulling requirements of the course, ethical courses and certifications and a proposal. Week 7 was used mostly for the rough draft of the paper and for creating the oral presentation. Week 8 was for turning in the final paper and presentation. This left 4 weeks for implementing this study.

Due to the time constraint, some concessions had to be made to complete the study. The first was to use only one severe weather event, tornadoes. Additional preprocessing would need to be done to extract the needed wind and hail reports. The author did not have time to extract and clean the data and chose to allow as much time as possible for the machine learning model development.

The study used the *North American Regional Reanalysis (NARR)* [12] from the *Research Data Archive (RDA)* at the *National Center for Atmospheric Research (NCAR)/University Corporation for Atmospheric Research (UCAR)* [28]. The dataset available from this source had a total size of 6.92 TB with data from January 1, 1979 through October 31, 2019. As will be explained below, the NARR has the domain of the Continental United States, but for this study it was decided to use a small sector to create a smaller domain as an attempt to speed up processing. This meant the full 6.92 TB of data would need to be downloaded. RDA restricted to only downloading 10 files at a time, and as such, was not a viable option. Another source of the data was found, but the dates of available data went through October 2014. This is another limitation that the full data set cannot be used, and as such, may miss other days that could give more data to the machine learning models to learn from. Due to this, only 12,783 days of data were usable and 11,180 of them had less than five tornadoes which leaves only1,603 with five or more.

The NARR has grid points, one every 32 km in the vertical and horizontal when viewed on a 2D surface, where vertical is latitude and horizontal is longitude. It is hypothesized that using each grid point in the machine learning algorithm would make it more accurate. The reasoning is there could be good convective parameters in Texas while not so good in Montana. With averaging, data gets washed out and as such the algorithm does not know there can be cases of localized high values. Due to this and the fear it would take too much processing power and time, the NARR was restricted to a subsection of the Continental US (CONUS). Then each variable used in this subsection was averaged to speed up the processing. As such, these two data manipulations are expected to reduce the accuracy of any machine learning algorithm.

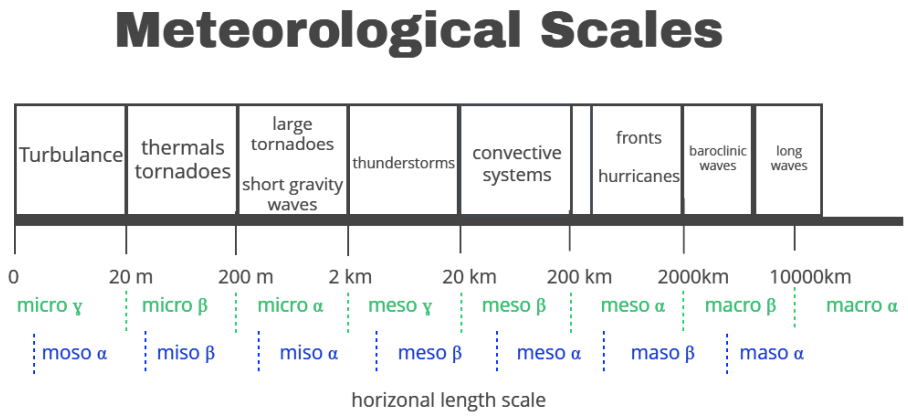

Fig. 1. Meteorology scales. Green is Orlanski (1975) and blue is Fujita (1981)

Another expected limitation is from the numbers given above. There are more days with less than 5 tornadoes than there are with more than 5. The author had a hypothesis that this might limit the accuracy of any models created. The reanalysis model used in this study had a small selection of convective parameters by default, and while other convective parameters could be calculated from the default parameters, there was simply not enough time to do so. As tornadoes are convective in nature, the limited number of convective parameters is also hypothesized to restrict the accuracy of the model.

There are two scales of meteorology that can be used to group meteorological events together. Orlanski came up with the first scale definitions in 1975 then Fujita simplified them into fewer groups in 1981. As seen in Figure 1, tornadoes fall into micro β or micro α as defined by Orlanski, or by looking at Fujita fall into the tail end of moso α, all of miso β and most of miso α [11]. The NARR has a grid of 32 km by 32 km which falls into Orlanksi's meso β and either in Fujita's meso β or meso α [11,12]. This means the NARR is unable to accurately predict tornadoes due to the differences in scale. The NARR has a better chance of resolving thunderstorms over tornadoes. This is one limitation of this study that cannot be changed.

For the purpose of this study, it was decided to use binary representation of 0 being less than five tornadoes on the day and 1 being five or more tornadoes on the day. This was decided due to the difference of scales between tornadoes and the NARR. The tornado data came from the *Storm Prediction Center (SPC)* [29]. This data set has inherited limitations as only reports that were seen by a person is reported. This is especially true in the earlier years of the dataset. With the growth in population and changes in technology, tornadoes have a higher probability of being noticed. This is part of the expanding bullseye effect noted by Strader and Ashley [35].

The short time frame also meant that trying to use different machine learning algorithms and spending time to manipulate their parameters to improve their accuracy was not possible. The author decided to use a neural network as it was hypothesized that it might be better at solving the complexity of the atmosphere.

Table 1. CAPE Typical Values from Vasquez [15]

Value	Description
< 300 J/kg	Mostly stable, little to no convection
300-1000 J/kg	Marginally unstable, weak thunderstorms
1000-2500 J/kg	Moderately unstable, severe thunderstorms possible
2500-3500 J/kg	Very unstable, severe thunderstorms, possible tornadoes
3500+ J/kg	Extremely unstable, severe thunderstorms with tornadoes likely

The final expected limitation is not being able to tune the parameters of the models, specifically epochs, optimizers, learning rates, momentum, etc. This was expected to limit the accuracy of the models as possible values to get the most accurate version might not be found.

5.2 Data Descriptions

Due to the limitation described above about downloading the NARR data, the data was instead downloaded from *National Centers for Environmental Information* NARR page. This dataset was from January 1, 1979 through October 2, 2014. This study used 1979 through 2013 to include only full years for the data. The full dataset was not needed to be downloaded as the site separated files by the format of YYYYMM/YYYYMMDD/narr-a_221_YYYYMMDD_HH00_000.grb. As this study used only 12 Zulu and 00 Zulu data, two files per day were downloaded instead of all three hourly files [13].

The tornado data came from the Storm Prediction Center (SPC) severe weather database. The dataset had data from 1950 through 2018, and the specific file used was the 1950-2018_actual_tornadoes.csv. This file, unlike the others available on the site, only stores the initial point of the tornado. The other datasets would list the same tornado in multiple rows if it passed through a certain number of counties or over state lines. Since SPC already did the work of filtering the dataset, it was decided to use it to save time [14].

The NARR variables used are as follows: 2 meter Temperature, 2 meter Dewpoint Temperature, Surface Convective Available Potential Energy (CAPE), Surface Convictive Inhibition (CIN), Surface Potential Temperature, Surface Pressure, Precipitable Water (PWAT), 2 meter Relative Humidity (RH), Storm Relative Helicity (SRH), Lifted Index, and 10 meter wind speed. In addition, at 250 millibar (mb), 500 mb, 700 mb, 850 mb and 925 mb the following variables are used: Geopotential Height, Temperature and Wind Speed.

CAPE is a predictor for thunderstorm potential and severe weather potential. This parameter was defined in 1982 by Morris Weisman and Joseph Klemp. The units of this parameter are joules per kilogram (J/kg). CAPE can be seen when a parcel of air lifted from some level of the atmosphere is warmer than the temperature of the atmosphere. Table 1 shows typical values for CAPE [15]. Since CAPE is a stability (or

Table 2. Typical values of CIN from Vasquez [15]

Value	Description
< 0 J/kg	No cap
0 to 20 J/kg	Weak capping
21-50 J/kg	Moderate capping
51-99 J/kg	Strong capping
100+ J/kg	Intense cap, storms are not likely

Table 3. Typical values of LI from Vasquez [15]

Value	Description
> 2	No significant activity
2 to 0	Showers and/or thunderstorms possible with a difference source of lift
0 to -2	Thunderstorms possible
-2 to -4	Thunderstorms likely, few may be severe
< -4	Severe thunderstorms possible

instability) measure, it is of high importance in trying to predict thunderstorms and tornadoes, and thus included in this study. There are different CAPE values, including surface, mean layer, and most unstable, though the NARR only has surface CAPE, so that is the only one used in this study.

