CS 178 Final Project: Rainfall Prediction

Kaggle Team: geoprism
Brian Wong
Sean Kim

Introduction

Our CS178 Final Project consisted of building predictive models to analyze satellite-based measurements of clouds, in order to accurately predict the presence or absence of rainfall in a particular location. This data was made available from the UC Irvine Center of Hydrometeorology and contains a set of 14 features that include information such as cloud area, temperature, density, humidity, etc. The target y value is a binary indicator whether there is rain or no rain at a location. In order to accurately predict rainfall, our team will be performing feature selection as well as utilizing different machine learning models. Lastly, we will build an ensemble “blend” of our predictors and submit our final score to Kaggle.

Data Visualization

Below(fig. 1) is a pair plot of each of the 14 features. This makes it easy to recognize any correlations between two features. One of the most important steps in machine learning is to identify key features in a dataset, which in turn, will allow for dimensionality reduction and simpler calculations. In the next stage, we will try to perform Principal Component Analysis(PCA), which should reduce the dimensionality and better prepare our input data.

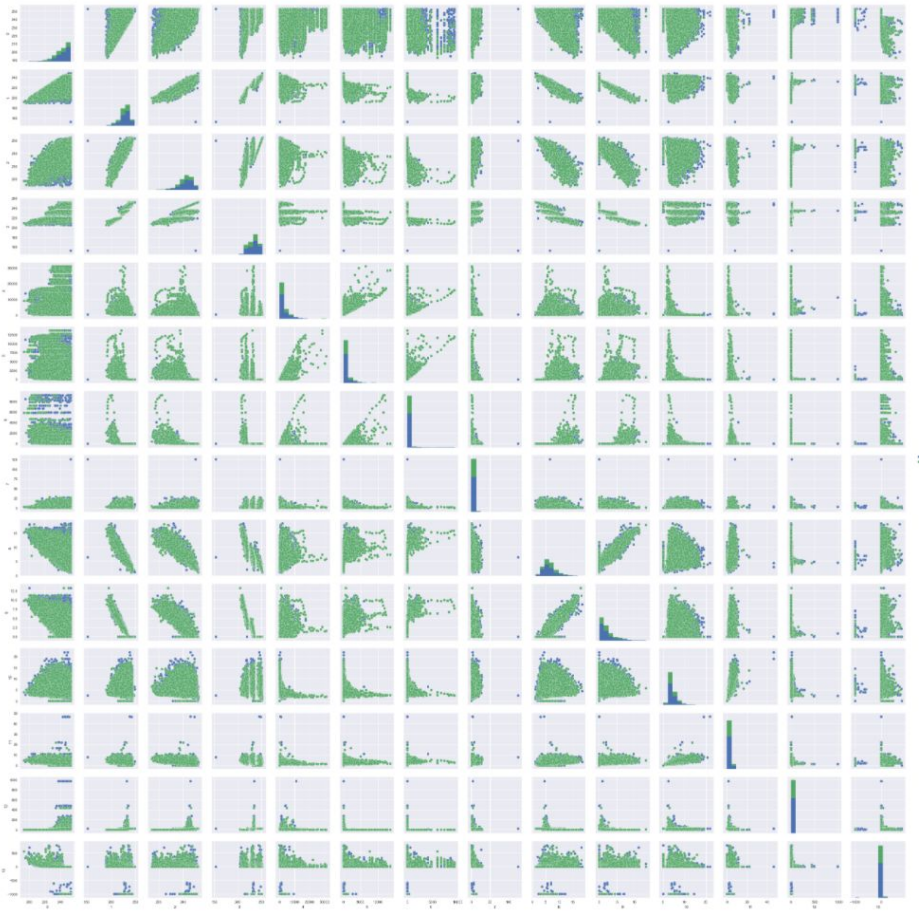
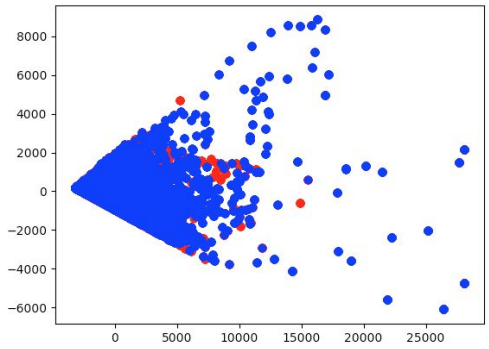


