# CS 273A Machine Learning - Project Report

RAHUL SRIDHAR
rsridha2@uci.edu
ID: 41608676

PRIYANKA RAVI
priyanr1@uci.edu
ID: 33246700

December 7, 2016

## I. PROBLEM

Predict whether there is rainfall at a location based on (processed) infrared satellite image information. The dataset is courtesy of UC Irvine's Center for Hydrometeorology and Remote Sensing, including Dr. Soroosh Sorooshian, Dr. Xiaogang Gao, Dr. Kuo-lin Hsu, Dan Braithwaite, Yumeng Tau, and Negar Karbalee.

## II. UNDERSTANDING THE DATA

Initially, we wrote several helper functions to understand the dataset better and speed up our process. These included functions for printing the summary statistics of the data, visualizing the relationship between features and the target and other features, performing sequential data sampling, balanced sampling and plotting errors as a function of different parameters.

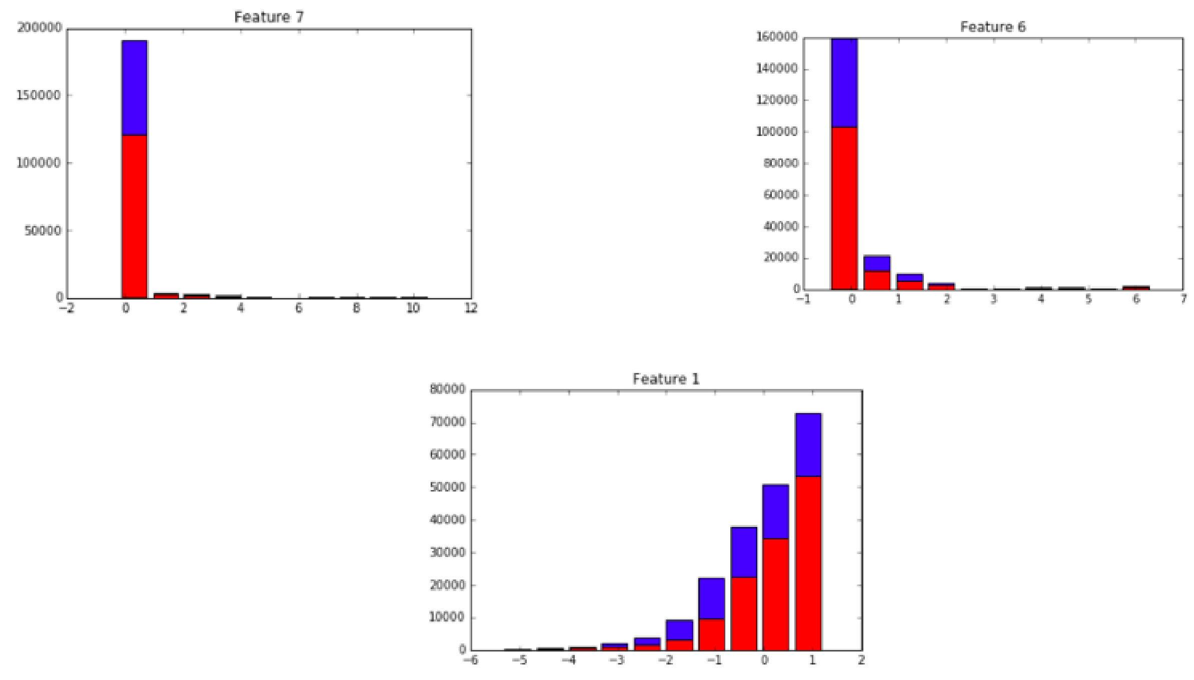


**Figure 1:** *Feature-target distribution (post scaling)*

We observed that the data corresponding to features 6, 7, 13, 14 had a very large number of zeros in the data distribution. Figure 1 shows the distribution of values for features 6 and 7. This can be compared with Feature 1 which has a lesser proportion of zeros in its distribution.

