An Ensemble Method for Rainfall Prediction

University of California, Irvine

Nile Hanov (12322688), Pai-Chun Wang (27609098), Tzu-Chi Lin (48964286)[1]

{nhanov, paichunw, tzucl2}@uci.edu

Abstract

In this report, we present our experiment results that were obtained with multiple approaches, ranging from Random Forest to Deep Neural Network. The final model was a mixed ensemble of all the individually tried, tested and tuned models.

1 Introduction

This in-class Kaggle competition is focused on predicting the presence or absence of rainfall at a particular position, using satellite-based measurements of cloud temperature. We experimented with several algorithms including logistic regression, gradient tree boosting, random forest, extra tree, k-nearest neighbor and deep neural networks. We then ensemble all the useful classifiers by voting and applying the weights based on its validation AUC.

2 Dataset Exploration

The training dataset includes 200,000 data points and each data has 14 features. The result is a binary class which represents whether it will rain or not. However, we do not know the meaning of each feature. We can only explore the dataset by its value to see if there is any pattern in it or whether it has irrelevant attributes.

2.1 Data Visualization (Tzu-Chi)

We tried to visualize the dataset to see if it has any patterns we can focus in our analysis. Principle Component Analysis (PCA) dimension reduction technique was used to plot 14 features on a 3-dimentional plot. Unfortunately, the resulting plot, *Figure 1*, didn't give us any additional insight into this dataset.

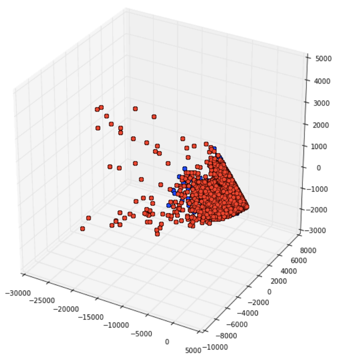

Figure 1. PCA Visualization

[1] *Ordered alphabetically*

2.2 Feature Selection (Nile)

Next, we tried to prune some of the 14 features to find potential ways of improvement. We trained a Random Forest classifier and then got the feature importance scores out of it. The following table shows the feature importance scores for rainfall dataset.

Table 1. Feature Importance Scores

Feature Index	Importance Score
1	0.25259183
2	0.0353897
3	0.07278631
4	0.06505834
5	0.06902842
6	0.04627751
7	0.01080431
8	0.064176
9	0.06861998
10	0.046647
11	0.07075086
12	0.0783339
13	0.05179551
14	0.06774031

By looking at the aggregation of feature scores we can see that some of the features are more important and have higher predictive power that the others.

Table 2. Aggregation of Feature Scores

Max	0.25259183
Min	0.01080431
Mean	0.07142857

After setting the score boundary we pruned our features to a smaller but more relevant subset and retrained our model. The cutoff boundary was set to 0.05 and that pruned the feature set from 14 to 10. However, the accuracy score didn’t improve with just these changes and actually dropped by 0.36%. This is still an interesting result because we removed almost 1/3 of all features and all the data, 28.6% to be more specific, but the accuracy only dropped by less than 1%. In many real-world applications, the data isn’t as clean and many features carry very little to no importance at all and this method shows how to deal with such cases.

3 Individual Models

All the following models are implemented with scikit-learn open-source library. It is a simple and efficient machine learning library written in Python that allows to easily implement and test various popular and powerful algorithms. This library includes classification, regression and clustering algorithms such as random forests, gradient tree boosting and many others.

3.1 K Nearest Neighbor (Pai-Chun, Tzu-Chi)

Since the dataset is based on a particular position, we assumed that the results could be dependent on the distance between the points. We started using KNN with different k values and found associated AUC values for the performance. In *Figure 2* we can see that when k equals to 10, AUC is at its highest point (0.723). This AUC is more than 0.7 so we included KNN into our final ensemble of all classifiers.