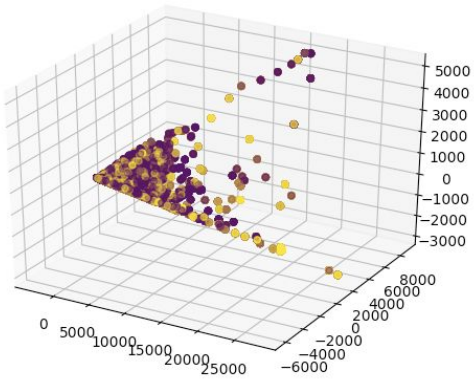
Fig.1 Pair Plot

Dimensionality Reduction

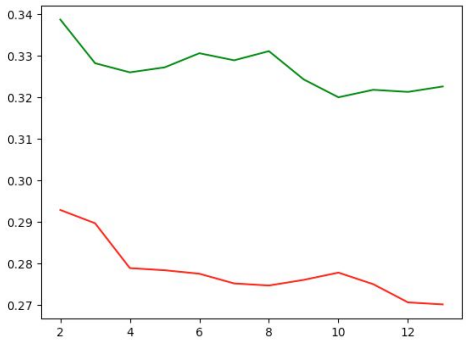
A reduction down to 2 or 3 features, allowed for a human readable graph, but revealed no noticeable trends to the naked eye. Next, we wanted to find the most optimal dimensionality for our data by performing Principal Dimensionality Analysis (PCA) and calculating the mean squared error on a decision tree model. Results will vary; however, we noticed most features were heavily correlated with each other and found it optimal to have a maximum of 10-14 features. Doing so would still keeping a low MSE for our validation error.



(Fig.2 Feature Reduced to 2)



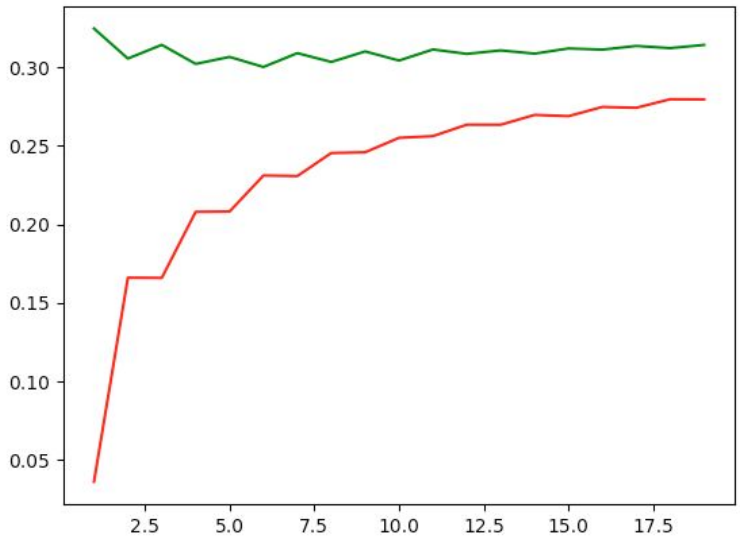
(Fig.2 Feature Reduced to 3)



(Fig. 3 Finding Best Feature Reduction)

K Nearest Neighbor

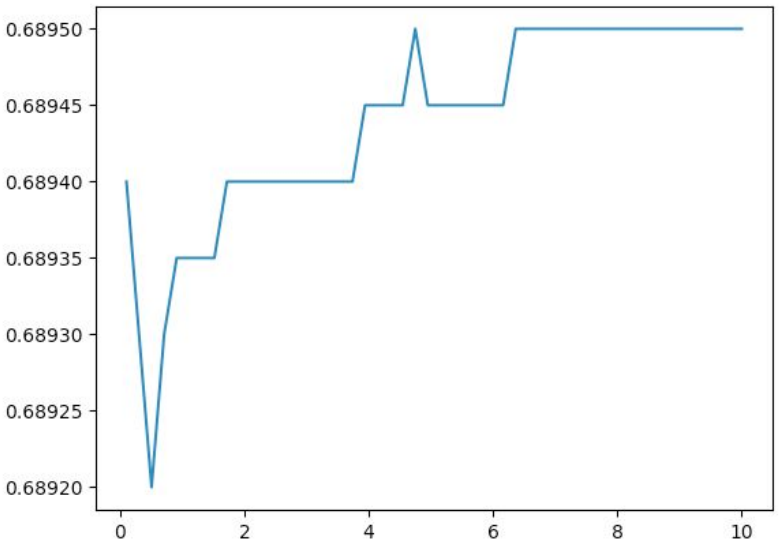
KNN is another straightforward model that classifies based on the K nearest features to a specific data point. We tried to find the most optimal K value which varied between 7-10(Fig. 4). We also standardized our feature data by removing from the mean and scaling to unit variances. We plan to use this model during our ensemble process.



