# CS 273A Introduction to Machine Learning

## Project Report
## Instructor: - Prof. Sameer Singh


**Arpita Huddar 87009145**
**Archana Senthilkumar 12001924**
**Soundarya Soundararajan 54288635**
**Team Name: MLennial**

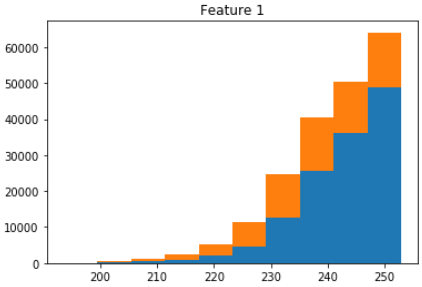
## Introduction

This Machine Learning project worked on data from UC Irvine Center for Hydrometeorology and Remote Sensing in order to predict Rainfall in a given location. We focussed most of our efforts in this project towards Data Analysis, Preprocessing and Feature Selection. Using the information gained from these steps we worked extensively with Decision trees in Ensembles like Bagging, Boosting and Random Forest learners as well as SVMs analysing different kernels.

## Preprocessing of Data

### Histogram Analysis

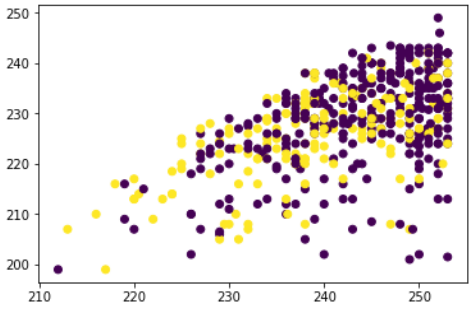
We plotted each of the features in Histograms and obtained the following graphs.




All feature graphs are analysed from the Appendix. We observed that in all features, all ranges have 80-90% no rain (blue) and 10-20% (orange) rain data classifications.

### Scatter Plot Analysis

Below, is a plot of Feature 1 versus Feature 2 in a scatter plot. We plotted many different combinations of the different features in scatter plots and observed that mostly, although there were some interesting differences in the relation between some features, in terms of classification, the data points and features were non-separable and that linear regression methods would likely not give good results on their own.




Feature 1 vs Feature 2

### Marking missing values:

The competition data based on satellite-based measurements could have missing data values. In such a case the learning algorithm might include the results of the missing values while predicting and we could end up with skewed results. The below output gives the number of data points that have a zero value and

the zero values are replaced with the Mean of the remaining values in each feature of the dataset. There were almost half of the number of features that had missing values. The AUC obtained with the transformed dataset with decision tree was 0.536.

**Feature Selection**

On analysing the data and plotting histograms of the distribution of the data, we found that many features seemed to have similar data distributions and collinear data points or with very low variance. Hence we decided that we could try and reduce the number of features if certain features do not contribute as much data.

**Chi 2**

Chi-squared tests can be used to calculate the correlation between the non-negative features. We used this to test the dependence between the variables, thereby removing independent features. With the various results we tried selecting between 5 and 12 features and training on 5000 data points, we found that the highest training AUC score on our basic Decision tree was obtained when selecting 8 features out of 14. We selected 1,4,5,7,8,9,10,and 13th features respectively as a result of this feature elimination, and trained the Bagging Ensembles.

**Recursive Feature Elimination**

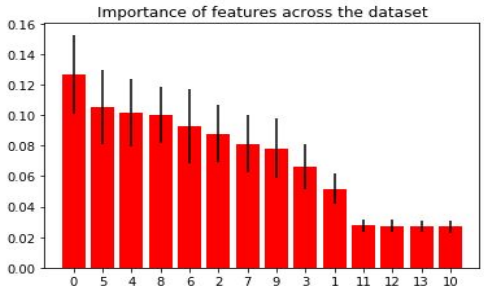
Recursive Feature Elimination selects features by recursively by reducing the size of the features it is sampling iteratively. Initially, the function is trained on all the features, and the importance of each feature is obtained by calculating feature importance attributes. By this score, the least important features are removed. This is then repeated until the required number of features is obtained.

We selected 8 features with this method from Scikit learner and the features returned were, 1,2,3,4,8,9,10 and 11th features.