CIN or *CINH* is the same calculation as CAPE, only in this case the parcel is colder than the atmosphere. This is stable air whereas CAPE is unstable air. The units are still J/kg. The higher the value, the more stable the atmosphere is which in turn makes it harder for storms to form. Table 2 shows typical values of CIN [15]. CIN is important to look at along with CAPE because you could have CAPE higher in the atmosphere, but if there is a cap (strong CIN) lower in the atmosphere, lifted parcels of air are not likely to get through and can prohibit any storms from forming.

Lifted Index (LI) is another stability index that was used mostly before the 1980s but can still be used today. It uses the same idea as CAPE and CIN of lifting a parcel of air, but in this case to a specific height instead of the entire atmosphere. It is accepted that it was created in 1956 by Joseph Galway. The value is calculated at the specified layer by subtracting the temperature of the parcel from the temperature of the atmosphere at that level. Table 3 shows typical values of LI. In terms of instability, values of 0 to -2 indicate weak instability, -3 to -5 is moderate instability, -6 to -9 is strong instability and -10 or less is extreme instability [15]. This is included just so the study has a second opinion on the stability of the atmosphere.

Table 4. Typical Values of SRH from Vasquez [15]

Value	Description
150-299	Weak tornado potential
300-499	Moderate tornado potential
450+	Strong tornado potential

Storm Relative Helicity (SRH) is a measure of potential for cyclonic updraft rotation in right moving supercells [16]. The main caveat of this measure is it needs to know the correct storm motion and is sensitive to the wind field. If the storm is left moving, this parameter is not accurate [15]. However, it can still be used as a tool if the caveats are known. As this study is specifically looking at tornado prediction, and this parameter measures rotation in a storm that can be an indicator of tornadic activity, it is included in the study. The units of this measure are meters squared per second squared. Table 4 shows typical values for this parameter.

Potential temperature is the temperature of a parcel of air when it is brought dry adiabatically to 1000mb [15]. This can show if the atmosphere is stable if the value increases with height [17]. This gives the study another look at the stability of the atmosphere.

Geopotential height is how high in the atmosphere you must go in order to reach a specified pressure. Paul Sirvatka at College of DuPage has a saying, "The thicker the warmer the thinner the cooler" [18]. This means the higher up a pressure is the warmer the atmosphere. As storms generally are warm season activity, especially tornadoes, using geopotential height is a first indicator at how warm the atmosphere is.

Out of all the variables in the NARR, the ones listed were what the author thought were most relevant to the study. Additional variables could be calculated that relate to convective weather, but the author did not have time to do so.

5.3 Data Pre-Processing Methodology

As the tornado dataset from SPC is for all 50 states, it is necessary to extract data that falls within the bounding box selected for this study and extract years 1979 through 2013 as the dataset has data starting in 1950 and ending 2018. The Pandas [30] library in Python was used for the first step of restricting the years. A mask was created and applied to get only tornadoes from January 1, 1979 through December 31, 2013. To restrict to the defined bounding box, GeoPandas [31] library was used. A GeoDataFrame object was created from the Pandas dataframe. From there, from the Shapely [32] library, the Polygon geometry function was imported. This created a polygon of the box using its four corners as input. GeoPandas has a geometry within function that created a GeoDataFrame of points only within the boundary box. Figure 2, next page, shows the result of this pre-processing.

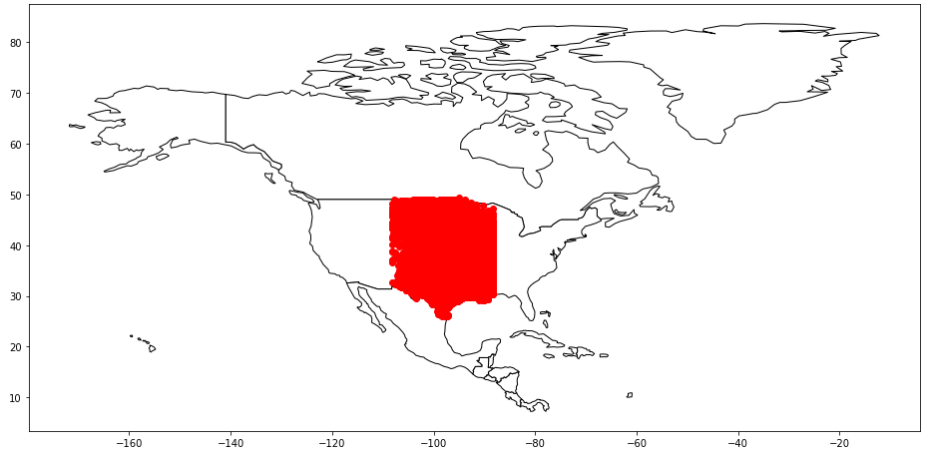

Fig. 2. Bounding box showing region used in this study. While not able to be seen, there is a red dot for each tornado.

By using Pandas groupby and size functions, a series was created that held each unique date and the number of tornadoes for that date. In this series, each date was an index object, and as such a reindex call was made to fill in all dates that had no tornadoes and initialed them to zero. This allowed the series to hold not just dates with tornadoes, but all dates within the time frame.

A Python script was created to download the 12Z and 00Z files from National Centers for Environmental Prediction (NCEP) NARR. An Excel book was created that dynamically created all URLs of the files that were then loaded into the Python script via text files. From here, the Grib file was ran through a Perl script called grib2ctl maintained by Wesley Ebisuzaki at Climate Prediction Center (CPC) [19]. The CTL file created by this script is necessary for OpenGrADS [20]. OpenGrADS is a modified version of Grid Analysis and Display System (GrADS). For simplicity, GrADS will be used to refer to OpenGrADS. To use the Grib and CTL file within GrADS, the CTL file had to be ran through gribmap script, which is supplied by GrADS.

A GrADS scripting language script was created to then extract the variables needed for this study. GrADS has a function called *aave* which takes an areal average over a region. Here the region was the bounding box specified. From here, a text file was created for each 12Z and 00Z file for each day in the study.

As the study used binary classification as a starting point, it was necessary to take the tornado counts and convert into a 0 or 1. 0 was set for days with 4 or less tornadoes while 1 was set for days with five or more tornadoes. This was all done using the Pandas library and a DataFrame.

5.4 Neural Network Setup

As already stated, due to the short time frame, only a neural network was used in the study. In order to get consistent results, it was necessary to seed the models. An environmental seed was set, a seed using the random library was set, and a seed within the TensorFlow [33] library was set as well. Setting these three seeds made the models consistent.

After loading in the tornado data into a Pandas DataFrame for the input variables and a Pandas Series for the binary representation of tornado counts, a training and testing split was performed. The scikit-learn [34] Python Library has a function that does this automatically. The test size was set to be 30% of the dataset.

To run the networks, the Python library TensorFlow was used. TensorFlow has built in functions that allow the quick development of machine learning algorithms. This assisted in speeding up the model generating process of this study.

6 Results

A total of five different models were generated for both 12Z and 00Z data. This was done to run different optimizers and tune their parameters to try to get the best model as possible. The models that were not deemed the best will be briefly looked at before looking into the model that had the best overall results.

All five models had the same network layout. Each started with an input layer, followed by a fully connected hidden layer, then a dropout layer and finally an output layer. The input and hidden layer both had 26 neurons, and they also had the ReLU activation function. The initial layer had an initial uniform weight vector. Each layer also had a regularization term that is a combination of L1 and L2 regularizers. The output layer had a sigmoid activation. Each model had 10 epochs and batch size of 6,500. An epoch is a full pass through the training set and a batch size is how many training examples are in a split of the dataset to allow faster computation [21].