The target label Y is distributed as follows:

- Number of rows with Y = 0 is 1,26,551
- Number of rows with Y = 1 is 73,449

We noticed that the data is slightly imbalanced – the number of data points with Y = 0 is 72% more than the data points with Y = 1

- The imbalance may just be a natural characteristic of the domain from where the dataset was obtained. So, we should be careful in correcting for this imbalance and not end up "tricking" the learner to believe that there is an equal probability for 0s or 1s in the world from which this data was obtained. Also, given the size of the dataset, this imbalance might not be a huge issue.
- Having said that, we should not completely ignore it either. We hence inferred that we should handle it by weighting the class Y = 1 by around 1 to 1.5 times the weight of class Y = 0 in the learners we build (wherever necessary). We also built models by under-sampling data points with Y = 0, but these underperformed on the validation data.

By plotting stacked bar plots to understand the distribution of values and targets in each feature, pair plots to infer the relationship of features with themselves and scatter plots to understand their relationship with the target, we observed a few more things:

- Target labels were more or less uniformly distributed with respect to each feature
- Certain features had a disproportionately high number of zeros
- Creating new features by combining existing features might not be very helpful in increasing the separability of the classes

Our findings from this phase primarily aided in the feature transformation and selection process during model building.

In the next section, we briefly describe our model building methodology.

## III. Methodology

At a high level, our approach for the model building exercise involved four phases. We discuss the finer details of each phase in subsequent sections:

- Phase 1: Quick prototyping of different models on subsets of data (using Weka)
- Phase 2: Elaborate model building on a subset of models (chosen from Phase 1 based on their performance). This phase primarily involves feature transformation, data sampling, feature sampling, parameter tuning and model refinement, some of which was dictated through our Data Understanding phase and some of which was decided based on error visualization and detecting under/overfitting (using scikit-learn and mltools)
- Phase 3: Model building on the entire set of data (using GraphLab, for efficiency purposes)
- Phase 4: Building an ensemble learner based on the model predictions from Phase 3

We now summarize the various experiments we performed and the results obtained from them.

## IV. Experiments and Results

### Phase 1 (using Weka)

We built Naïve Bayes, Decision Trees, Random Forests, Logistic Regression and Sequential Minimal Optimization models on subsets of the dataset

- Naïve Bayes was omitted from future analysis due to its relatively poor performance

We computed the Information Gain of the features, which guided us in performing feature selection for certain learners

- Features in decreasing order of Information Gain:

  1, 14, 9, 3, 13, 4, 10, 2, 12, 6, 11, 7, 8, 5

### Phase 2 (using scikit-learn and mltools)

We built models for SVM, Logistic Regression, Decision Trees and Random Forests using the mltools and scikit-learn libraries (mainly for more powerful parameter tuning). The results are summarized in Figure 2. Some of the common steps we followed for all the models are listed below:

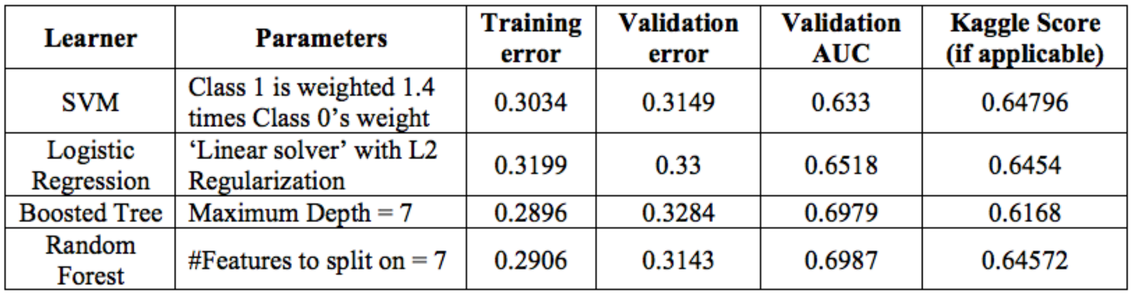
- Experimented with feature selection based on their information gain and certain feature transformations based on visualizing the relationship between features and the target.
  - We experimented with various feature transformations, namely exponential, log, sine, and polynomial transforms of individual features and visualized their relationship with Y. This also included building models with additional features such as (X7*X7 to weight non-zero values in X7), (X2-X3 to see if the difference of the distributed non-zero values in both improved the separability of the data). There were a few transforms that improved the performance of models minutely, but we felt the additional performance boost did not justify the increased model complexity introduced by including the transformed features and hence worked with the existing feature set.
- Different models were compared based on their performance on training and validation data (error rates, ROC and AUC) and in a few cases, using their Kaggle scores