**Tree based feature selection:**

The tree based estimators are used to identify the best set of features that can used with the data model. This would eliminate irrelevant data features and with the turn can be used to discard irrelevant features. This implements an estimator that fits a multiple random decision trees on a number of sub-samples from the dataset and average to improve the accuracy. This shows that the features from 0 to 9 have higher estimates compared to the other data points. This correlates to a certain extent with the number of missing data in the feature set.

```
In [26]: runfile('C:/Users/Soundarya/tree1-selection.py', wdir='C:/Users/Soundarya')
```



## Learners

### Decision Tree

We decided to pursue Decision trees, as they trained on the data quickly and there were many options for selecting features and training parameters. Similar to the analysis done in Assignment 4, we completed a basic trend analysis across the maximum depth of the tree, the minParent and minLeaf values. Finally we decided on the following parameters: -
Maximum Depth=12, minParent=16, and minLeaf=2. This was one of our first project submissions to Kaggle and got an AUC score of 0.71735.

### Bagging with Decision tree

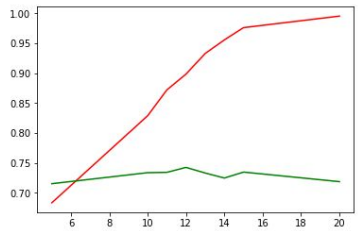
We decided to continue with Decision trees and explore how they could be used within Ensembles. The first one we decided to implement was Bagging. We wanted to combine and work with more Decision tree parameters. Bagging with Bootstrap aggregation is the method of training models on multiple subsets of the data, and take an average of the final prediction. To ensure that we weren't overfitting the data, we set our maxDepth to 12 and min_samples_split and leaf values to 16 and 2 respectively .

Training AUC 0.99809927741
Validation AUC 0.742531194296

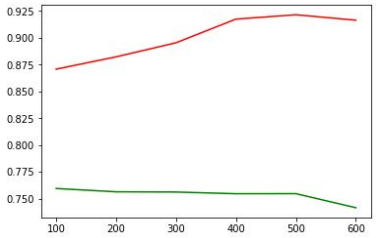
### Adaboost Decision Tree

The next model we tried was using Adaptive Boosting algorithm. Adaboost machine learners are most commonly used with Decision Trees and worked well with our progress thus far. We used Scikit Learners AdaBoost learning model. The parameter n_estimator can be set to determine the number of boosts in such a way that the model does not overfit the data either.

Keeping MaxDepth as a constant 12, we varied the number of boosts between 100 and 600. we trained the Decision Tree Regressor and plotted the Training and Test AUCs against the number of estimators to determine the best value.




Training and Test AUC vs Max Depth




Training and Test AUC vs Number of Estimators (boosts)
Red - Train AUC, Green - Test AUC

Finally with the Parameters maxDepth =12 and Number of estimators = 300 which had an AUC score of 0.762. Our final Adaboost Decision Tree submission was submitted by Archana (MLennial) and we obtained a score of 0.71.

Additionally, we also tried with the selected features from Chi 2 and RFE methods, and since RFE provided a much higher AUC, we repeated the Adaboost Decision tree with the same parameters, 10 selected features and made a submission to Kaggle expecting a better score.
However, we did not get a an improved score despite having trained on all the data and getting an AUC of 0.79 on our validation data before submission.

**Bagging the Boost Regressor**

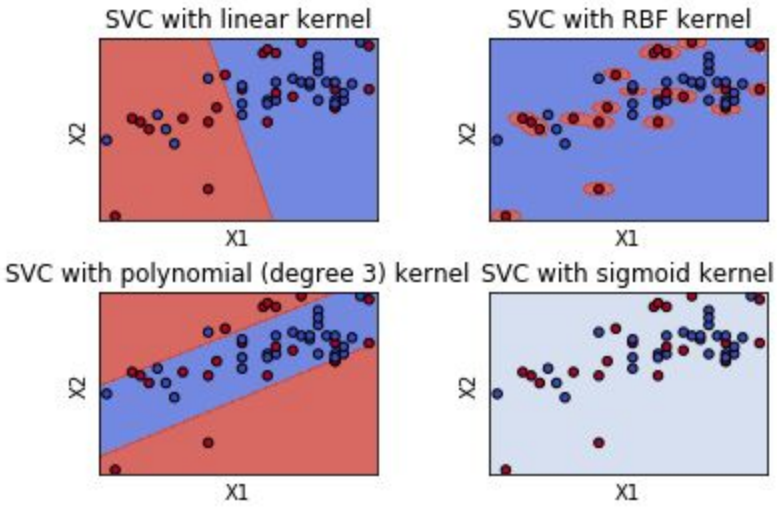
We knew that Boosting is especially useful in improving weak learners and combining them appropriately, and once we obtained a reasonable result with our Adaboost predictions, we wanted to test if we could stack the Bagging model on top of the Boost learner to improve our results. We also spent a considerable amount of time trying to get the best parameters by repeating the exercise of tuning the parameters for our Bagging learner on top of our Boost ensemble. Since boosting reduces bias and also reduces the variance of the model, we believed that might be improved by the cross validation and majority voting of the Bagging model. We varied the number of estimators as well as the min samples split and leaf parameters. This resulted in a slight improvement of the AUC on validation data, but a much lower score on Kaggle. Based on the possibility that the model is underfitting the data.