Rectified Linear Unit (ReLU) has been shown to have a good foundation in mathematics and has been shown to improve neural networks [22]. Its formula is simple and is shown in equation 1.

$$f(x) = max(0,x) \tag{1}$$

The equation of ReLU means that the value of the function will be 0 if x is less than 0 and will output a linear function when x is greater than or equal to 0 [22]. ReLU has been shown to have faster learning over other activation functions such as logistic or tanh [23].

Regularization is an attempt to control the complexity of the model. This term is added to the loss function. The theory behind regularization is that when the model minimizes the function it will try to make the loss and complexity term smaller [24].

The dropout layer is an attempt to prevent overfitting of the model. Within the dropout layer, the model randomly chooses a neuron, or neurons to temporarily not include in the next layer [25].

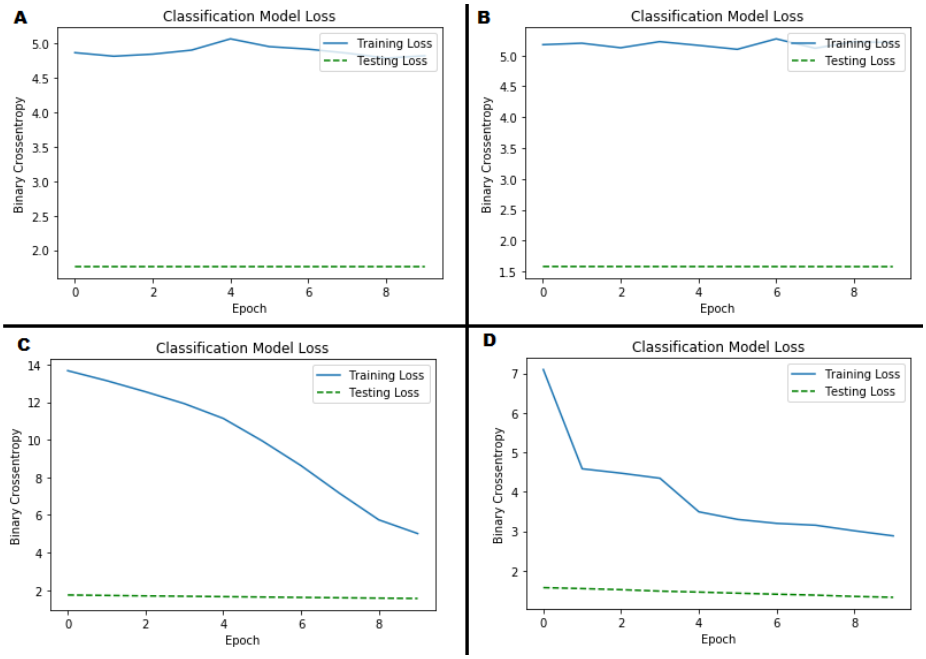

Fig. 3. A is loss function from Adadelta optimizer, B from Adagrad, C from Adam and D from NAdam. Testing label is labeled wrong, should read validation.

6.1 12Z Neural Network Models

The first unsuccessful model used the Adadelta optimizer with default values of learning rate at 0.001, decay rate of 0.95 and a constant 1 x 10^{-7}. In figure 3A, the training loss is much higher than the validation (testing) loss, and both are overall consistent. A sign of an underfit model is a high training and validation error, while an overfit model would have low training and high testing error [26]. Looking at Figure 3A, it appears that the model is underfit based on the indicators. It does not have any indicators as overfit. In addition, the training and validation losses never converge, and because of this and the underfit reason, this model is thrown out.

The next model used the Adagrad optimizer. The tensorflow default values for this optimizer are learning rate of 0.001, initial accumulator of 0.1 and a constant of 1 x 10^{-7}. In Figure 3B, the training loss is higher than the validation loss, and they are both consistent as well. Based on these losses, this model appears to be underfit as well. Adadelta is based off of this optimizer, so that could explain the similarities of the losses [27].

Table 5. Confusion Matrix of the model using Adam optimizer

	Predicted No	Predicted Yes
Actual No	1721	191
Actual Yes	529	116

The third model uses the Adam optimizer with the default values being learning rate of 0.001, first exponential decay rate of 0.9, the second exponential decay rate of 0.999 and a constant of 1 x 10^{-7}. Figure 3B shows the losses of this model. This model is already doing better than the first two, though it is underfit as well. The training loss is decreasing as it learns the data, but it never converges, at least with the 10 epochs used in this case. This model does get a testing accuracy of 71.8%. Table 5 shows the confusion matrix of this model. From the confusion matrix the following percentages can be calculated: error rate, recall, precision and F1 Score.

$$Error\ Rate = 1 - Accuracy \tag{2}$$

The *error rate* states how often the model incorrectly predicts the data. For instance, if it predicts a 1 instead of a yes, this goes towards the error rate. It is as simple as equation 2. This model has an error rate of 28.2%.

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

Equation 3 is *Recall*. Recall shows how many yes days were predicted correctly by the model. For this model, it has a recall value of 18%. Recall can also be calculated for no days, by changing the true positive in equation 3 to be true negative and the false negative to be false positive. In this case the recall for days with less than five tornadoes is 90%. This means the model can predict days without five or more tornadoes better than days with more than five tornadoes.

$$Precision = \frac{TP}{TP+FP} \tag{4}$$

Equation 4 is *Precision.* Precision is how many true days were predicted correctly out of all predicted true days. Precision can also be calculated for false days by switching true positive to true negative and false positive to false negative. The false precision is 76% while the true precision is 38%. This still shows this model is better at predicting no days versus yes days.

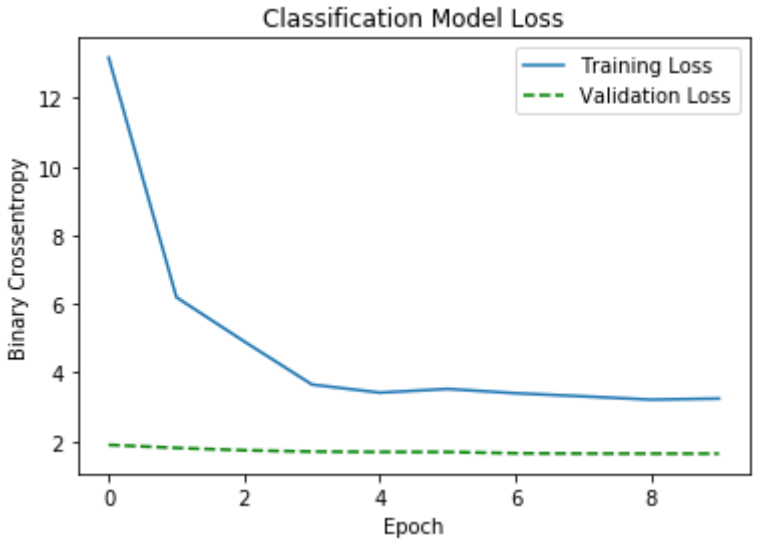

Fig. 4. Binary Cross-Entropy loss for Stochastic Gradient Descent Optimizer

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (5)$$

Equation 5 is *F1 Score*. F1 score tries to balance the precision and recall of the results. F1 score is useful when there is an uneven class distribution, which this dataset does indeed have [38]. Another way to think of F1 score is a weighted average of precision and recall [39]. When we calculate F1 score for false days we get 82.41% and for true days we get 24.43%. Even with this measure, we still see that the model is much better at predicting days with less than five tornadoes than those with five or more tornadoes.

The last model that was thrown out used the Nadam optimizer. Figure 3D shows the loss functions. In this case, both training and validation losses are decreasing, showing that the model is learning better, but they still do not converge in the epochs it went through. We can also claim that this model is underfit.

The model that this study found to be the most successful out of the five tested, used the Stochastic Gradient Descent Optimizer (SGD). TensorFlow uses default values for learning rate at 0.01 and momentum at 0.0. Upon tuning these two values, it was found that a learning rate of 0.001 and momentum of 0.5 gave the best results. As seen in Figure 4, both training and validation loss decrease over epochs with training having the best improvement. While the losses never converge within the epochs run, this is the closest any model has been to converging. Therefore, this model has been selected to be the best, even though it is also underfit.