(Fig. 4 K Nearest Neighbor MSE on Validation vs Training)

Logistic Regression

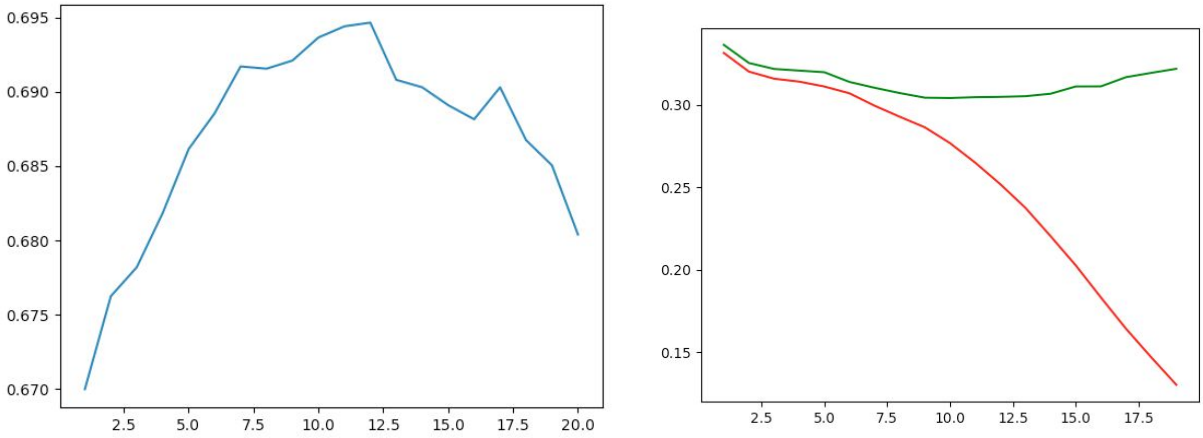
Logistic regression is fairly intuitive and usually the go to method for binary classification. This model formulates a probability function that can take in a given input, and map to anyone of our various y class values. To increase our predictive score, we played with the inverse regularization strength(Fig.5), and found a stable value at 10. We also standardized our feature data by removing from the mean and scaling to unit variances. We plan to use this model during our ensemble process.



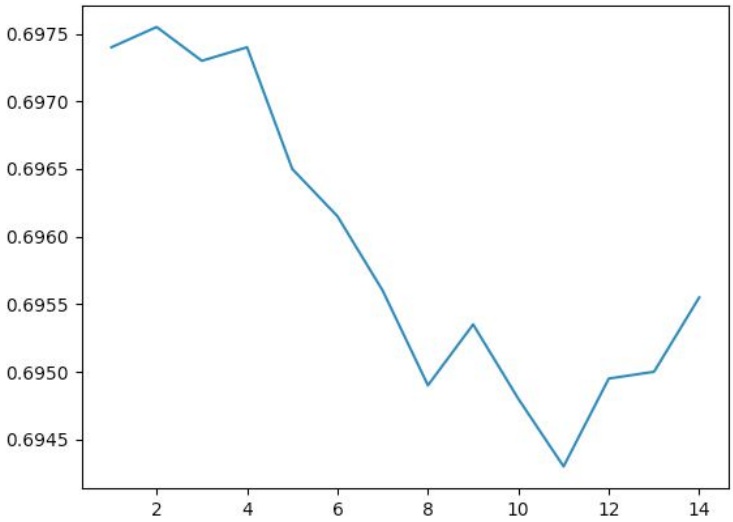
(Fig.5 Regularization vs. Accuracy)

Decision Tree (Classification)

Decision Tree is the same model that was used in homework 4, except this time we used SciKit Learn's version. The graph in Fig.6 shows an obvious optimal depth at 10. We then tested for an optimal minLeaf in Fig. 7, which show no legitimate improvement between scores. Largest difference between minLeaf scores was 0.003 which can be considered negligible. We also standardized our feature data by removing from the mean and scaling to unit variances. We plan to use this model during our ensemble process.