SVM:

- Initially, we built an SVM model using a rbf kernel on a subset of the data. But due to the size of the data and consequently, the slow running time, which would be further detrimental when building on the entire 200k data, we decided to switch to linear SVM
- Class weights for Y = 1 (with respect to Y = 0) were varied in the range [1, 1.5]. The model with weight = 1.4 performed best. Lowering or increasing the weights led to severe under-performance on both the training and validation data
- We also performed clustering on the entire data set using k-means and EM algorithms (with k = 2 initially, to reflect the number of classes in the target, but later increased up to k = 5) and included the cluster assignments as a feature to the SVM model. Again, this gave an insignificant improvement and we did not find the feature valuable enough to include in further analysis

Logistic Regression

- Logistic regression typically works pretty well even if the data is imbalanced
- The bare-bones version of it tended to overfit the data and also predict the majority class 0 predominantly. So, we decided to perform L2 regularization and increased the L2 penalty parameter. This improved the model performance by a good margin



| Learner | Parameters | Training error | Validation error | Validation AUC | Kaggle Score (if applicable) |
|---|---|---|---|---|---|
| SVM | Class 1 is weighted 1.4 times Class 0's weight | 0.3034 | 0.3149 | 0.633 | 0.64796 |
| Logistic Regression | 'Linear solver' with L2 Regularization | 0.3199 | 0.33 | 0.6518 | 0.6454 |
| Boosted Tree | Maximum Depth = 7 | 0.2896 | 0.3284 | 0.6979 | 0.6168 |
| Random Forest | #Features to split on = 7 | 0.2906 | 0.3143 | 0.6987 | 0.64572 |

**Figure 2:** *Results from Phase 2 (Dataset size: Training = 20,000; Validation = 10,000)*

Decision Trees

- Using training/validation data splits and cross validation, we compared different decision trees (including a simple decision stump) by changing the split criterion (gain/entropy).
- On observing increased error rates in the validation data (signs of overfitting), we tried to control the model complexity by varying min_samples_split, min_samples_leaf and max_depth
- We decided to build a boosted version of the decision tree and check for further improvement in model performance. The number of learners used was varied accordingly to find a model that struck a balance between under and overfitting. Increasing the number of learners beyond a point led to severe overfitting

Random Forests

- We built a model that doesn't underfit/overfit the data by varying the different parameter settings of nFeatures, minLeaf and minSplit. As expected, having very low values of minLeaf and minSplit led to overfitting on the data and conversely, very high values led to underfitting due to the low model complexity
- We built an ensemble of the decision trees by bootstrapping data using different number of learners ranging from 5 to 20 to find the model that performs best