Table 6. Confusion Matrix for Stochastic Gradient Decent Model

	Predicted No	Predicted Yes
Actual No	1596	141
Actual Yes	654	166

Table 6 shows the confusion matrix for this model. The accuracy of the testing dataset is 68.9%, not the best accuracy, but when looking at the other measures, is where the improvement is seen. Recall for negative days is at 92% while recall for positive days is at 20%. Precision for negative days is 71% while positive days is at 54%. Finally, the F1 score is 80.15% for negative days and 21.19% for positive days. This shows this model has improved on predicting the number of days with more than five tornadoes when looking at precision and recall, but the F1 score has slightly decreased on positive days. Some of the calculations have decreased slightly on the negative days, but since the goal is to predict tornadoes we would like to see improvement on those predictions. While the F1 score is decreased on positive days, it is the author's opinion that the small decrease isn't a major issue.

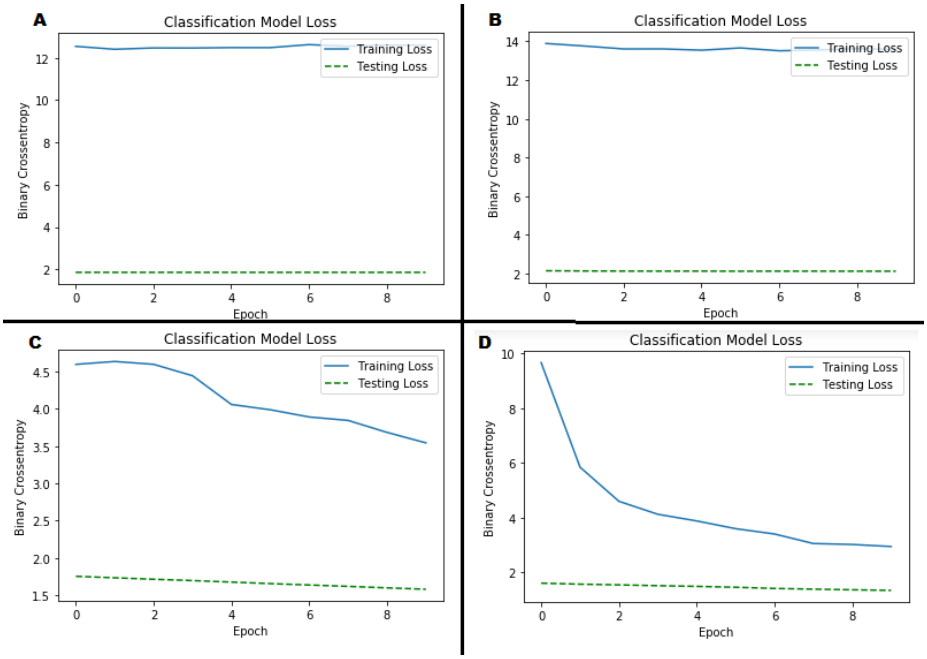

Fig. 5. A is loss function from Adadelta optimizer, B from Adagrad, C from Adam and D from NAdam. Testing once again should be labeled validation.

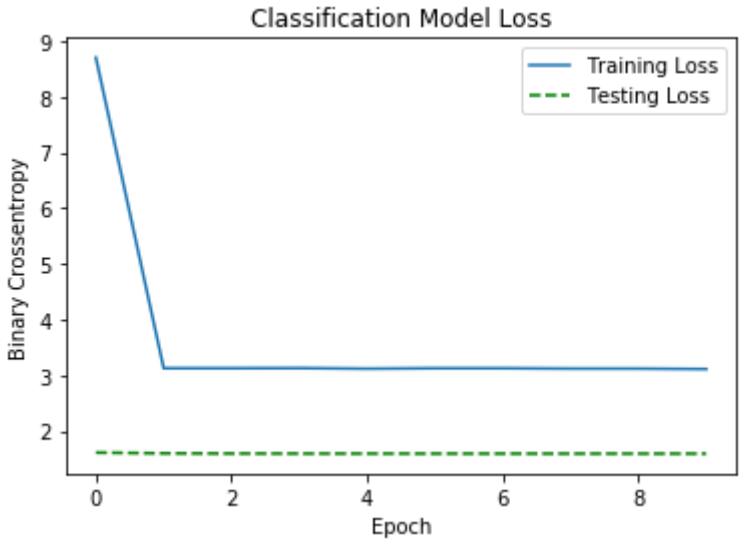

Fig. 6. SGD loss functions

Table 7. SGD Confusion Matrix

	Predicted No	Predicted Yes
Actual No	1903	255
Actual Yes	339	60

6.2 00Z Neural Network Models

For 00Z data, the same five neural networks were run. Generally, the losses seen in Figure 5 are similar to those in Figure 3. Once again, it appears all four models are underfit. The first three models were thrown out.

Since the SGD optimizer performed best for the 12Z data, it was decided to focus on this model specifically with 00Z data. For consistency, it is thought to be best to use the same neural network setup. With the other four models in mind, Nadam looks to be the best out of them for 00Z data. Therefore, both the SGD and NAdam models will be discussed further.

Figure 6 shows the loss functions for the network using SGD. In this instance, it appears that the network only needs to go through two epochs as that is the last place the training loss decreases. The validation loss never decreases in this network. This network is underfit as well. Table 7 shows the confusion matrix from which we can find the other measures to help describe the network.

Accuracy of this network overall is 76.77%. Precision for negative days is 85% while for positive it is 19%. Recall is 88% for negative days and 15% for positive days. F1 score is 86.47% for negative and 16.76% for positive. Compared to the 12Z

Table 8. Confusion Matrix of network using NAdam

	Predicted No	Predicted Yes
Actual No	2191	274
Actual Yes	51	41

network using the exact same layout, the accuracy increased overall, and precision, recall and F1 score increased for negative days but decreased on positive days.

With those results in mind, as well as seeing that the model only learns twice and never again, this model is thrown out for the 00Z data. This makes Nadam now look to be better, and this will be shown next.

When we look at Nadam and review Figure 5D, both the training and validation losses decrease, and they never converge, but they get close. Once again, the losses also show that the model is underfit. Table 8 shows the confusion matrix of this network. Here the precision for negative days is 98% and 13% for positive days. Recall is 89% for negative days and 45% for positive days. F1 score is 93.28% for negative days and 20.17% for positive days. The overall accuracy is 87.29%. Precision and recall increased for negative days while only recall increased for positive days. F1 score increased on negative days and decreased slightly for positive days. While not an improvement on this calculation for positive days, it is close enough to the 12Z data that it is not too bad overall.

7 Conclusions

There are issues with all 10 models utilized in this study. There are multiple reasons that could cause these issues. These include the expected limiting factors described, not enough days with more than 5 tornadoes in the dataset, and the possibility that neural networks are not the best machine learning algorithm for this problem.

Each network for 12Z and 00Z was underfit. This means the network was unable to learn enough from the dataset. The networks were able to learn and predict days with less than 5 tornadoes better than it could with days with five or more tornadoes. The author feels this is the biggest limiting factor of the study at this point. The other factors play a role, but those can be fixed by the author, while this one cannot.

There are some promising signs that there is use in using machine learning for predicting tornadoes. Both 12Z and 00Z data had a network or two where it seemed that the training and validation loss might converge either with more epochs or different hyperparameters. While the recall and precision for days with five or more tornadoes are low, the models do pick up on some of those days. This gives some promise as to the effectiveness of using machine learning to predict tornadoes.

8 Future Work

There are many areas of this problem that can be used for future work. The first is hyperparameter tuning. Given more time, more effort could be put into tuning the networks to find optimal learning rates, momentum and other parameters for the various optimizers. In addition, getting data for years 2014 through 2018 would add more data to the data set and increase the days with more than five tornadoes.

Another potential for further work would be to remove the weaker tornadoes from the dataset. This would help eliminate instances of localized conditions that may not be seen by the NARR. While this study used machine learning calculations from the confusion matrixes, adding meteorological calculations would be beneficial. Research on these calculations would need to be done in order to include them.

