DIY Meteorology: The Search for a Tornado-Predicting Model for Everyday Users

Riley Carlin

Abstract

Tornadoes are somewhat infrequent weather phenomenon that can form and dissipate in a matter of minutes, making them difficult to collect data on (and, by extension, study and predict). However, they can also cause major damage to property and lives, making the ability to predict them a highly-valuable skill. In pursuit of this prediction method, I used cross validation to train a logistic regression model and multiple decision trees on easily-observable weather data from the Dallas-Fort Worth Metroplex. Unfortunately, no model performed particularly well at this task. However, this project did demonstrate the importance of knowing how to prepare data for machine learning models.

1. Introduction

This project seeks to develop an easy-to-use machine learning model to predict whether or not a tornado is nearby or likely to form given real-time meteorological readings, such as temperature, wind speed, and precipitation. In this case, "easy-to-use" specifically means a model that non-meteorologists and/or non-computer scientists can easily understand, access, and use. Not everyone has highly specialized weather data, nor does everyone have the technology or know-how to run sophisticated computer models on said data. A simple tornado-predicting tool would be beneficial in the hands of these everyday users as it could be a real-time, local tornado warning system in addition to local weather radio or broadcast (or it could be the sole warning if users do not have access to those professional warning methods); and in the case of a tornado, any advanced warning could save lives.

Furthermore, because this model is meant to be used by anyone, it must only rely on easily-obtained weather data– atmospheric temperatures, while helpful, are not always available in certain areas. And on the subject of data, a sub-goal I have for this project is to identify which weather variables (e.g., wind speed) and/or weather conditions (e.g., 40 MPH) are more conducive to the development of tornadoes.

A decision tree meets all of these goals; it is easy to interpret, can be used in real time (and without a computer), and can identify important variables and conditions. Thus, I developed two decision trees – a basic one and a more complex one – to take in weather data and predict whether or not a tornado is likely to be occurring. I also made a simple logistic regression model to serve as a baseline for the decision trees' results. Overall, the models did not perform terribly well; the accuracy and recall results often had large variances, making it difficult to determine the best model, if there was any. Additionally, no model managed to have its average accuracy and recall score both be above 65%. However, the experiments were not a complete loss, since they demonstrated that blindly replacing NaN values with medians is not always the best way to correct missing values, and addressing the tornado/ non-tornado data imbalance is key to having a recall score consistently above 20%.

2. Related Work

Researchers have found several ways of developing machine learning models to predict tornadoes. For example, this[1] paper from 2020 describes a convolutional neural network to predict imminent tornadoes by looking at radar images of storms (Lagerquist et al., 2020). While the authors of the paper were satisfied with their model's performance, they admitted that the model was fooled by tornadoes that didn't *look* like storms and non-tornado storms that *looked* like tornadoes (which is unsurprising if the model only has radar images to go off of). The authors also noted that sometimes performance was lackluster due to a lack of tornado examples, since tornadoes are relatively rare.

While my project and this 2020 research have similar goals, we approach it in vastly different ways. For one, the inputs are different; radar images may be helpful tools for identifying a tornado, but they are not necessarily easy to obtain. Would the average American be able to find and extract high-quality radar images and feed them into a model? And what about if the user doesn't have power, which is not unusual for a strong storm? In contrast, it is fairly easy to find or gauge weather variables like temperature, wind direction,

[1] *Deep Learning on Three-Dimensional Multiscale Data for Next-Hour Tornado Prediction*

and precipitation. However, both this past research and my own ran into the issue of having a severely limited number of tornado-positive examples.

Other radar-based projects, such as this[2] 2007 paper, use SVM to help improve performance. While a SVM could have improved the performance of my logistic regression model, I am not entirely sure they can be used *with* a decision tree, so I chose not to implement a SVM so that my experiments could be more comparable. Interestingly enough, this 2007 paper seems to use similar inputs to what I use! However, there doesn't seem to be the same desire to make the model non-expert friendly.

3. Dataset and Evaluation

The models will be reading in data from `tornadoes_FINAL.csv`, a file I scraped and calculated by hand from the National Oceanic and Atmospheric Administration (NOAA)'s "5-Minute Surface Weather Observations from the Automated Surface Observing Systems (ASOS) Network"(NOAA National Weather Service, U.S. Federal Aviation Administration, U.S. Department of Defense, NOAA National Centers for Environmental Information, 2023)[3]. As its name implies, this dataset stores weather readings at very small intervals, which is crucial when studying a weather phenomenon that typically lasts less than an hour. Every other dataset I found online either did not have my weather variables of interest, or were for time intervals that were much too wide to capture the difference between tornado-producing storms and non-tornado-producing storms. Thus, I decided to use the ASOS dataset.

3.1. Collecting the Data

I had to scrape the ASOS data by hand because each weather report is just a line in a large .txt file. Plus, many of the weather variables (which would later become columns in my `tornadoes_FINAL.csv`) are optional, not always delimited by spaces, and not always in the same order. Because of these inconsistencies, I decided it would be easier (albeit tedious) to read the files myself rather than try to have the computer scrape for me. While weather reports are taken on a 5-minute basis, I decided to only scrape the 10-minute intervals so I could have at least 5 storms by this final report (each storm takes a while to scrape).