(Fig 6. Depth vs. Accuracy and Training Error vs. Validation Error)



(Fig.7 MinLeaf vs Accuracy (Max Difference: 0.003))

Decision Tree (Regression)

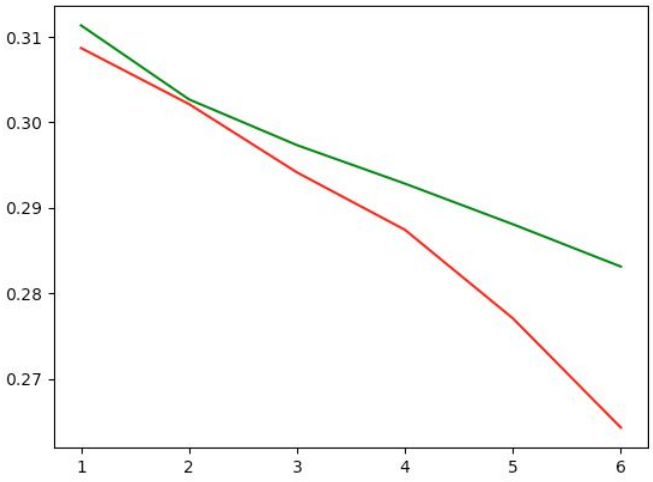
We splitted the data set into 75% / 25% training/test and to be consistent with our results, we chose not to shuffle the data and use the first 75% of the data for training and the rest for testing. We will try to determine the best parameter values for our predictor and see where the overfitting begins.

Depth	Train Error	Validation Error	Depth	Train Error	Validation Error
1	0.21414737138996137	0.21429714121695848	**11**	0.1805112756652484	0.21081789672391343
2	0.21198822639978146	0.21188405282911613	**12**	0.1716400381852704	0.2144397484019022
3	0.2097693622167887	0.21012892273249195	**13**	0.16153980823496494	0.21982930306707732
4	0.20780819808547196	0.20891163853632802	**14**	0.14999398469264677	0.2269282888193063
5	0.2057177286369151	0.20739453992675375	**15**	0.13794380465902176	0.23505774573418522
6	**0.20345375977656263**	**0.20587697416717649**	**16**	0.12503223556651183	0.24346313817747475
7	**0.20069398582645276**	**0.20560766766869365**	**17**	0.12503223556651183	0.24346313817747475
8	0.1974823940241835	0.2058216650786626	**18**	0.09922819731431107	0.26271016193005275
9	0.19310593316247762	0.20694685188693024	**19**	0.08738531088434245	0.2707490121532585
10	0.18749738436374247	0.20837423931573998	**20**	0.07587509663862962	0.2810490738998003

Gradient Boosting

“Gradient boosting is a machine learning technique which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.” -Wikipedia

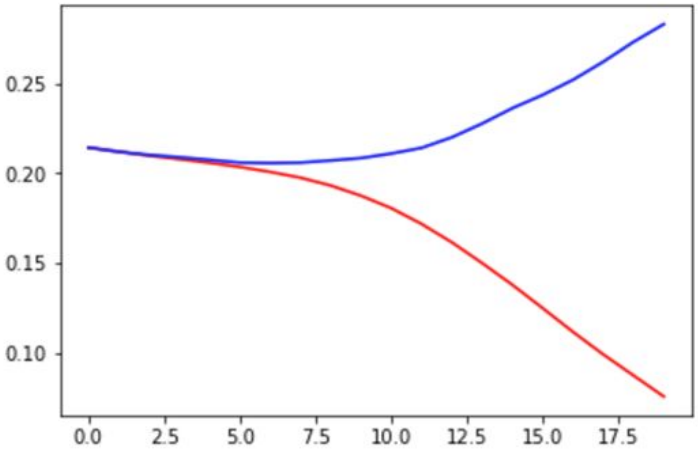
Since Gradient Boosting works well with decision trees, we decided to find the optimal number of trees to achieve lowest error rate with highest AUC score. To choose the best number of trees and best performance, we choose several values at random. This graph shows an optimal maxDepth to be around 3-4. Just to be consistent with our previous tree regression example, we splitted the data set into 75% / 25% training/test.