Using cross-validation is another research area to explore that could help improve the results with the current data set. Cross-validation is taking the data and splitting it into subsets where the model will train on one subset and use another to evaluate the performance of that model. Multiple rounds are then used with different subsets from the exact same data. The final step is that it will combine the validation results to estimate the model's predictive ability [36]. Specifically, k-folds would be examinced. K-folds is a method that will randomly divide the dataset into k groups (folds) of close to equal size. The first group is kept for testing and the model is then trained on *k-1* groups [37].

Additional future work could include trying other machine learning algorithms. Also using individual gridpoints instead of an average and using the full CONUS could help improve the models. One additional idea would be to include severe wind and hail in the dataset, not just tornadoes. Overall, there are many ways to improve this study that could produce a much better model. There is still hope for this problem in meteorology and machine learning.

9 Acknowledgements

The author would like to thank Lewis University Computer Science Department, and especially Dr. Jason Perry for granting access to the department's supercomputer and installing a different version of Perl for a program needed for this study to run. The author would also like to thank all the instructors he had for both his Bachelors Degree and Masters Degree from the Computer Science Department. Without them all, the author would not have been able to learn as much as he did or make it possible to figure out how to incorporate data science into meteorology. The author also thanks Dr. Vittorio (Victor) Gensini, Department of Geographic and Atmospheric Sciences at Northern Illinois University for his guidance and conversations that assisted the author in narrowing down ideas for this project and future projects.

References

1. Zhang, Z., Cayan, D.R., Pierce, D.W.: SUBSEASONAL TO SEASONAL TEMPERATURE PREDICITON SKILL OVER ..., https://www.energy.ca.gov/sites/default/files/2019-07/Projections_CCCA4-CEC-2018-010.pdf.
2. Cohen, J., Coumou, D., Hwang, J., Mackey, L., Orenstein, P., Totz, S., Tziperman, E.: S2S reboot: An argument for greater inclusion of machine learning in subseasonal to seasonal forecasts. Wiley Interdisciplinary Reviews: Climate Change. 10, (2018).
3. Hwang, J., Orenstein, P., Cohen, J., Pfeiffer, K., Mackey, L.: Improving Subseasonal Forecasting in the Western U.S. with Machine Learning. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD 19. 2325–2335 (2019).
4. Holmstrom, M., Liu, D., Vo, C.: Machine Learning Applied to Weather Forecasting. (2016).
5. Haberlie, A.M., Ashley, W.S.: A Method for Identifying Midlatitude Mesoscale Convective Systems in Radar Mosaics. Part I: Segmentation and Classification. Journal of Applied Meteorology and Climatology. 57, 1575–1598 (2018).
6. Liu, J.N.K., Hu, Y., You, J.J., Chan, P.W.: Deep Neural Network Based Feature Representation for Weather Forecasting. In: Deep Neural Network Based Feature Representation for Weather Forecasting. pp. 1–7 (2014).
7. Mcgovern, A., Elmore, K.L., Gagne, D.J., Haupt, S.E., Karstens, C.D., Lagerquist, R., Smith, T., Williams, J.K.: Using Artificial Intelligence to Improve Real-Time Decision-Making for High-Impact Weather. Bulletin of the American Meteorological Society. 98, 2073–2090 (2017).
8. Ghosh, T., Krishnamurti, T.N.: Improvements in Hurricane Intensity Forecasts from a Multimodel Superensemble Utilizing a Generalized Neural Network Technique. Weather and Forecasting. 33, 873–885 (2018).
9. Verbeten, E.: Machine learning and its radical application to severe weather prediction, https://news.wisc.edu/machine-learning-and-its-radical-application-to-severe-weather-prediction/.
10. Liu, Y., Racah, E., Correa, J., Khosrowshahi, A., Lavers, D., Kunkel, K., Wehner, M., Collins, W.: Application of Deep Convolutional Neural Networks for Detecting Extreme Weather in Climate Datasets. In: Application of Deep Convolutional Neural Networks for Detecting Extreme Weather in Climate Datasets. , San Francisco.
11. Markowski, P., Richardson, Y.P.: Mesoscale meteorology in midlatitudes. Wiley-Blackwell, Chichester (2010).
12. North American Regional Reanalysis (NARR), https://www.ncdc.noaa.gov/data-access/model-data/model-datasets/north-american-regional-reanalysis-narr.
13. North American Regional Reanalysis (NARR), https://nomads.ncdc.noaa.gov/data/narr/.
14. Storm Prediction Center WCM Page, https://www.spc.noaa.gov/wcm/#data.
15. Vasquez, T.: Weather forecasting red book: forecasting techniques for meteorology. Weather Graphics Technologies, Garland, TX (2009).
16. Numerical Model Data, https://weather.cod.edu/forecast/.
17. Annane, B.: (2019), ftp.aoml.noaa.gov/pub/hrd/annane/prelim_notes/Potential_Temperature.pdf.
18. Sirvatka, P.: .
19. https://www.cpc.ncep.noaa.gov/products/wesley/grib2ctl.html.
20. OpenGrADS, http://opengrads.org/.
21. Unsupervised Feature Learning and Deep Learning Tutorial, http://deeplearning.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/.

22. Agarap, A.F.: Deep Learning using Rectified Linear Units (ReLU).
23. Gordon, S.: PDF, https://athena.ecs.csus.edu/~gordonvs/180/WeeklyNotes/11A_DeepLearning.pdf.
24. Bhaskara, A., Roy, C.D.: http://www.cs.utah.edu/~bhaskara/courses/theoryml/scribes/lecture19.pdf.
25. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research. 15, 1929–1958 (2014).
26. Koehrsen, W.: Overfitting vs. Underfitting: A Complete Example, https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765.
27. Zeiler, M.D.: ADADELTA: AN ADAPTIVE LEARNING RATE METHOD.
28. Atmospheric Reanalysis Data, https://rda.ucar.edu/.
29. NOAA/NWS Storm Prediction Center, https://www.spc.noaa.gov/.
30. Python Data Analysis Library, https://pandas.pydata.org/.
31. GeoPandas 0.6.0, http://geopandas.org/.
32. Shapely, https://shapely.readthedocs.io/en/latest/.
33. TensorFlow, https://www.tensorflow.org/.
34. scikit-learn, https://scikit-learn.org/stable/.
35. Strader, S.M., Ashley, W.S.: The Expanding Bulls-Eye Effect. Weatherwise. 68, 23–29 (2015).
36. Khandelwal, R.: K fold and other cross-validation techniques, https://medium.com/datadriveninvestor/k-fold-and-other-cross-validation-techniques-6c03a2563f1e.
37. Khandelwal, R.: K-fold and Dropout in Artificial Neural network, https://medium.com/datadriveninvestor/k-fold-and-dropout-in-artificial-neural-network-ea054a89fb4e.
38. Shung, K.P.: Accuracy, Precision, Recall or F1?, https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.Scikit learn
39. sklearn.metrics.f1_score, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