I decided to only compile weather data from one geographic region (North Texas) and season (spring); that way, the model would be less likely to rely on irrelevant differences in weather readings caused by differences in the climates/ elevations/ ambient temperatures etc. of the locations/ time of year of the storms. To further avoid these kind of confounding variables, I narrowed my focus to a cluster of weather stations that are likely to experience the same storm. If the tornado and non-tornado entries are from the same storm, then differences between the readings may be more revealing than if they were from different weather events entirely. Also, instead of collecting data for an entire day, I only pulled data starting an hour before the first tornado touched down and ending an hour after the last tornado happened. I feel this range of times is limited enough to make scraping by hand feasible, while still including enough data points to get a sense of the nature of the storm.

As for *which* weather variables I collected, I took any variables that are known to effect the formation of tornadoes (such as wind speed and direction), as well as any variables that could realistically be correlated to tornadoes occurring (like cloud cover or hail). The readouts could get pretty confusing, so I relied heavily on the *Federal Meteorological Handbook No.1: Surface Weather Observations and Reports* (US Department of Commerce and National Oceanic and Atmospheric Administration, 2019) to interpret many of the variables and abbreviations. Unfortunately, there were some lone numbers on the weather reports that I could not interpret, so I did not include them as to avoid risking training the model on something totally irrelevant.

For your convenience, there is a chart of the weather variables I collected from the ASOS reports and a brief description of them following the conclusion. `tornadoes_FINAL.csv` also includes some non-weather variables like `TIME` and `STATION` for bookkeeping purposes[4], and some calculated variables to monitor *changes* in the weather.

3.2. Modifying the Data

Since these ASOS readings are simply 10 minute snapshots of the weather, I created three calculated columns, `WIND_DIR_CHANGE`, `WIND_SPEED_CHANGE`, and `TEMP_C_CHANGE` to track how these variables changed over time. For all of these columns, I simply took the the current time's value and subtracted the value from 10 minutes prior. There were some instances, however, where the `WIND_DIR` was 0 (no wind) or "variable (but weak)." For simplicity, I treated these directions as the same as the previous direction.

The last column I added was the `TORNADO` label, which would hold a 1 if there was a tornado nearby at the time of the weather report. This column serves as the prediction

[2] *Active Learning with Support Vector Machines for Tornado Prediction* (Trafalis et al., 2007)

[3] https://www.ncei.noaa.gov/metadata/geoportal/rest/metadata/item/gov.noaa.ncdc:C00418/html

[4] These columns are dropped before the data is given tot he models.

target for my models. Technically, most stations did not report experiencing a tornado even if there was one less than five miles away. So, for the purpose of this project, "nearby" means within 15 miles. I selected this distance because one station *did* report a tornado that was roughly 15 miles away. That particular ASOS report noted the existence of the tornado for 20 minutes (even though it was allegedly "very brief" (El Paso Times, 2023)). So, as a rule of thumb, I labeled the record immediately before and after a "touchdown" as also tornado-positive.

3.3. Filling in the Data

As mentioned previously, many variables of these weather variables are optional, meaning `tornadoes_FINAL.csv` has plenty of NaN values. While some models can handle missing data, logistic regression and basic decision trees can not. Thus, it's necessary to address these gaps.

A common practice is to fill missing values with the medians of their respective columns, so I did this for several of the experiments. However, this method does not always make sense for this particular dataset. For example, `SL_PRESSURE` is about 88% missing values, so how realistic can that median be? `PK_WIND_DIR` and `PK_WIND_KT` are in a similar situation, but have the added complication of the fact that they store data on the fastest wind *of the hour*, meaning they are sometimes out of the 30-minute tornado interval and thus a little out of date. Because of these issues, I left these three columns out of the dataset for several of the experiments.

Additionally, for the various sky condition heights, I inputted 300 (i.e., clouds at 30,000 feet) instead of simply the average cloud height. I did this because inputting the average value means that there were clouds at that elevation but the weather station chose not to record for some reason. These NaN values are not so much missing as deliberately left out. I chose 300 specifically because it was the highest cloud elevation I saw in the ASOS dataset (I didn't want to pick a number that was too high to be realistic) but was still much higher than any of the average elevations, and I believe clouds at 30,000 feet may be high enough to not be related to the storm occurring below them.

3.4. Storm Selection and Data Breakdown

As mentioned earlier, I wanted my data to be from the same region. I chose North Texas because I am familiar with the area, which it easier to find the weather stations in ASOS and their nearby tornadoes. ASOS had eight weather stations in the Dallas-Forth Worth Metroplex, as displayed in Figure 1.

Storms were selected by exploring *El Paso Times*' interactive map of tornadoes (El Paso Times, 2023).For each year since 2010 (too far back and global warming could become a confounding factor), I took note of when a tornado happened and which weather stations were within 15 miles of its path. From that set of dates, I chose spring days with multiple tornadoes to help generate a high tornado to non-tornado ratio (well... as high as possible, at least).