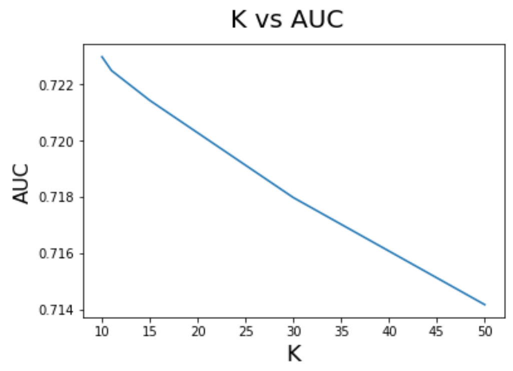

Figure 2 K vs Validation AUC plot for KNN model

3.2 Gradient Tree Boosting (Nile, Tzu-Chi)

For this model, we looked at optimizing multiple parameters. We started with max_leaf_node, which is the parameter responsible for maximum number of terminal nodes in the tree. In gradient tree boosting, if max_leaf_node set to a higher value the model is less likely to overfit the data. Hence, we tune this parameter to control model complexity. From Figure 3, we can see that the best AUC is 0.7754 when max_leaf_node=150.

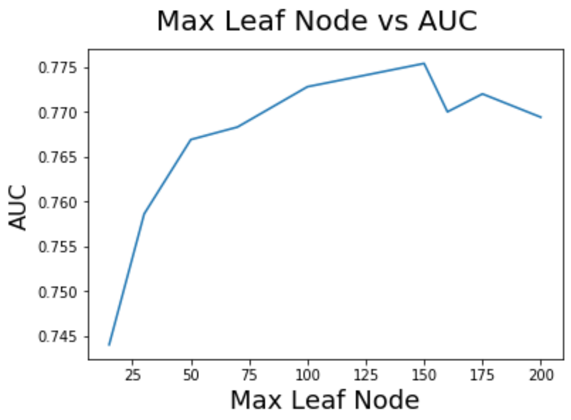

Figure 3 Max Leaf Node vs AUC for Gradient Tree Boosting

We then tune the parameter n_estimator, which represents the number of boosting stages to perform. Typically, a large number results in better performance but gradient boosting can easily overfit. However, from *Figure 4* we can see that the best result is AUC equal to 0.7754 when n_estimator is 100.

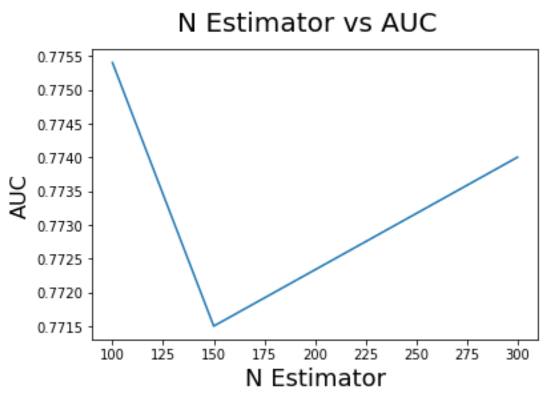

Figure 4 N Estimator vs AUC for Gradient Boosting

From the plot, we can see that it also has a high score when n_estimator is equal to 300, so we tried different values of max_leaf_node under n_estimator equal to 300. The result for this attempt is shown in *Figure 5* with best AUC equal to 0.774, which is still lower than 0.7754. Hence, we choose n_estimator value of 100 and max_leaf_node value of 150 for our ensemble.

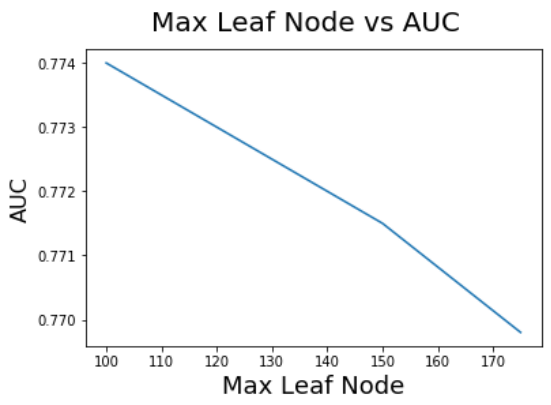

Figure 5 Max Leaf Node vs AUC for Gradient Boosting

3.3 Random Forest (Nile, Tzu-Chi)

We then tested Random Forest and tuned its parameters to get the best performance. Tuning n_estimators, max_features and min_samples_leaf proved to be the most effective. To control overfitting, we tried changing min_samples_leaf which represents minParent parameter of the decision tree covered in the lecture. It means that the branch will stop splitting once the leaves have that number of samples each. Originally, it was set to one but that resulted in a model that was overfit. The best results were achieved with min_samples_leaf set to 5. We then gave n_estimators the following set of values: 10, 100, 1000, 10000. Each time we increased this value the accuracy improved. Finally, max_features controls how many features each tree is randomly assigned. If the value is large the model will overfit and if it is too small it will underfit. The optimal value for max_features for most random forest models is the square root of the number of features ($\sqrt{14}$). This is also something we found to be true from our experiments so we stayed with this value.