Appendix A – Download NARR from NCEP

Also available at: https://github.com/sburgholzer/MSDS-Capstone-Project/blob/master/implementation/ncepDownload.py

```
import requests, os, sys, shutil
from multiprocessing.pool import ThreadPool

year = sys.argv[1]
dataDir = "/mnt/data3/scott/" + year + "data"
textDir = "/mnt/data3/scott/textfiles"
scriptDir = "/mnt/data3/scott/scripts"

def check_file_status(filepath, filesize):
    sys.stdout.write('\r')
    sys.stdout.flush()
    size = int(os.stat(filepath).st_size)
    percent_complete = (size/filesize)*100
   sys.stdout.write('%.3f %s' % (percent_complete, '% Completed'))
    sys.stdout.flush()

def download_url(url):
    os.chdir(dataDir)
    print("\ndownloading: ", url)
    file_name_start_pos = url.rfind("/") + 1
    file_name = url[file_name_start_pos:]

    r = requests.get(url, allow_redirects=True, stream=True)
    filesize = int(r.headers['Content-length'])
    if r.status_code == requests.codes.ok:
        with open(file_name, 'wb') as f:
            chunk_size=1048576
            for chunk in r.iter_content(chunk_size=chunk_size):
              f.write(chunk)
              if chunk_size < filesize:
                check_file_status(file_name, filesize)
        check_file_status(file_name, filesize)
            #for data in r:
            #    f.write(data)
    return url

def extractFromGrib(dataDir, scriptDir):
    os.chdir(dataDir)
    print('\nExtracting from Grib')
    gribFiles = [f for f in os.listdir(dataDir)]
    for gribFile in gribFiles:
```

```
        # Create the CTL file
        os.system('perl ' + scriptDir + '/grib2ctl.pl -verf ' + dataDir + '/' +
            gribFile + '>' + gribFile + '.ctl')
        # Create the IDX file
        os.system('gribmap -q -v -i ' + gribFile + '.ctl')
        # run the GrADS script
        os.system('grads -bxcl "run ' + scriptDir +
              '/arealAverages.gs ' + gribFile + '.ctl ' + year + '"')

# make the new textfile directory
if not os.path.isdir(textDir + "/" + year):
  os.mkdir(textDir + "/" + year)

# make the new data directory
if not os.path.isdir(dataDir):
  os.mkdir(dataDir)

with open('/mnt/data3/scott/scripts/' + year + 'files.txt', 'r') as file:
  tempfiles = file.read().replace('\n','').replace('\t', '')

filelist = tempfiles.split(",")
while("" in filelist) :
    filelist.remove("")

#print(filelist)

for myfile in filelist:
    download_url(myfile)


extractFromGrib(dataDir, scriptDir)

# remove the data folder
#shutil.rmtree(dataDir)

os.remove(scriptDir + "/" + year + "files.txt")
```

Appendix B – arealAverages.gs

Also available at: https://github.com/sburgholzer/MSDS-Capstone-Project/blob/master/implementation/gradsScripts/arealAverages.gs

```
* get the passed in filename
function createTxtFile(args)
 filename=subwrd(args,1)

* open the file
'open /media/sf_D_DRIVE/MSDS-Capstone-Project/notebooks/'
filename

* set time to the 12Z hour
'set t 1'

* query the time so we can appropriately name the text
file
'q time'
datestr=subwrd(result,3)
vyear = substr(datestr,9,4)
vmonth = substr(datestr,6,3)
vday = substr(datestr,4,2)
vhour = substr(datestr,1,2)
newdatestr = vmonth '-' vday '-' vyear '-' vhour 'Z'

* textfile name
textfile='/media/sf_D_DRIVE/MSDS-Capstone-
Project/notebooks/textfiles/' newdatestr '.txt'

* set up the text file headers
dummy=write(textfile, 'VAR,VAL,UNITS')

* GET THE DATA

* 2M Temp
't = aave(tmp2m, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'2mTemp,' tmp ',Kelvin',append)

* 2M Dewpoint
't = aave(dpt2m, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'2mDpt,' tmp ',Kelvin',append)
```

```
* CAPE SFC
't = aave(capesfc, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'sfcCAPE,' tmp ',J/kg',append)

* CIN SFC
't = aave(cinsfc, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'sfcCIN,' tmp ',J/kg',append)

* Potential Temp SFC
't = aave(potsfc, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'sfcPotentialTmp,' tmp ',Kelvin',append)

* SFC Pressure
't = aave(pressfc, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'sfcPressure,' tmp ',Pa',append)

* PWAT
't = aave(PWATclm, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'PWAT,' tmp ',kg/m^2',append)

* RH 2M
't = aave(rh2m, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'2mRH,' tmp ',%',append)

* Storm Relative Helicity
't = aave(hlcy0_3000m, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'srh,' tmp ',m^2/s^2',append)

* Lifted Index
't = aave(lftx500_1000mb, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
```

```
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'lifted index (500-100mb),' tmp ',Kelvin',append)

* lifted index 2
't = aave(no4LFTX180_0mb, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'lifted index 180-0 mb above gnd
Best (4-layer),' tmp ',Kelvin',append)

* 10 m wind
't = aave(mag(ugrd10m, vgrd10m), lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'10m wind speed,' tmp ',m/s',append)


* 250 MB
'set lev 250'

* Geopotenial height
't = aave(hgtprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'250mb Geopotenial height,' tmp ',gpm',append)

* Temperature
't = aave(tmpprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'250mb Temperature,' tmp ',Kelvin',append)

* wind
't = aave(mag(ugrdprs, vgrdprs), lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'250mb wind speed,' tmp ',m/s',append)

* 500 MB
'set lev 500'

* Geopotenial height
't = aave(hgtprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
```

```
tmp=subwrd(result,4)
dummy=write(textfile,'500mb Geopotenial height,' tmp ',gpm',append)

* Temperature
't = aave(tmpprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'500mb Temperature,' tmp ',Kelvin',append)

* wind
't = aave(mag(ugrdprs, vgrdprs), lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'500mb wind speed,' tmp ',m/s',append)



* 700 MB
'set lev 700'

* Geopotenial height
't = aave(hgtprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'700mb Geopotenial height,' tmp ',gpm',append)

* Temperature
't = aave(tmpprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'700mb Temperature,' tmp ',Kelvin',append)

* wind
't = aave(mag(ugrdprs, vgrdprs), lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'700mb wind speed,' tmp ',m/s',append)


* 850 MB
'set lev 850'

* Geopotenial height
't = aave(hgtprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
```

```
tmp=subwrd(result,4)
dummy=write(textfile,'850mb Geopotenial height,' tmp ',gpm',append)

* Temperature
't = aave(tmpprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'850mb Temperature,' tmp ',Kelvin',append)

* wind
't = aave(mag(ugrdprs, vgrdprs), lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'850mb wind speed,' tmp ',m/s',append)


* 925 MB
'set lev 925'

* Geopotenial height
't = aave(hgtprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'925mb Geopotenial height,' tmp ',gpm',append)

* Temperature
't = aave(tmpprs, lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'925mb Temperature,' tmp ',Kelvin',append)

* wind
't = aave(mag(ugrdprs, vgrdprs), lon=-108.265141, lon=-88.213196, lat=25.60345, lat=49.332204)'
'd t'
tmp=subwrd(result,4)
dummy=write(textfile,'925mb wind speed,' tmp ',m/s',append)
```

Appendix C – Extract Tornadoes for box.ipynb

Also available at: https://github.com/sburgholzer/MSDS-Capstone-Project/blob/master/implementation/Extract%20Tornadoes%20for%20box.ipynb

```
%matplotlib inline import pandas as pd import io, requests, os, os.path import geopandas import matplotlib.pyplot as plt


# get the tornado data from the SPC

s = requests.get('https://www.spc.noaa.gov/wcm/data/1950-2018_actual_tornadoes.csv').content
df = pd.read_csv(io.StringIO(s.decode('utf-8')))

# convert the date column to date time and then restrict to the years we care about
df['date'] = pd.to_datetime(df['date'])
mask = (df['date'] >= '1979-1-1') & (df['date'] <= '2013-12-31')
df = df.loc[mask]

gdf = geopandas.GeoDataFrame(
    df, geometry=geopandas.points_from_xy(df.slon, df.slat))
```

```
from pylab import rcParams
rcParams['figure.figsize'] = 5, 10
world = geopandas.read_file(geopandas.datasets.get_path('naturalearth_lowres'))

# We restrict to South America.
ax = world[world.continent == 'North America'].plot(
    color='white', edgecolor='black')

# We can now plot our ``GeoDataFrame``.
gdf.plot(ax=ax, color='red')

plt.show()

data = {'Lat':[49.332204, 25.60345, 25.60345, 49.332204,
            49.332204], 'Lon':[-108.265141, -108.265141,
            -88.213196, -88.213196, -108.265141]}
polydf = pd.DataFrame(data)
polydf.head()

gdf2 = geopandas.GeoDataFrame(
   polydf, geometry=geopandas.points_from_xy(polydf.Lon, polydf.Lat))
print(gdf2.head())
```

```
world = geopandas.read_file(geopandas.datasets.get_path('naturalearth_lowres'))
plt.rcParams["figure.figsize"] = [16,9]
# We restrict to South America.
ax = world[world.continent == 'North America'].plot(
    color='white', edgecolor='black')

# We can now plot our ``GeoDataFrame``.
gdf2.plot(ax=ax, color='red')

plt.show()


from shapely.geometry import Polygon
polygon_geom = Polygon(zip(gdf2['Lon'], gdf2['Lat']))


within_boundary = gdf[gdf.geometry.within(polygon_geom
)]

world = geopandas.read_file(geopandas.datasets.get_path('naturalearth_lowres'))
plt.rcParams["figure.figsize"] = [16,20]
# We restrict to South America.
ax = world[world.continent == 'North America'].plot(
    color='white', edgecolor='black')

# We can now plot our ``GeoDataFrame``.
within_boundary.plot(ax=ax, color='red')

plt.show()

tor_counts = within_boundary.groupby('date').size()

idx = pd.date_range('01-01-1979', '12-31-2013')
tor_counts.index = pd.DatetimeIndex(tor_counts.index)
tor_counts = tor_counts.reindex(idx, fill_value=0)

torCountsDF = pd.DataFrame({'date':tor_counts.index,
    'count':tor_counts.values})
```

```
torCountsDF['date'] = pd.to_datetime(torCountsDF.date)
```

```
torCountsDF.sort_values(by='date', inplace=True)

torCountsDF.to_csv(r'D:\MSDS-Capstone-Project\implementation\tornadoCounts.csv')
```