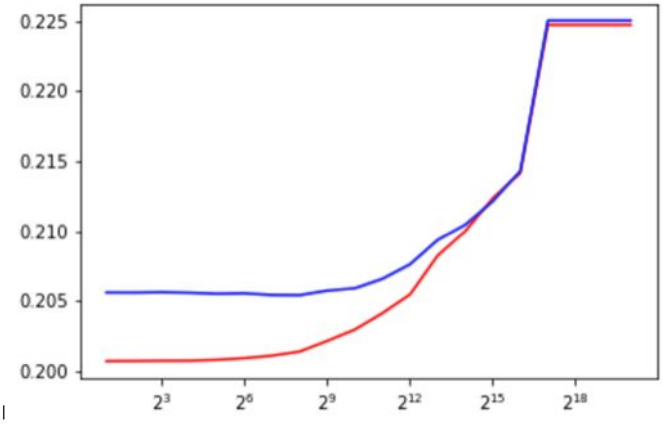
(Fig.8 Max Depth vs. Training/Validation Error)

We have simulated simple decision tree gradient boosting for MSE. We have used a strong complexity control with optimal maxDepth we found in the earlier graph and step size 0.5.

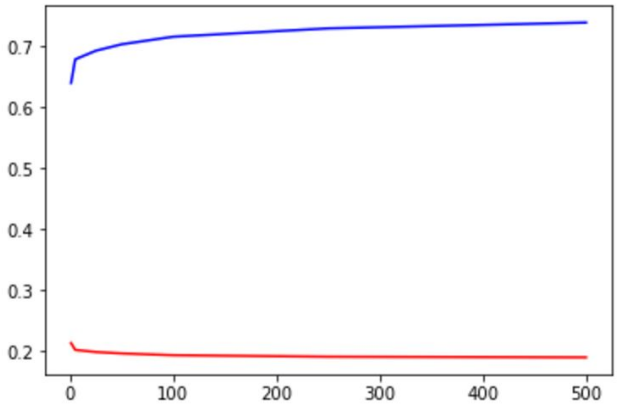
# of learners	MSE	AUC
1	0.21386101133854193	0.6401627858736998
10	0.20258054696789746	0.6787986255563768
25	0.1991363411897412	0.6929406987288125
50	0.196789340803985	0.7034856377825708
100	0.19387342162783844	0.7158861850850414
250	0.19143507238238766	0.7295533280068582
500	0.19036674520545457	0.7390768228797353
1000	0.19010203753497398	0.740167418992712



(Fig. 9 A depth higher than 7 caused overfitting)



(Fig.10 MinParent > 2^7 caused underfitting)



(Fig.11)

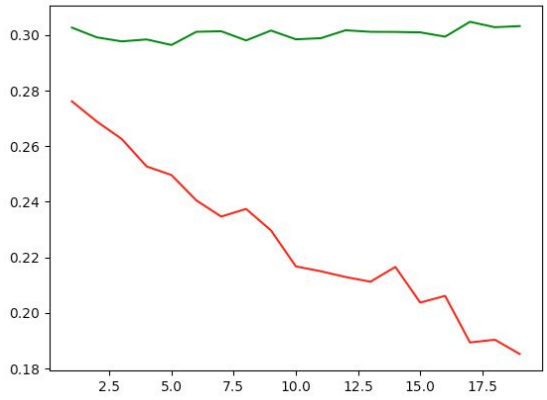
As the number of the learners were increased, MSE seems to decrease (Fig.11). As we'd expected, ensembling technique seems to work much better than a single estimator. Although we observed the trade offs with time complexity, It makes sense that as we increase the number of estimator the performance is better.

Support Vector Machine

SVM provided minimal value to our project. By nature, SVM fit time complexity is more than quadratic which complicates things when our training data could potential be 80,000 values. Also, finding optimal value for penalty and gamma parameters were too computationally consuming; our local machines were not very fond of the intensive calculations.