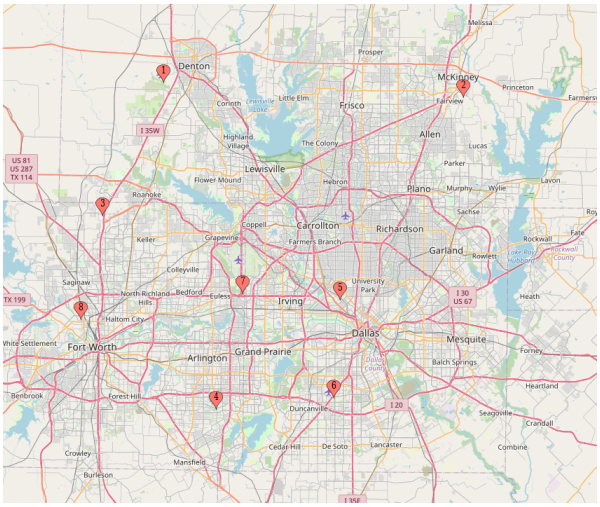

Figure 1. Map of Dallas Area Weather Stations in the ASOS Dataset. The area shown is roughly 70 miles wide and 55 miles high.

Because I have a limited amount of data – and because the data comes in nice bite-size (or rather storm-sized) pieces – I chose to use cross validation to train and test my models. In each iteration, one storm would be the test set and the remaining four would serve as the training set. A summary of these splits and the tornado/ non-tornado breakdown can be found in Figure 2.

3.5. Evaluation

Model performance will be evaluated based on accuracy and recall scores. Since this is ultimately a classification task, the accuracy score gives an overall idea of how "good" the model is. Recall is also important because when it comes to predicting something deadly like a tornado, false negatives can be catastrophic whereas false positives are just annoying. Because of the rarity of tornadoes, `tornadoes_FINAL.csv` has many more non-tornado entries than tornado entries. This disparity can make the accuracy seem too good to be true; if the model always predicts no tornado, it will be right almost 90% of the time. Thus, I will be relying more on recall to evaluate the models. *That being said*, the reverse can also happen, where the model may learn to always predict that a tornado is happening, which is also unproductive. Thus, it is important to not completely abandon accuracy in favor of a high recall score.

Storms Compiled					
	Storm 1	Storm 2	Storm 3	Storm 4	Storm 5
Date	05/24/2011	04/03/2012	03/29/2017	05/29/2019	03/21/2022
Tornadoes	5	8	3	3	4
Total Entries	205	219	110	111	160
Tornado Entries	22	41	16	16	20
Non-Tornado Entries	183	178	94	95	140
Tornado Percent	10.7	18.7	14.5	14.4	12.5

Figure 2. Summary of collected storm data.

4. Methods

For this project, I implemented two main types of models: logistic regression and decision trees (one basic and one complex). The former serves as a baseline to compare the performance of the trees. While these are two different methods of classification, the experiments I ran on them have several similarities. As mentioned earlier, I use a cross-validation method of training/ testing the models. I also experimented with various ways of addressing missing data with each type of model: ignore columns with missing, fill missing values with median values, fill or ignore missing based on problems mentioned in section 3.3. By the midterm report, I hadn't gotten the chance to do anything with the missing data other than ignore it, so I was excited about filling in the missing values so that the regression and basic decision tree had more options for variables to split on (I had hoped more options meant better splits).

Another change I made since the midterm is including experiments that limited the number of non-tornado examples in the train and test set. I simply counted the number of tornado examples in a set and then selected that many (random) non-tornado examples from the same set. This ended up improving recall across the board because the models couldn't cheat their way into a high accuracy score by always predicting non-tornado.

Ultimately, there were 36 sub-experiments, one for each combination of: 1) model (regression, basic tree, complex tree) 2) NaN fill technique (ignore, median fill, various fill[5]) 3) method for addressing the tornado/ non-tornado imbalance (limit or don't limit non-tornado examples). Now that I have gone over what the experiments were generally like, I will go into detail about each specific model!

4.1. Logistic Regression

The logistic regression model is the simplest of the three models, and will thus serve as a baseline. For this model, I used `sklearn.linear_model`'s `LogisticRegression`, which performs a regularized logistic regression. Essentially, this model will take in a dataframe of d weather variables (in this case, up to five), and try to fit a "straight" line in d-D space that separates the tornado examples from the non-tornado examples. When given a test weather report, it will predict based on which side of its fitted line the new report lands on.

I began by modifying the train and test sets as needed; this included addressing the missing values and limiting (or not limiting) the number of non-tornado examples. For these experiments, I never included the peak wind columns or `SL_PRESSURE` because I was worried that a simple model like logistic regression would perform badly when working with columns that were originally over half missing. If the experiment was a "median fill" test, then every other missing value was filled in my its column's median. If, however, the experiment was "various fill," then the `ALTIMETER` NaNs would be filled with the median while the "missing" sky condition elevations would be filled by the number 300.