Appendix D – 12Z Model – Randomize the data.ipynb

Also available at: https://github.com/sburgholzer/MSDS-Capstone-Project/blob/master/implementation/12Z%20Model%20-%20Randomize%20the%20data.ipynb

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling
2D, Dropout, Flatten, Dense

import numpy as np
import os
import sys

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import IPython
from six.moves import urllib

print(tf.__version__)
```

```
data = pd.read_csv(r'D:\MSDS-Capstone-Project\implementation\Data\12Z.csv')
data['Date'] = pd.to_datetime(data['Date'])
data = data.sort_values(by='Date')
data.set_index('Date', inplace=True)
del data['lifted index (500-100mb)']
```

```
tor_count = pd.read_csv(r'D:\MSDS-Capstone-Project\implementation\Data\tornadoCounts.csv',
   index_col=1)
tor_count = tor_count.loc[:, ~tor_count.columns.str.co
ntains('^Unnamed')]
tor_count.index = pd.to_datetime(tor_count.index)
```

```
tor_count.rename(columns={'count':'COUNT'}, inplace=True)
tor_count.loc[tor_count.COUNT < 5, 'countBinary'] = 0.
tor_count.loc[tor_count.COUNT >= 5, 'countBinary'] = 1.
tor_count.countBinary = tor_count.countBinary.astype(int)
```

```
merge = pd.merge(data,tor_count, how='inner',
     left_index=True, right_index=True)
```

```
merge = merge.sample(frac=1)
```

```
merge.to_csv(r'D:\MSDS-Capstone-Project\implementation\Data\12Z-randomized.csv')
```

Appendix E – 00Z Model – Randomize the data.ipynb

Also available at: https://github.com/sburgholzer/MSDS-Capstone-Project/blob/master/implementation/00Z%20Model%20-%20Randomize%20the%20data.ipynb

```
import tensorflow as tf from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPool-
ing2D, Dropout, Flatten, Dense import numpy as np im-
port os import sys import matplotlib.pyplot as plt im-
port pandas as pd import seaborn as sns import IPython
from six.moves import urllib print(tf.__version__)
```

```
data = pd.read_csv(r'D:\MSDS-Capstone-Project\implementation\Data\00Z.csv')
data['Date'] = pd.to_datetime(data['Date'])
data = data.sort_values(by='Date')
data.set_index('Date', inplace=True)
del data['lifted index (500-100mb)']
data.head()
```

```
data.index = data.index.shift(-1, freq='d') # convert
from UTC day.. 00Z is next day, but for US it is still
same day. So subtract 1 day from it
```

```
data = data.drop(data.index[0])
```

```
tor_count = pd.read_csv(r'D:\MSDS-Capstone-Project\implementation\Data\tornadoCounts.csv',
      index_col=1)
tor_count = tor_count.loc[:, ~tor_count.columns.str.contains('^Unnamed')]
tor_count.index = pd.to_datetime(tor_count.index)

tor_count.rename(columns={'count':'COUNT'}, inplace=True)
tor_count.loc[tor_count.COUNT < 5, 'countBinary'] = 0.
tor_count.loc[tor_count.COUNT >= 5, 'countBinary'] = 1.
tor_count.countBinary = tor_count.countBinary.astype(int)
merge = pd.merge(data,tor_count, how='inner',
     left_index=True, right_index=True)
merge = merge.sample(frac=1)
```

```
merge.to_csv(r'D:\MSDS-Capstone-Project\implementation\Data\00Z-randomized.csv')
```

Appendix F – 12Z Model.ipynb

Also available at: https://github.com/sburgholzer/MSDS-Capstone-Project/blob/master/implementation/12Z%20Model.ipynb

```
import os
os.environ['PYTHONHASHSEED']=str(0)

import random
random.seed(0)

from numpy.random import seed
seed(0)

import tensorflow as tf

print(tf.__version__)
from tensorflow import keras
tf.random.set_seed(0)

from tensorflow.keras.wrappers.scikit_learn import KerasC
lassifier
from tensorflow.keras.regularizers import l2, l1, l1_l2


import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
#tf.keras.wrappers.skikit_learn.KerasClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report,
    accuracy_score, confusion_matrix
```

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data = pd.read_csv(r'D:\MSDS-Capstone-Project\implementation\Data\12Z-randomized.csv')
data.rename( columns={'Unnamed: 0':'Date'}, inplace=True )
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
data = data.sample(frac=1)
data.loc[data.COUNT < 5, 'countBinary2'] = 0
data.loc[data.COUNT >= 5, 'countBinary2'] = 1
```

```
# creating input featues and target variables
X = data.iloc[:,0:26]
y = data.iloc[:,28]
```

```
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.2, random_state=0)
X_train, X_val, y_train, y_val =
    train_test_split(X_train, y_train, test_size=0.25, random_state=0)
```

```
scaler = StandardScaler().fit(X_train)
X_Train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_val = scaler.transform(X_val)
```

```
def plot_history(hist):
    fig = plt.figure(figsize=(6,4))
    # # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'], 'g--')
    plt.title('Classification Model Loss')
    plt.ylabel('Binary Crossentropy')
    plt.xlabel('Epoch')
    plt.legend(['Training Loss', 'Validation Loss'],
           loc='upper right')
    print ("Loss after final iteration: ",
         history.history['val_loss'][-1])
    plt.show()
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
      activation='relu',
      kernel_initializer=
      keras.initializers.he_uniform(seed=0),
      kernel_regularizer=l1_l2(0.001),
      input_shape=(26,)))
model.add(keras.layers.Dense(units=26,
      activation='relu',
      kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
      activation='sigmoid',
      kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
      optimizer=keras.optimizers.SGD(
      learning_rate = 0.001, momentum=0.5),
      metrics=['accuracy'])

history = model.fit(X_train, y_train,
      validation_data=(X_val, y_val), epochs = 10,
      batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
```

```
model.evaluate(X_test, y_test, verbose=0)
```

```
plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))


predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'

print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
       confusion_matrix(predictions,y_test))
```

```
print(classification_report(predictions, y_test))
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
       activation='relu',
       kernel_initializer=keras.initializers.he_uniform(
       seed=0), kernel_regularizer=l1_l2(0.001),
       input_shape=(26,)))
model.add(keras.layers.Dense(units=26,
       activation='relu',
       kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
       activation='sigmoid',
       kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
       optimizer=keras.optimizers.Adadelta(),
       metrics=['accuracy'])

history = model.fit(X_train, y_train,
       validation_data=(X_val, y_val), epochs = 10,
       batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
model.evaluate(X_test, y_test, verbose=0)
```

```
plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))
predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'

print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
       confusion_matrix(predictions,y_test))

print(classification_report(predictions, y_test))
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_initializer=keras.initializers.he_uniform(
        seed=0), kernel_regularizer=l1_l2(0.001),
        input_shape=(26,)))
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
        activation='sigmoid',
        kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
        optimizer=keras.optimizers.Adagrad(),
        metrics=['accuracy'])

history = model.fit(X_train, y_train,
        validation_data=(X_val, y_val), epochs = 10,
        batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
```

```
model.evaluate(X_test, y_test, verbose=0)

plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))
predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'

print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
        confusion_matrix(predictions,y_test))

print(classification_report(predictions, y_test))
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
```

```
        activation='relu',
        kernel_initializer=keras.initializers.he_uniform(
        seed=0), kernel_regularizer=l1_l2(0.001),
         input_shape=(26,)))
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
        activation='sigmoid',
        kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
        optimizer=keras.optimizers.Adam(
        learning_rate=0.001), metrics=['accuracy'])



history = model.fit(X_train, y_train,
        validation_data=(X_val, y_val), epochs = 10,
        batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
```

```
model.evaluate(X_test, y_test, verbose=0)
```

```
plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))

#predictions

predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'

print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
        confusion_matrix(predictions,y_test))

print(classification_report(predictions, y_test))
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_initializer=keras.initializers.he_uniform(
               seed=0),_regularizer=l1_l2(0.001), input
        _shape=(26,)))
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
        activation='sigmoid',
        kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
        optimizer=keras.optimizers.Nadam(
        learning_rate=0.003), metrics=['accuracy'])



history = model.fit(X_train, y_train,
        validation_data=(X_val, y_val), epochs = 10,
        batch_size=6500)

eval_model=model.evaluate(X_train, y_train)
print(eval_model)

model.evaluate(X_test, y_test, verbose=0)

plot_history(history)


#predictions

predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'

print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
        confusion_matrix(predictions,y_test))

print(classification_report(predictions, y_test))
```