**Random Forest**

With decision tree implemented and an AUC value of 7.1 and parameters being max_depth =20, min_parent=16, min_leaf=5, random forest was considered to fit a number of decision tree classifiers on multiple sub-samples of the dataset and finally using mean to improve the accuracy of the model as well as control over-fitting. The parameter set also has the support for replacement with bootstrap set as true.
We have tried varying the values of max_depth, min_samples_leaf and min_samples_split and got best results for max_depth of 30, min_samples_leaf = 2 and min_samples_split = 5. AUC score obtained on validation data is 0.7513. AUC score on kaggle for test data is 0.73671.

**SVM**
Support Vector Machines are effective with high dimensional data. Given the dataset to predict has 14 data features, SVM could provide a better AUC. The general idea was to incorporate the high dimensional dataset by maximizing the functional margin with a construction of a hyperplane that has largest distance from the training data points. Support Vector Machine implemented with Scikit-learn internally uses Libsvm for prediction.

**Comparison between Kernels for our dataset:**
The kernel considered for implementation is the Gaussian kernel. The 2D plot represented does below, has the dataset implemented with 4 different types of kernel. From the plot it is shows that the data is not linearly separable. The linear kernel is the most significant one as it faster to train in comparison to any other kernels, but the data is not linearly separable in this case. The polynomial function with degree 3, does not give a proper classifier. Even with the increase in the polynomial the results were not effective, which was the same with a sigmoid kernel. The kernel chosen for the given dataset is the Gaussian kernel is that gives a better classifier compared to any of the 3 kernels that are implemented.

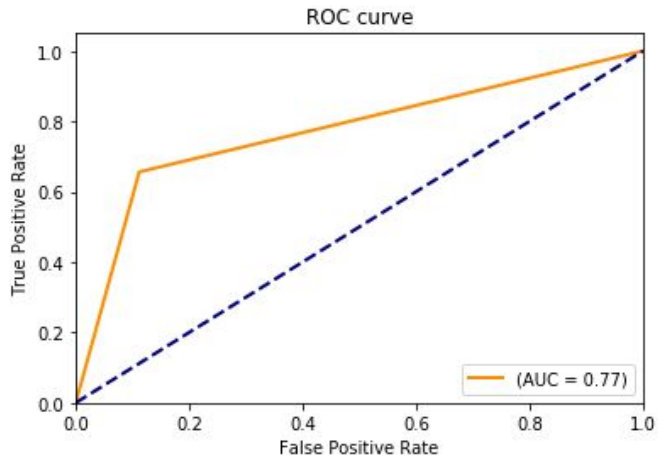



**Gaussian Kernel with the Parameters:**

**C:** the penalty incurred with the error term. This is a trade off between the misclassification of training examples against simplicity of the decision boundary. When the value C is low, the decision surface smooth and with a high C, the model attempts to classify all training examples correctly. The gamma value considered was across a span of 1 to 15 and the chosen value implemented for the given data set was 10 as the model was to close to predicting all the validation data points effectively.
**Gamma :** the kernel coefficient for the specified kernel implemented. When the value of Gamma is high, the samples other than the ones closer to the boundary would be impacted. With the sequence of gamma values, the accuracy score that was evaluated did not improve and the chosen value was 2.

**XGBoost**
We decided to try XGBoost as its fast and gives good results. XGBoost is short for eXtreme gradient boosting. It is a library designed and optimized for boosted tree algorithms. It supports regression, classification, ranking and user defined objectives.