For each of the five iterations (one for each test storm), I broke the training set (the other 4 storms) into a random 80/20 train_a/ dev split, with `stratify = TORNADO` to ensure the train_a and dev sets had the same proportion of tornado instances. I then used the `.corr()` function to find the five weather variables that were the most strongly correlated with the `TORNADO` label based on the train_a set. I then tested the model with the dev set to find how many input variables would result in the highest dev recall score. Once I knew how many input variables to include, I trained the regression model on the entire training set, with the n variables as input and `TORNADO` as the output. I then stored this final model's accuracy and recall scores so I could compare these metrics across the different test storms.

4.2. Decision Trees

As for the trees, I implemented a basic decision tree – `sklearn.tree`'s `DecisionTreeClassifier` – as well as a more complex one – `HistGradientBoostingClassifier` (HGBC). Both take in a dataframe of *all* weather variables[6] and attempt to separate the tornado examples from the non-

[5]To be explained in the following subsections

[6]Again, not including the bookkeeping variables.

tornado examples by continuously splitting the data based on a weather variable and condition/ level, until a leaf node is exclusively one class or the max depth has been reached. The basic decision tree decides the best split by minimising the gini impurity score, while the complex tree relies on ensembling to make its prediction accurate. HGBC is particularly suited for this project because it gracefully handles missing data during both training and testing.

Both trees went through the same experimental process. First I addressed the missing data and limited the non-tornado examples as necessary. For the basic decision tree, I either dropped columns with NaNs, filled in all missing with median values, or used "various fill." The HGBC had similar conditions, but since it can handle NaNs, I simply left the columns unchanged instead of dropping them if they had missing values. The "various fill" for the decision trees is almost exactly like the "various fill" for the logistic regression, except the peak wind columns and `SL_PRESSURE` are dropped here instead of for all decision tree experiments.

Once again, for each test storm, I broke the remaining training storms random 80/20 train_a/ dev split, with `stratify = TORNADO`. This time, the split was so I could pick the best `max_depth` for the model, which I chose based on whichever depth (up to 30, since that is the default for HGBC) had the best dev recall score. Once I had my `max_depth`, I fit the tree to entire train set and then recorded the accuracy and recall scores of the model on the test storm.

5. Experiments

A quick disclaimer before I discuss results: because I included randomness in my train_a and dev selection, *as well as* when limiting the non-tornado examples, the results varied (sometimes noticeably) between runs. While this is a bit annoying for replicating my results, this is also a red flag for the models. Since a slight change in inputs can greatly affect the predictions, the models aren't stable and thus potentially not reliable.

With 36 sub-experiments, analysing each and every output (i.e., hyperparameter selection, confusion matrix, metric scores, etc.) would take a very long time and ultimately be fairly boring/ repetitive (spoiler: no sub-experiment did exceptionally better than the others). So instead, I will just walk through the results of the best-performing sub-experiment for each type of model to give a general picture of how the models performed. My choice for best sub-experiments can be seen in Table 1 – I chose them based on the highest $\mu - \sigma$ recall score.

A few things to note before diving into each model: first, all three top models limited the non-tornado examples. I have already explained the importance of addressing the tornado/ non-tornado data imbalance, but it's cool to see it working in practice. However, this technique seems to make the accuracy hover around 50 to 60%, which is not spectacular for a model. Plus, the recalls all seem to have a large variance, which suggests that even when the model does well, it's not terribly reliable. This variance comes from the model performing really well on one test storm but poorly on others.

Table 1. Accuracy and recall scores for models with best recall lower bound. (Sorry the technique goes over onto the second lines)

MODEL	ACCURACY ($\mu \pm \sigma$)	RECALL ($\mu \pm \sigma$)
REGRESSION; VARIOUS FILL; LIMIT	0.521±0.032	0.760±0.262
BASIC TREE; VARIOUS FILL; LIMIT	0.618±0.172	0.622±0.251
HGBC; NO FILL; LIMIT	0.559±0.074	0.608±0.231

Additionally, none of these models ignored all columns with missing values or filled in all NaNs with medians. To see my various-fill method come out on top (though just barely) is comforting; it shows that my intuition that we shouldn't always fill in "missing" values with the median when they are missing on purpose is justified. However, I'm not completely surprised that the HGBC preferred not having the missing values adjusted, since it is *designed* to know how to handle NaNs.

5.1. Logistic Regression

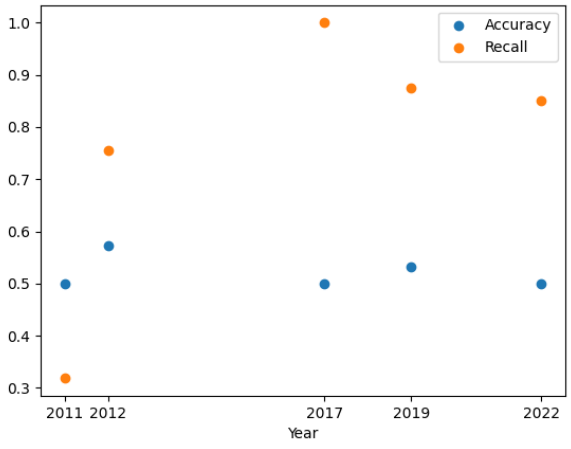

Figure 3. Accuracy and recall scores for every test storm for the logistic regression model with various fill and limit on non-tornado examples.