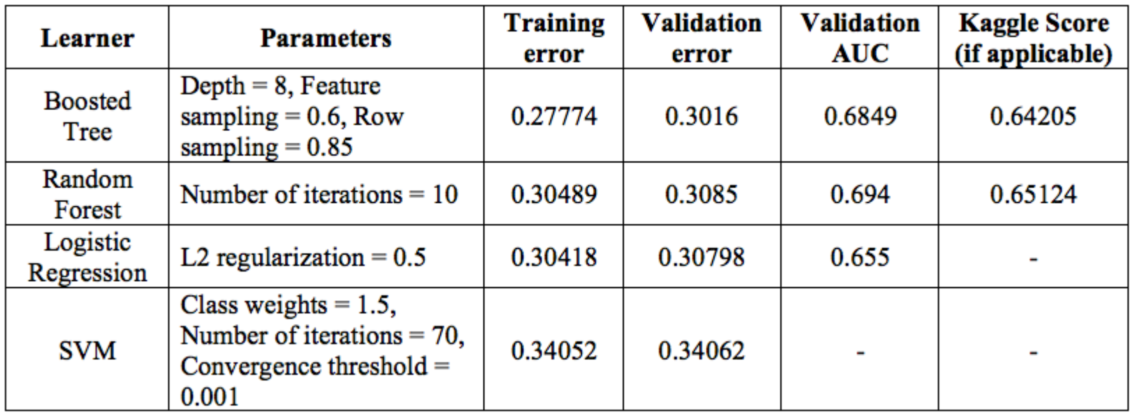
The primary objective of this phase was to test various models on subsets of the training data to quickly identify the types of models which we should build the final learner on. At the end of this phase, we realized that a stack of learners may do a better job than a single learner (based on the results obtained and Kaggle scores).

## Phase 3 (using GraphLab)

Up until now, the models were built with subsets of the entire data set. With the notion that more data may improve performance of the chosen models, we used GraphLab for fast model building on the entire data set.

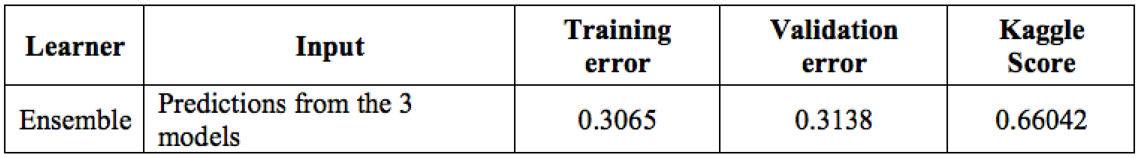
As the size of the dataset used to build the model increased in this phase, the parameter settings of the individual learners had to be re-tuned accordingly. This was performed in a similar fashion to Phase 1. In all the four models chosen at the end of Phase 2, we noticed a common tendency for the model to overfit and predict majority class 0. Keeping in mind the bigger picture to build an ensemble of these learners, we imposed several complexity measures on these models. The performance of each of these is shown in Figure 3 along with their complexity controls.

Although each of the individual learners could do a 'decent' job at predicting on the test data, they may still be looked at as a group of weak learners on which an ensemble model can be built.



| Learner | Parameters | Training error | Validation error | Validation AUC | Kaggle Score (if applicable) |
|---|---|---|---|---|---|
| Boosted Tree | Depth = 8, Feature sampling = 0.6, Row sampling = 0.85 | 0.27774 | 0.3016 | 0.6849 | 0.64205 |
| Random Forest | Number of iterations = 10 | 0.30489 | 0.3085 | 0.694 | 0.65124 |
| Logistic Regression | L2 regularization = 0.5 | 0.30418 | 0.30798 | 0.655 | - |
| SVM | Class weights = 1.5, Number of iterations = 70, Convergence threshold = 0.001 | 0.34052 | 0.34062 | - | - |

**Figure 3:** *Results from Phase 3 (Dataset size: Training = 150,000; Validation = 50,000)*



| Learner | Input | Training error | Validation error | Kaggle Score |
|---|---|---|---|---|
| Ensemble | Predictions from the 3 models | 0.3065 | 0.3138 | 0.66042 |

**Figure 4:** *Ensemble model results*

Subsequently, we built an ensemble learner on these model predictions, the details of which are given in the next phase.

## Phase 4

With the four models and their predictions, we could build an ensemble to make the final prediction by either:

1. Predicting the mean of these values, or
2. Building another model on these values

Considering the complexity of the underlying models, we did not want to build another complex model on top of this and hence decided to build a Naïve Bayes model on these values. Given that each of the predictions are conditionally independent given the class, we felt this would be a good model to choose as the assumption behind it would not be violated. The final results are shown in Figure 4. The performance of this model on the test data (Kaggle) was the best out of all the learners tested on Kaggle.