Appendix G – 00Z Model.ipynb

Also available at: https://github.com/sburgholzer/MSDS-Capstone-Project/blob/master/implementation/00Z%20Model.ipynb

```
import os
os.environ['PYTHONHASHSEED']=str(0)

import random
random.seed(0)

from numpy.random import seed
seed(0)

import tensorflow as tf

print(tf.__version__)
from tensorflow import keras
tf.random.set_seed(0)

from tensorflow.keras.wrappers.scikit_learn import
        KerasClassifier
from tensorflow.keras.regularizers import l2, l1, l1_l2


import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow import keras
#tf.keras.wrappers.skikit_learn.KerasClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report,
        accuracy_score, confusion_matrix
```

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
data = pd.read_csv(r'D:\MSDS-Capstone-Project\implementation\Data\00Z-randomized.csv')
data.rename( columns={'Unnamed: 0':'Date'}, inplace=True )
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
data = data.sample(frac=1)
data.loc[data.COUNT < 5, 'countBinary2'] = 0
data.loc[data.COUNT >= 5, 'countBinary2'] = 1
data
```

```
# creating input featues and target variables
X = data.iloc[:,0:26]
y = data.iloc[:,28]
```

```
X_train, X_test, y_train, y_test = train_test_split(
       X, y, test_size=0.2, random_state=0)
X_train, X_val, y_train, y_val =
       train_test_split(X_train, y_train,
       test_size=0.25, random_state=0)
```

```
scaler = StandardScaler().fit(X_train) X_Train =
  scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_val = scaler.transform(X_val)
```

```
def plot_history(hist):
    fig = plt.figure(figsize=(6,4))
    # # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'], 'g--')
    plt.title('Classification Model Loss')
    plt.ylabel('Binary Crossentropy')
    plt.xlabel('Epoch')
    plt.legend(['Training Loss', 'Testing Loss'],
       loc='upper right')
```

```
    print ("Loss after final iteration: ",
        history.history['val_loss'][-1])
    plt.show()
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_initializer=keras.initializers.he_uniform(
        seed=0), kernel_regularizer=l1_l2(0.001), input
_shape=(26,)))
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
        activation='sigmoid',
        kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
        optimizer=keras.optimizers.SGD(learning_rate =
        0.001, momentum=0.5),
       metrics=['accuracy'])

history = model.fit(X_train, y_train,
        validation_data=(X_val, y_val), epochs = 10,
        batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
```

```
model.evaluate(X_test, y_test, verbose=0)
```

```
plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))


predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'
```

```
print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
      confusion_matrix(predictions,y_test))

print(classification_report(predictions, y_test))
```

```
model.add(keras.layers.Dense(units=26,
      activation='relu',
      kernel_initializer=keras.initializers.he_uniform(
      seed=0), kernel_regularizer=l1_l2(0.001),
      input_shape=(26,)))
model.add(keras.layers.Dense(units=26,
      activation='relu',
      kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
      activation='sigmoid',
      kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
      optimizer=keras.optimizers.Adadelta(),
      metrics=['accuracy'])

history = model.fit(X_train, y_train,
      validation_data=(X_val, y_val), epochs = 10,
      batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
```

```
model.evaluate(X_test, y_test, verbose=0)
```

```
plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))
predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'

print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
```

```
        confusion_matrix(predictions,y_test))

print(classification_report(predictions, y_test))
```

```
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_initializer=keras.initializers.he_uniform(
        seed=0), kernel_regularizer=l1_l2(0.001),
        input_shape=(26,)))
model.add(keras.layers.Dense(units=26,
        activation='relu',
        kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
        activation='sigmoid',
        kernel_regularizer=l1_l2(0.001)))
  model.compile(loss='binary_crossentropy',
  optimizer=keras.optimizers.Adagrad(),
  metrics=['accuracy'])

history = model.fit(X_train, y_train,
        validation_data=(X_val, y_val), epochs = 10,
        batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
```

```
model.evaluate(X_test, y_test, verbose=0)
```

```
plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))
predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'

print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
        confusion_matrix(predictions,y_test))
```

```
print(classification_report(predictions, y_test))
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
       activation='relu',
       kernel_initializer=keras.initializers.he_uniform(
       seed=0),kernel_regularizer=l1_l2(0.001),
       input_shape=(26,)))
model.add(keras.layers.Dense(units=26,
       activation='relu',
       kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
       activation='sigmoid',
       kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
       optimizer=keras.optimizers.Adam(
       learning_rate=0.001), metrics=['accuracy'])



history = model.fit(X_train, y_train,
       validation_data=(X_val, y_val), epochs = 10,
       batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
```

```
model.evaluate(X_test, y_test, verbose=0)
```

```
plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))

#predictions

predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'

print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
```

```
       confusion_matrix(predictions,y_test))

print(classification_report(predictions, y_test))
model = keras.Sequential()
model.add(keras.layers.Dense(units=26,
     activation='relu',
     kernel_initializer=keras.initializers.he_uniform(
     seed=0), kernel_regularizer=l1_l2(0.001),
     input_shape=(26,)))
model.add(keras.layers.Dense(units=26,
       activation='relu',
       kernel_regularizer=l1_l2(0.001)))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.Dense(units=1,
        activation='sigmoid',
        kernel_regularizer=l1_l2(0.001)))
model.compile(loss='binary_crossentropy',
        optimizer=keras.optimizers.Nadam(learning_rate
=0.003), metrics=['accuracy'])



history = model.fit(X_train, y_train,
     validation_data=(X_val, y_val), epochs = 10,
     batch_size=6500)
```

```
eval_model=model.evaluate(X_train, y_train)
print(eval_model)
```

```
model.evaluate(X_test, y_test, verbose=0)
```

```
plot_history(history)
```

```
predictions = pd.DataFrame(model.predict(X_test))

#predictions

predictions[predictions > 0.5] = 1 #'5 or more Tornadoes'
predictions[predictions <= 0.5] = 0 #'Less than 5 Tornadoes'
```

```
print('accuracy', accuracy_score(predictions, y_test))
print('confusion matrix\n',
     confusion_matrix(predictions,y_test))

print(classification_report(predictions, y_test))
```