As mentioned earlier, each test storm had its own hyperparameters as chosen by a dev subset of the training set. As we can see from Figure 3, the 2017 storm had the best recall score, so we will look at its dev performance. Figure 4 shows that the recall and accuracy was best when there was only one input variable, which is not a great sign; how could something as complex as a tornado be described by one variable? The answer is that it can't. If we look at the confusion matrix on the 2017 test storm in Figure 5, we can see that the model always predicted tornado even though only half the examples were tornadoes.

Thus, it may be more telling to explore the results of, say, the 2012 test storm, since its accuracy and recall scores are closer to the model's average across all years. While Figure 3 shows a significant difference between 2017 and 2012's recall scores, 2012's confusion matrix in Figure 6 looks more reliable than 2017's confusion matrix in Figure 5. I believe these two observations can coexist because there are many more data points for the 2012 storm than the 2017 storm (note there are almost more true positives in Figure 6 than all examples in Figure 5!). This suggests that 2017's recall may be abnormally good because the test set was much smaller.

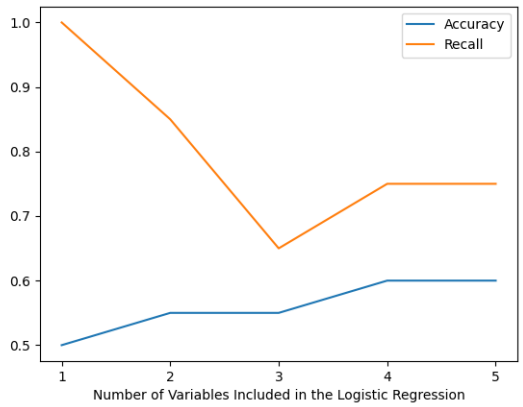

Figure 4. Accuracy and recall scores for the dev set (when the test storm is from 2017), based on the number of variables included in the model input.

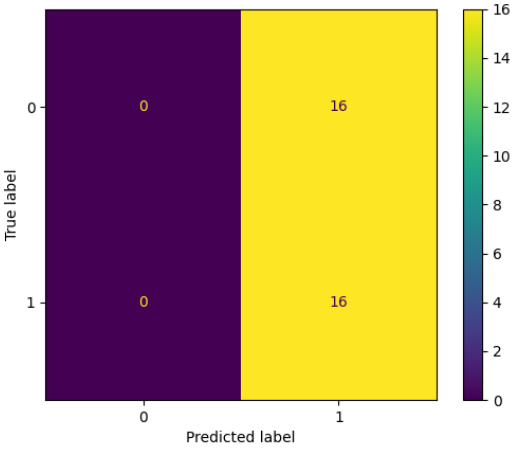

Figure 5. Confusion matrix for the logistic regression model with various fill methods and non-tornado limit on the 2017 test storm.

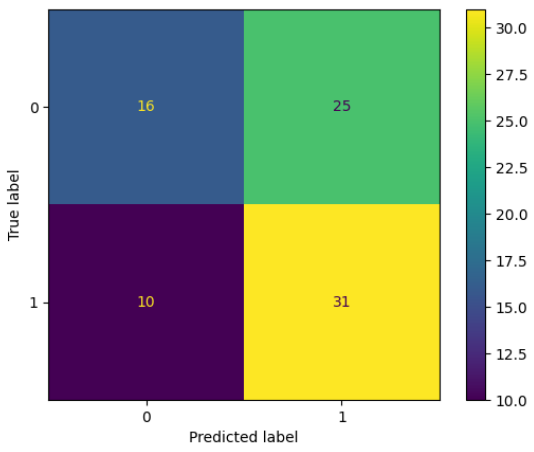

Figure 6. Confusion matrix for the logistic regression model with various fill methods and non-tornado limit on the 2012 test storm.

5.2. Basic Decision Tree

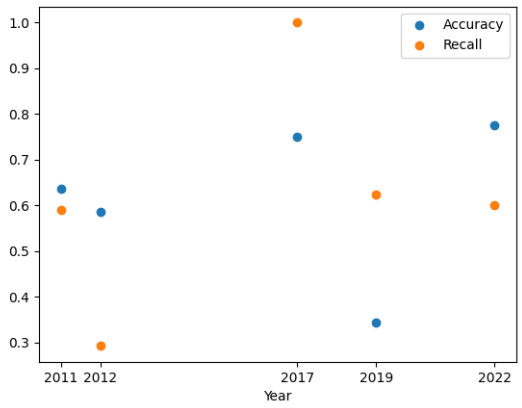

Figure 7. Accuracy and recall scores for every test storm for the basic decision tree with various fill and limit on non-tornado examples.

The best-performing basic decision tree model had missing values filled in in various ways and a limit on the number of non-tornado examples in the train and test set. While the 2017 storm has the best test recall score again, the model doesn't blindly predict all examples are tornadoes this time! Figure 10 shows us that this model is capable of correctly identifying all of 2017's tornadoes, as well as roughly half of the non-tornadoes, which is definitely better than the logistic regression model.