- We also observed that ensembling by averaging the predictions fared relatively poorly as compared to building a Naïve Bayes model on the predictions from the four models
- The ensemble seemed to perform better on the validation data without SVM and hence we decided to eliminate SVM from being one of the underlying models the ensemble is built on. Given how different its training and test errors are from the other models, this was to be expected (shown in Figure 3).

Thus, the final learner built was an ensemble learner that was a Naïve Bayes model over a layer of Boosted Trees, Random Forests and Logistic Regression models. A holistic view of the entire process can be seen in Figure 5.

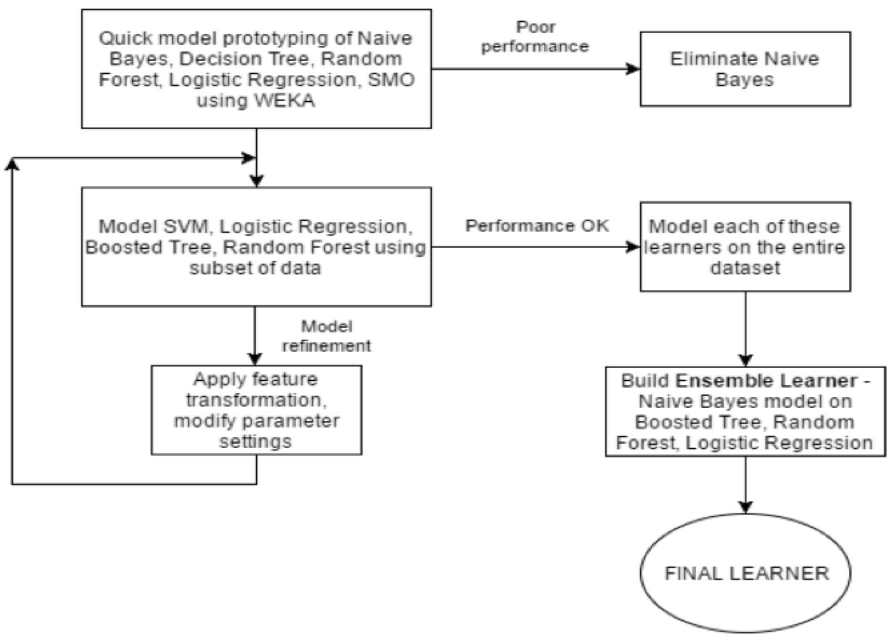

Quick model prototyping of Naive Bayes, Decision Tree, Random Forest, Logistic Regression, SMO using WEKA
Poor performance
Eliminate Naive Bayes
Model SVM, Logistic Regression, Boosted Tree, Random Forest using subset of data
Performance OK
Model each of these learners on the entire dataset
Model refinement
Apply feature transformation, modify parameter settings
Build Ensemble Learner - Naive Bayes model on Boosted Tree, Random Forest, Logistic Regression
FINAL LEARNER

**Figure 5:** *Holistic view of the model building process*

## V. Conclusion

Building a learner in machine learning involves a large amount of work trying to understand the data, how the features interact with each other and with the target while being able to modify various parameters of the models themselves to achieve better performance.

Every step performed in the model building process should be soundly reasoned. We could choose to build numerous models, each of a different kind. The key is to constantly try to improve them while being sure not to underfit or overfit. We also have to make trade-offs with respect to complexity and interpretability every step of the way.

Even though a lot of the steps one would take in the process would be reactive (in terms of changing the models based on how they perform on the data), it certainly helps to have an overarching, structured methodology in place upfront.

## VI. Team Contribution

Both of us were equally involved throughout all phases of the project. We initially chalked out the methodology for building the models and then split the tasks in each of the phases as follows:

- Phase 1: Work together to narrow down the set of models to be tested in subsequent phases
- Phase 2: Using subsets of the data, Priyanka experimented with Decision Trees and Random Forests, while Rahul experimented with SVM and Logistic Regression
- Phases 3 and 4: Each of us built the corresponding learners in GraphLab with the entire dataset and worked together to build the final ensemble learner