How Much Did it Rain:
First place solution

Devin Anzelmo, devinanzelmo@gmail.com

Summery

This is a description of my solution for the competition How Much Did it Rain? I used multi-class classification with soft labels to estimate rain amount. The dataset was split based on the number of radar scans to allow for better feature extraction and ameliorate some of the issues surrounding class imbalance. My final model, after debugging and simplification, scored 0.007485 on the private leaderboard and 0.00751 on the public leaderboard.

Feature Extraction

Each variable in the original dataset was expanded so that each scan had it's own column. The number of error codes were counted(-99903, -99900 etc) for each row, as well as the total number of non-error code features. The error codes were then replaced by np.nan and various descriptive statistics were calculate for each row(ignoring the nan values). For HydrometeorType the number of each code for each row is counted. Using these methods created 191 features which were used for all subsets of the data.

In addition for the subsets with 8-17 radar scans and for greater then 18 radar scans, TimeToEnd was sorted and used to break the hour into 10 and 20 equal length segments respectively. In each segment the mean was calculated and used as a feature. These features increased the score for the dataset with more then 17 scans by a large amount but did little for subsets with less scans.

Feature Selection

Though feature selection could remove approximately 50% of the features without worsening the score I did not remove many features. For the subset with only 1 radar scan all columns with identical values were dropped. Additionally as DistanceToRadar, and the count of -99903 and -99900 error codes for HybridScan dramatically worsened the score when the labels above 70mm were included I dropped them. After removal of large rain amount samples these features only slightly decreased the score and but were still left out.

Modeling Approach

My overall modeling approach was treat the predictions as a linear combination of cdfs that I estimated from the training set. Each component cdf was weighted by the class probability for the cdf's associated label, as given by the classification algorithm.

The best prediction for a sample with the true rain amount of 0mm is a row of all ones, and for 1mm true rain amount the best prediction is all ones except for a single zero at the 0mm position. Using this

it seemed reasonable that if the classifier predicted p(rain=0mm)=0.5 and p(rain=1mm)=0.5 then the best cdf would be the average of all ones, and all ones except at zero([0.5,1,1,...]).

Based on this intuition I decided to model the final distribution as a linear combination of step functions and empirical distributions from the training set.

There was a decent amount of class imbalance in the dataset. About 85% of the data had no rain at all, and at rain amounts greater 50mm there were less then ten examples per class. To combat this I aggregated the higher rain amounts into a single class label. I then used the distribution of rain amounts from the training data falling into the aggregate to estimate a cdf.

While doing feature engineering I decided to split dataset apart based on the number of radar scans. It was not possible to make the same type of features for a sample with 20 radar scans as it was for one with only 1 radar scan. Splitting the dataset decreased training time as I found I could get by with fewer features for a majority of the data. Additionally tackling the problem this was alleviated some of the class imbalance problems for the datasets with more scans. In the dataset with only one scan 95% of the data had 0mm of rain, and there were very few with large amount of rain. This allowed for modeling the dataset with only 3 labels. In contrast for the subset with greater then 17 scans only 48% of the data had 0mm of rain. For this set I was able to use 12 different labels.

Xgboost's Gradient Boosted Decision Trees classifier was used for all supervised learning. Parameters were ballpark estimates of what might prevent over fitting. Whenever I tuned parameters my leaderboard score would either not change or worsen so I stopped tuning.

At the end I added some post-processing which improved the score by ~0.00001. For each of the subsets of the data I subtracted a constant value from all the predictions. The amount I subtracted was roughly the same as the proportion of radar scans greater then 70 in the train set. I am not sure whether this was a coincidence or that because large rain amounts increased the score the most subtracting a little from each prediction mitigated their impact.

Code Description

`standard_features.py` processes the given train and test file by expanding the columns. It generates labels, count features and stats features.

`binned_featues.py` creates features based on segmenting the hour for each sample. It is called from `make_binned_features_train.sh`, and `make_binned_features.sh`

`make_binned_features_train.sh` creates features for the train set for data subsets 4 and 5. It currently spawns 7 processes at a time and then waits for completion this can be increased if you have more cores.

`make_binned_features_test.sh`: same as above except for the test set.

`functions.py`: Most of the code is in `functions.py`. It contains functions to calculate CRPS, load the data, process the labels, generate CDFs, train the classifiers, and make the final predictions.

`train_subset1.py` through `train_subset5.py` each try models for one of the five subsets of

the data. They save models into the folder 'models'

`make_predictions.py` loads the models generated earlier and creates the final solution from the test set.

`main.sh`: runs all the other code. First `standard_features.py` then the make_binned_features code, followed by training and finally prediction. Run this to generate solution.

Dependencies

- Python 2.7.9
- Pandas 16.0
- Numxpr 2.3.1
- xgboost 0.4
- numpy 1.9.2

All code was ran on a machine with 32gigs of ram, with ability to run 8 threads. Final solution requires approximately 12GB of disk space. OS was ubuntu 14.04, will likely run on most linux distributions. There have been some changes in the master branch of xgboost as of 6/6/2015 which will change the performance of this solution, therefore 0.40 is recommended.

README(how to generate a solution)

Put `train_2013.csv` and `test_2014.csv` into the `input` folder. Then `cd` into `code`, folder and type

```
./main.sh
```

The following folders should be in the top level directory, `input, code, processed, models, output`

Simple Features and methods

In `train_subset4.py` and `train_subset5.py` change the line `use_xtra_features = True` to `False`.

In `main.sh` comment out the lines `./make_xtra_features_train.sh`, and `./make_xtra_features_test.sh`.

In `make_predictions.py` change the line with `fn.make_prediction4(.....use_xtra_features=True)` to `fn.make_prediction4(.....use_xtra_features=False)`, repeat for `fn.make_prediction5`

Now run `main.sh` and it should shave off ~5 hours of run time. Final score about 0.00750(still enough for first place)

Acknowledgement

I would like to thank everyone on the forums who gave away helpful information and hints.

References

Xgboost can be found at “https://github.com/dmlc/xgboost” Thanks to Tianqi Chen for making such useful software.