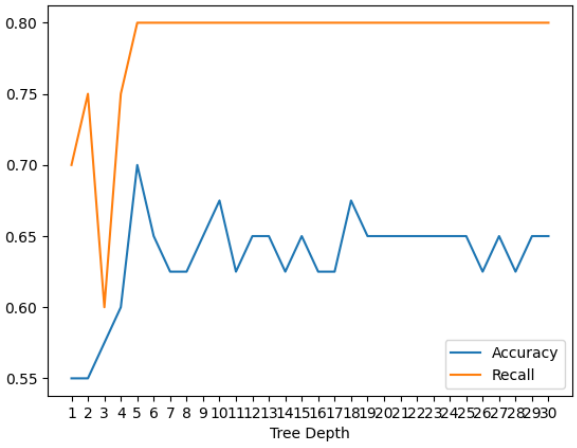

Figure 8. Accuracy and recall scores for the dev set (when the test storm is from 2017), based on the max depth of the tree.

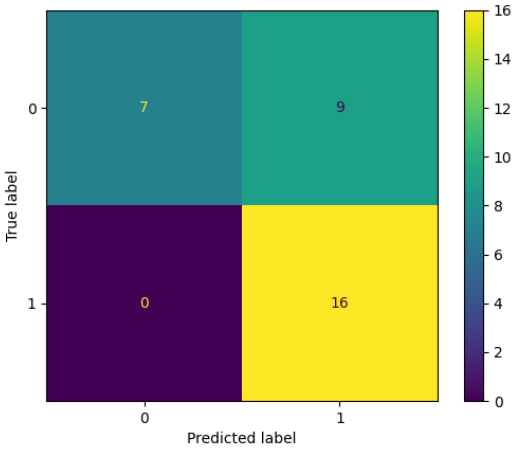

Figure 9. Confusion matrix for the basic decision tree with various fill methods and non-tornado limit on the 2012 test storm.

However, it's hard to get too excited about this basic decision tree when the accuracy and recall scores are all over the place (as seen in Figure 7), especially when it performs the best on the test storm with fewest tornado examples and the worst on the storm with most tornado examples.

5.3. Complex Decision Tree

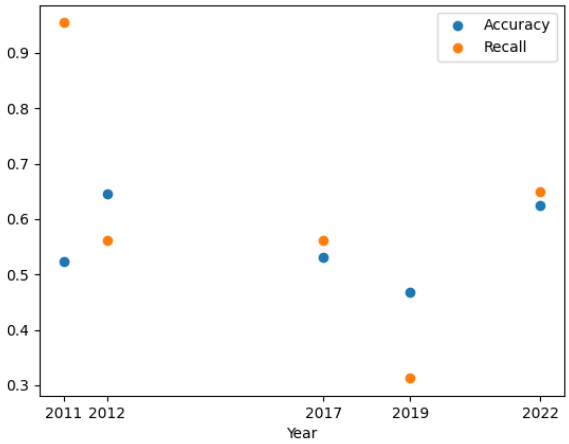

Figure 10. Accuracy and recall scores for every test storm for the complex decision tree (HGCB) with no fill but a limit on non-tornado examples.

The best of the HGCB models was the one where I didn't address the missing values, but I put a limit on the number

of non-tornado examples. Despite being the most complex model I tried, the HGCB decision tree still suffered from having its accuracy and recall scores vary wildly from storm to storm. Unlike the previous decision tree, this one performed the best on the storm with the second most tornado examples an worst on on of the smallest storms... but like the logistic regression, this high recall was caused by the fact that the model always predicted a weather report was from a tornado. I honestly did not think a decision tree would make this mistake, but it makes sense when we see the dev recall score peaked around `max_depth = 4`, which is oddly low for having the opportunity to split on so many weather variables (similar to how to logistic regression for 2017 chose to use only one variable).

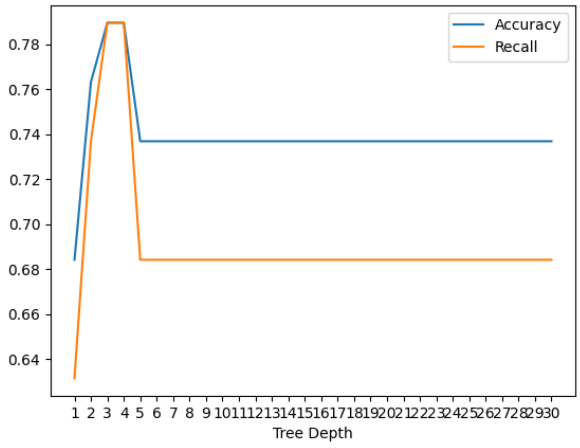

Figure 11. Accuracy and recall scores for the dev set (when the test storm is from 2011), based on the max depth of the tree.