AdaBoost

AdaBoost is a unique classifier that begins fitting on the original dataset that then fits additional copies of classifiers on the same dataset where the weights are incorrectly classified. This classifier tended to overfit on our training data, but seemed intuitively useful for our final kaggle submission. We plan to blend this with KNN and Logistic Regression. With 100 estimations and a Decision with a max_depth of 10, this classifier produced an accuracy score of 0.67. A standalone decision tree with max depth of 10 produces an accuracy score of 0.69. We definitely recognized overfit occurring, so we decided to compare n_estimators to Training vs Validation Error in Fig.12. A good prediction required 5 estimations.



(Fig. 12 Error vs. N_estimations)

Ensembles

In our attempts to blend classifiers we used the bootstrap aggregation method to train our individual models, then we took the mean of each prediction and tested for accuracy against our validation data.

First Ensemble Attempt (KNN, Logistic Regression, Decision Trees): We decided to use this set of classifiers because we noticed that all presented a similar accuracy score around 0.69. This was our first attempted at a Kaggle submission, we also did not standardize our feature values. We noticed a simple increase of the number of bags lead to a significant jump in AUC score from kaggle.

Number of Bags	Kaggle Score
30	0.73886
50	0.74511

Second Ensemble Attempt (KNN, Logistic Regression, AdaBoost): This time we tried adding Adaboost into our blend instead of a standalone decision tree. We also modified our KNN function to classify based on the 30 closest nearest neighbors. This increase in K was not beneficial, but the addition of an Adaboost classifier did help.

Number of Bags	Kaggle Score
50	0.74876

- Third Ensemble Attempt (KNN, Decision Tree, Gradient Boosting): Individually, KNN scored with an average of 0.69, a decision tree scored an average of 0.69, and Gradient Boosting scored an average of 0.70. All of these individual models scored the highest on testing data, so it would be reasonable to assume they would do well as an ensemble. Again, we are using the bootstrap aggregation method to select specific points to train our classifiers on.

Number of Bags	Kaggle Score
50	0.74588
150 (Sledgehammer attempt)	0.74639
50 (removing KNN)	0.74953
50 (Gradient Boosting increase learners)	0.76048

Final Kaggle Assessment

Ensemble of: Decision Tree, K Nearest Neighbors, Gradient Boosting, Logistic Regression
Ensemble method: Bootstrap Aggregation
Number of Bags: 150 (More would have been better, but limited by time/ computing power)
Parameters:

- Decision Tree: max_depth = , min_leaf_samples =
- KNN: n_neighbors =
- Gradient Boosting: n_estimators = on decision trees
- Logistic Regression: C =
- Adaboost: n_estimators =

Final AUC Score: 0.77243

Conclusion

This project entailed predicting rainfall given a set of 14 features. Initially my team decided to visualize the data in a 2D pair plot to notice any linearly separable instances. Unfortunately, this project was not that easy so we performed Principal Component Analysis to shrink the data into 2/3D graphs to present a new visualization. We avoided feature reduction in our models because we agreed that 14 features was still a manageable feature set and a majority of feature seemed reasonable correlated. When then went about choosing different learner models to see how they would predict on our training data. We eventually settled upon Decision Trees, KNN, Gradient Boosting and Logistic Regression. Gradient Boosting, alone, produced the highest predictive scores at 0.72 accuracy, but required the most computer power/time to calculate. Our finally ensemble consisted of using the bootstrap aggregation method which would take certain data points at random (with replacement), and train our individual models. We would then average the confidence scores from each learner and submit to kaggle. Overall, we did a fairly decent job with AUC score of, 0.76048.

This final project was definitely one of my more enjoyable ones. For the first time, I was able to apply the theories/algorithms I learned in this class to a real world classifying problem. I also enjoyed being able to decide the value of a certain classification model, ie. if it produced high accuracy or if it was under/overfitting my training data. Some of the more obscure lessons I learned in this project is machine learning is still at its youth and the limits with which ML can take us unimaginable. At this point, it seems that ML is mostly limited by computing power, as evident in our final ensemble classification with Gradient Descent. It also seemed common that our highest predictors usually took a large amount of time to compute. Google is slowly solving this issue, allowing GPUS help carry the load with numerous calculations. All in all, this project motivated me to continue my studies in machine learning and I look forward to tackling more challenging projects like this one.

Final Kaggle AUC Score: 0.77243