3.4 Extra Tree (Tzu-Chi)

Extra tree classifier in scikit is very similar to random forest. It implements a meta estimator that fits a number of randomized decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting. Hence, it is also easier for the model to overfit. We tuned the max_depth parameter to control this issue. From *Figure 6*, we can see that the best performance is AUC equal to 0.7836 when max_depth is set to 50.

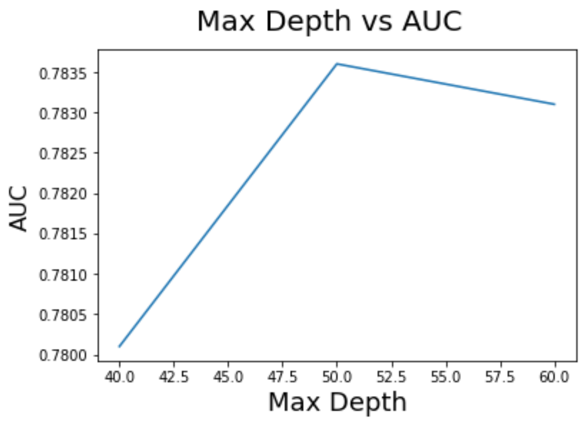

Figure 6 Max Depth vs AUC for Extra Tree Classifier

3.5 Deep Neural Network (Pai-Chun, Tzu-Chi)

The deep network was implemented with MLPClassifier library. We tried to use three-layers for the model with different number of hidden node to see its performance. We also used pipeline to speed up the execution time. According to Table 3, when the hidden node is (100, 50, 100), the performance is the best. We choose it as our model parameter for the final ensemble.

Table 3. NN Performance

Hidden Layer and Node	**AUC**
(100,20,100)	0.7308
(100,30,100)	0.7335
(100,50,100)	0.7403

4 Ensemble Method (Tzu-Chi)

Assume $\mathrm{G} = \frac{1}{T}\sum_{t=1}^{T}\alpha(t)g(t)$, where g is individual model, α is its weight, and G is blending model. By the theorem, we could prove that AVG(Error(g)) $\geq$ Error(G), which means we can always improve our performance creating an ensemble of all the models. We use Voting Classifier to blend all the above-mentioned models and set its weights based on its validation AUC. The final result becomes AUC equal to 0.7974 which indeed improves the performance.

5 Conclusion

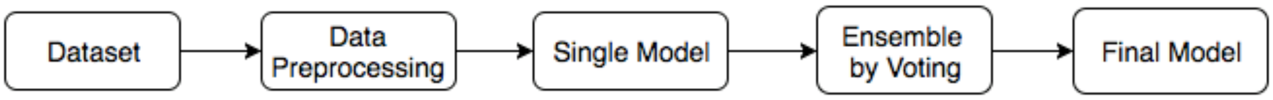

Figure 7. Process Flow

First, we explored the dataset and trimmed the useless attributes to reduce the noise. We then implemented several individual models and combined them into an ensemble by voting classifier. As we can see from Table 4, the accuracy of the Voting Classifier is higher than all of them.

Table 4. Performance of all models

Classifier	AUC
K Neighbors Classifier	0. 722975348052
Gradient Boosting Classifier	0. 775414018092
Random Forest Classifier	0.786400407683
Extra Trees Classifier	0.783597743574
MLP Classifier	0.740351583633
Voting Classifier	**0.797482749581**

We choose the Voting Classifier as our final submission. The final result of our score on the public leaderboard is 0.79731 (rank 32) and private leaderboard score is 0.79420 (rank 33).

The best performing submission is named **Y_submit_Final2.txt.**

Table 5. Kaggle User Names

Kaggle Names	**Student Name**	**File Submissions**
Tzu-Chi Lin	Tzu-Chi Lin	**Y_submit_Final2.txt**
Pai-Chun Wang	Pai-Chun Wang	Y_submit_ensemble.txt
Nile Hanov	Nile Hanov	Y_PROJECT_3.txt