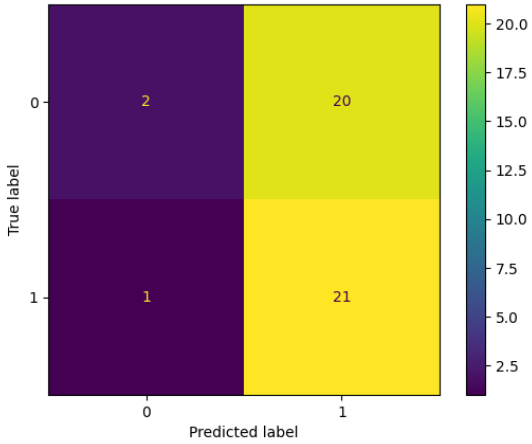

Figure 12. Confusion matrix for the complex decision tree with no fill but a limit on the non-tornado examples on the 2011 test storm.

The results for the 2022 test storm were closer to the model's overall averages. When looking at its confusion matrix in Figure 13, we can see that at least the model managed to predict both positive and negative cases, though only a little better better than chance.

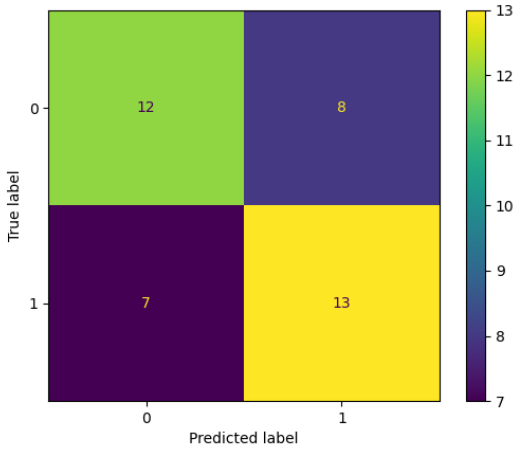

Figure 13. Confusion matrix for the complex decision tree with no fill but a limit on the non-tornado examples on the 2022 test storm.

5.4. Overall

While these last models were just three of the "best" 36 sub-experiments, the other sub-experiments also had greatly varying accuracy and recall scores. This not only made it hard to choose the best models, but also makes it hard

to trust the model outputs. I don't think any model type (regression or tree) is the best... or even good for that matter, and I believe the large variance on the recall scores is partly to blame.

That does not mean, however, that there was nothing to be gained from these experiments! In Figure 14, I have calculated the average[7] accuracy and recall scores for each of the techniques (model type, way of addressing NaNs, and limits on non-tornado examples). While most of the accuracy and recall averages overlap from technique to technique, there is a noticeable difference when we consider tests with no limit on non-tornadoes vs tests with a limit. Without a limit, the accuracy hovers around 80%, which is essentially the proportion of non-tornado examples. This means the models didn't *need* to learn anything, they could just always predict no tornado. The increase in recall from no limit to limit is further confirmation that the tornado/ non-tornado imbalance needs to be addressed in some way.

Unrelated to the technique averages, I noticed that there were popular variables for the logistic regression to use or the basic decision tree to split on. For example, `TEMP_C_CHANGE`, `VISIBILITY_SM`, and the sky condition elevations were often some of the most correlated variables with `TORNADO`. And the basic tree often split on `TEMPERATURE_C`, `ALTIMETER`, and `WIND_SPEED_CHANGE`. While the models overall did not perform too well, these reoccurring "important" variables may suggest that there are some features that are more related to the formation of tornadoes than others!

6. Discussion

For consistency and simplicity, I will try to explain why the three sub-experiments I discussed earlier misclassified what they did (other than just blindly predicting all tornadoes or all non-tornadoes). The logistic regression would sometimes misclassify tornadoes with `SKY_CON_2_FT` and `SKY_CON_3_FT` = 300 (i.e., there originally had been no cloud elevation for these levels) and non-tornadoes with clouds in layer 2 and 3 below 30,000 ft. While this happened for several of the test storms, it was particularly bad for 2011. Upon closer inspection, I realized that stations KGKY, KRBD, and KFTW all were within 15 miles of a tornado (i.e., tornado-positive) despite having clear skies over the weather tower. Obviously it would be weird to expect a tornado during a sunny day, so this test storm is a little unfair to the model. Other test years had the model thinking weather with low-visibility were tornadoes and high-visibility were non-tornadoes, despite the opposite being true. Again, that sounds like a reasonable heuristic but unfortunately it did not line up with reality in this case.

The basic decision tree experienced similar problems, where tornadoes with few/ high cloud layers and non-tornadoes with many/low cloud layers confused the model. I also saw instances of warmer tornadoes and tornadoes without recorded wind gusts being mistaken for non-tornadoes, though these patterns were less prominent. Unlike with the logistic regression, there were some mistakes that were less obvious. These were examples where I knew were misclassified I but couldn't point to the variable(s) at fault. These mistakes might be caused by examples being ambiguous or near a decision boundary, in which case it's not horrible that the model misclassified them.

I ran into this problem a lot more for the GHBC model, where I could hardly ever tell what exactly went wrong. I had assumed that it would make similar mistakes to the basic decision tree, but that wasn't the case. There didn't seem to be an obvious difference between the tornadoes it got wrong and the non-tornadoes it got wrong, again suggesting that maybe these examples are just too close to each other to reliably separate. I wonder if the ensembling that GHBC does contributes to the lack of pattern amongst misclassified examples. Maybe having more data would help, or maybe these variables just aren't enough to determine what makes a tornado.