We created Xgboost specific Dmatrix data format from the numpy array. Xgboost can work with numpy arrays directly or we can load data from svmlight files and other formats. We tried Xgboost with numpy arrays directly and also the svmlight file format. As we realized from initial data analysis, that there are lot of zero values in features we decided to use dump_svm_light file format for XGBoost as It does not store zero valued features. Svm file format gave better results.
We set the objective parameter to multi:softprob. As a result of this, the output will be predicted probability of each data point belonging to each class. Then we considered the first column in the output as it corresponds to the probability of rainfall occurring in our case.
We varied eta (the training step for each iteration) , max_depth, number of boosting iterations. And the values which gave us the best results are eta = 0.1, max_depth = 20, number_of_iterations = 20.
We got an auc of 7.7 when we tested it on some part of the training data given to us.




Auc Values we got on kaggle after uploading the predictions of X_test is around 7.6 as seen below.

**Semi-Supervised Learning:**

**Expectation MAximization with Gaussian Mixture Model:**

This method is used for binary classification with Gaussian Mixture Model with Maximum Likelihood estimation. This involve a segment of labeled and unlabeled data and uses Naive Bayesian model to classify the data.

```
In [34]: runfile('C:/Users/Soundarya/untitled3.py', wdir='C:/Users/Soundarya')
AUC for test train with GAUSSIAN MIXTURE MODEL
roc_auc_score for testing data 0.582973145899

In [35]:
```

Performance Comparison:

| Model | AUC on Kaggle | Submission File Name | Account username |
|---|---|---|---|
| Decision Tree- marked missing values | 0.55349 | Y_submit.txt | soundas1 |
| SVM | 0.64642 | Y_submitmeanzero.txt | soundas1 |
| Bagging and Boost Regressor | 0.65402 | Y_submitbag and boost.txt | asenthil |
| Bagging with Decision Tree | 0.71085 | Y_submitselectedbagging.txt | ahuddar |
| Adaboost | 0.7147 | Y_submitAdaboost12Alldata.txt | asenthil |
| Random Forests | 0.73671 | Y_submit_ful_randomForest.txt | ahuddar |
| XGBoost | 0.76870 | Y_submit_ful_data_numpy_xgb.txt | ahuddar |

**Statement of Collaboration**

We all continuously collaborated and were in discussion about how to proceed with a particular model and how to fine tune parameters. We all worked on feature selection and preprocessing of data. Archana and Arpita drove the discussion related to Decision trees, Bagging, Boost ensembles. Arpita and Soundarya drove the results obtained in Random forests and XgBoost, and Soundarya and Archana drove the discussions on SVMs and Semi supervised learning.

## References:


[1]https://www.researchgate.net/publication/268439113_Prediction_of_Rainfall_Using_Support_Vector_Machine_and_Relevance_Vector_Machine

[2]http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

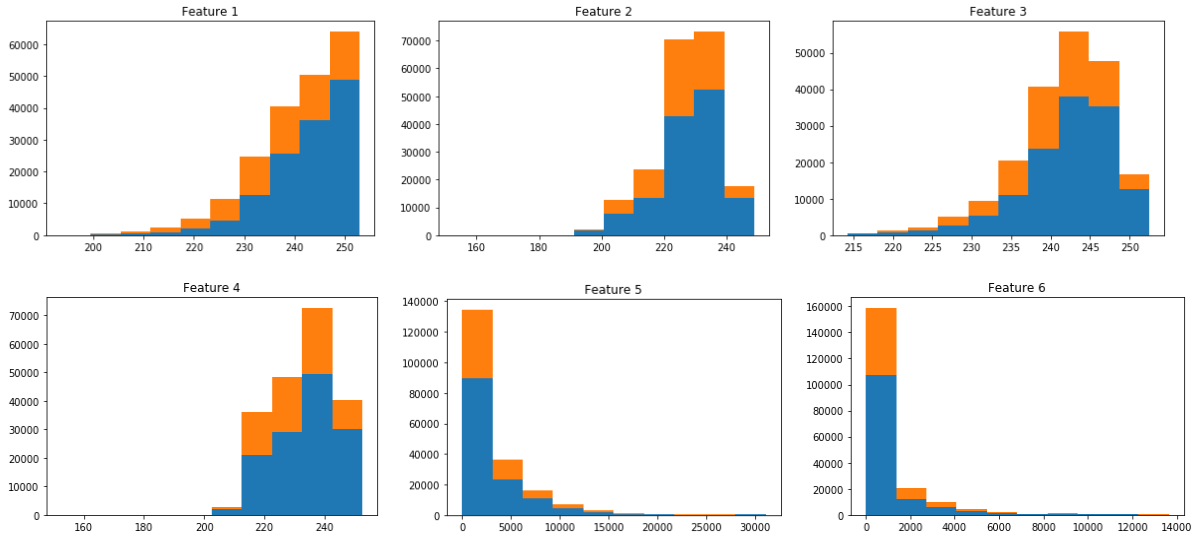
[3]Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

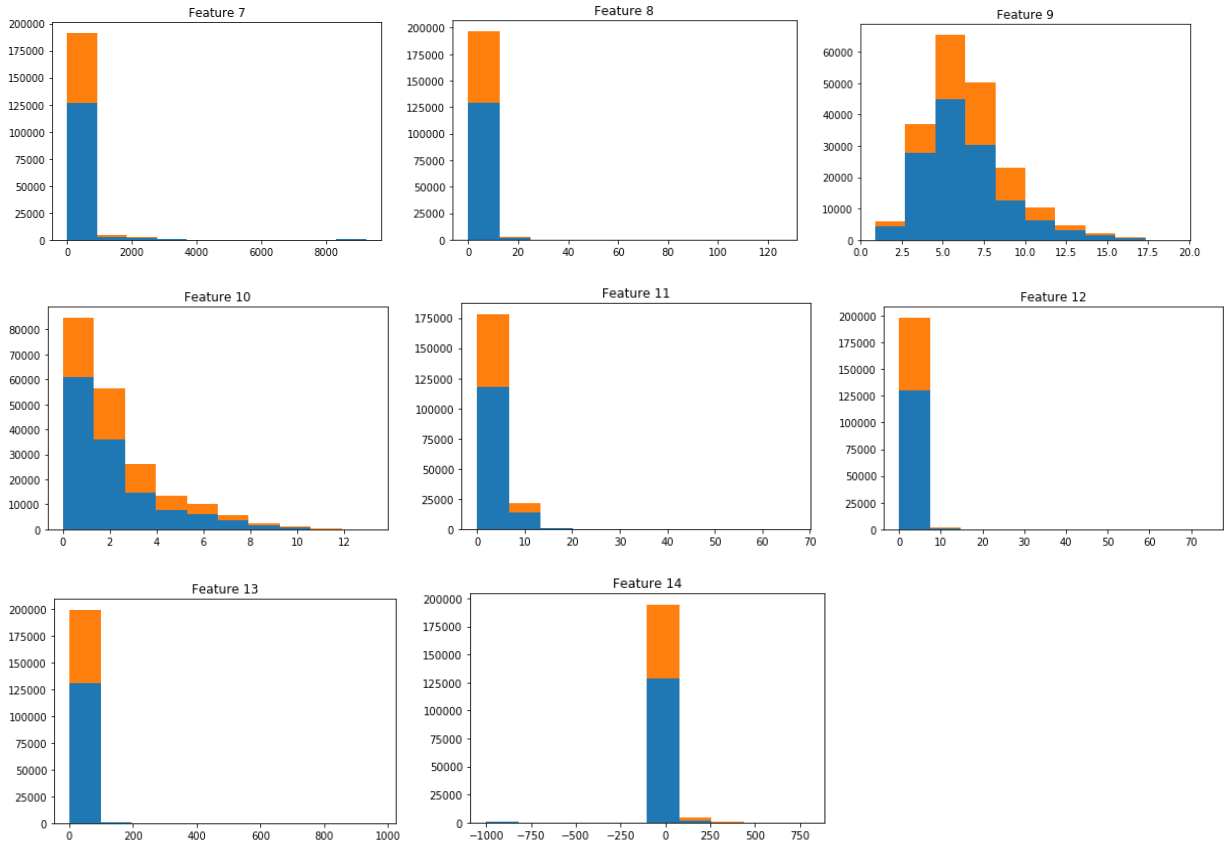
[4]https://www.kdnuggets.com/2017/03/simple-xgboost-tutorial-iris-dataset.html

[5]https://github.com/dmlc/xgboost/blob/master/doc/parameter.m d


## Appendix:

Pre-processing data plots:






Data Features and Marking Mean Value:

```
[0, 0, 0, 0, 0, 40409, 161255, 2488, 0, 47880, 2488, 2488, 0, 147909]
Feature 13
104639 43270
2.54958476285 20.5572012942
Feature 11
1713 775
200.112868342 252.323028335
Feature 10
1713 775
322.509334367 375.578828516
Feature 9
35769 12111
7.21340348098 13.327685161
Feature 7
1713 775
258.355328091 267.294437471
Feature 6
108556 52699
163.018340764 188.279606824
Feature 5
30293 10116
3566.32426633 7672.71085409
```