All in all, I had assumed that the decision trees would be able to perform much better than they ended up doing. When I first started collecting the data, it seemed obvious when I reached a tornado; looking back on it, I think I was really only identifying severe weather. Since going through some of the misclassified examples, I've realized that the data has more traps than I first thought (like how fewer clouds doesn't always mean there isn't a tornado nearby). With how complex and confusing the weather is, I'm not terribly surprised or disappointed in the lackluster results of all of my experiments. As for areas of improvement, I think the models could do better with more (training and test) data, as well as a different definition of what counts as a tornado (instead of simply within 15 miles).

7. Conclusion

In this project, I sought a model that could take in weather readings and predict whether or not those readings came from near a tornado. I made a logistic regression model, a basic decision tree, and a complex decision tree, but none had consistently high accuracy and recall across the various storms I performed cross validation on. While I was a little surprised that the decision trees did not significantly outperform the baseline logistic regression, a close inspection of where the models went wrong revealed that the dataset would likely be hard for even a human to get correct! That

[7] The numbers averaged came from the results of each test year for each model that included the given technique.

Technique	Accuracy ($\mu \pm \sigma$)	Recall ($\mu \pm \sigma$)
Logistic Regression	0.680±0.190	0.343±0.372
Basic Decision Tree	0.651±0.167	0.349±0.244
HGBC	0.693±0.144	0.338±0.295
Drop Missing*	0.664±0.168	0.352±0.329
Median Fill	0.674±0.179	0.291±0.272
Various Fill	0.681±0.171	0.382±0.327
No Non-Tornado Limit	0.804±0.094	0.137±0.146
Non-Tornado Limit	0.545±0.115	0.551±0.281

Figure 14. Summary average accuracy and recall scores by technique. *Does not apply to the HGBC

being said, the experiments did succeed in demonstrating the risk of having a very unbalanced dataset, as well as potentially finding some key weather variables when it comes to predicting tornadoes.

This project let me work more with decision trees, which we had only briefly discussed in class. While I enjoyed using this type of model, I was a little surprised that not every type of decision tree could handle missing values. This gave me the opportunity to learn about more advanced decision trees (like HGBC) and made me consider questions about how to address missing data (which I have not had to deal with to this degree previously). Plus, I got to learn about weather data; it's not machine learning, but was definitely fun to deal with and read about!

As for next steps, if I thought a decision tree really could predict tornadoes (and I think it would be really cool if they could!), I think I would need more data points and more weather variables. I also think my way of labeling positive and negative tornado examples could use a rework, since 15 miles seems to sometimes be too far away.

References

El Paso Times. A history of twisters: Tornadoes in texas since 1950, 2023. URL https://data.elpasotimes.com/tornado-archive/texas/.

Lagerquist, R. et al. Deep learning on three-dimensional multiscale data for next-hour tornado prediction. *Monthly Weather Review*, 148:2837–2861, 2020.

NOAA National Weather Service, U.S. Federal Aviation Administration, U.S. Department of Defense, NOAA National Centers for Environmental Information. 5-minute surface weather observations from the automated surface observing systems (asos) network. 2023.

Trafalis, T. B., Adrianto, I., and Richman, M. B. Active learning with support vector machines for tornado prediction. *Computational Science - ICCS 2007*, pp. 1130–1137, 2007.

US Department of Commerce and National Oceanic and Atmospheric Administration. Federal meteorological handbook no.1 surface weather observations and reports, 2019.

A. Appendix

Weather Variables Scraped from ASOS	
Name in tornadoes.csv	Description
WIND_DIR	Wind direction in degrees
WIND_SPEED	Wind speed in knots
WIND_CHAR	Character of wind, often gusts
WIND_CHAR_SPEED	Speed of wind character
PK_WIND_DIR	Direction of fastest wind that hour
PK_WIND_KT	Speed of fastest wind that hour, in knots
VISIBILITY_SM	Surface visibility in statute miles
SKY_CON_X_DESC	Cloud cover of the xth layer recorded
SKY_CON_X_FT	Height of xth layer recorded, in 100 ft
CLOUD_TYPE	Type of clouds observed
TEMPERATURE_C	Temperature in degrees Celsius
DEW_PT_C	Dew point in degrees Celsius
ALTIMETER	Normalized pressure reading in fractions of inches of mercury
SL_PRESSURE	Sea-level pressure, in hectopascals
FALLING_RAPID	Indicates if the air pressure is quickly falling
RISING_RAPID	Indicates if the air pressure is quickly rising
TSTORM	Indicates if it was storming
RAIN	Indicates if it was raining
HAIL	Indicates if it was hailing
WINDSHIFT	Indicates if there was a windshift

Link to data, code, and misc. math:
https://drive.google.com/drive/folders/1EQct8OtWVZXL_2FaMbQ7XfaJh_2tdE3d?usp=sharing
Note sometimes the underscores don't show up when you
copy the link, so verify that there's an underscore between
L and 2 and h and 2!