Machine Learning Project Report

Adithya Bhat
bhat5@wisc.edu

Srinivasan Ravichandran
sravichandr2@wisc.edu

Abstract

In this project, we have tackled a *binary classification problem* of predicting if a person makes over $50,000 a year using census data from the UCI repository (originally from the US Census Bureau). We have used Support Vector Machine, Decision Tree, k-Nearest Neighbor and the Naive-Bayes algorithms for this task. In addition to this, we have also made use of the ensemble methods - Random forests and AdaBoost. Finally, we compare the performance of each of these algorithms with the help of precision-recall curves.

1 Introduction

Our focus in this project was to gain experience in taking up and solving a classification problem. After obtaining a dataset, as a first step, we focused on pre-processing data and taking care of missing values. Following that, we tried throwing the classic machine learning algorithms (Naive Bayes, k-NN, decision trees, SVM) at our problem. In the process, we investigated the effect of varying the parameters of each of the algorithms on the precision and recall. We also examined feature selection and its impact on the precision and recall. Later on, in order to improve the performance, we tried using ensemble methods, specifically Random Forests and AdaBoost. In this report, we list all the interesting observations from our project.

2 Problem and Dataset Information

We have made use of census data from the UCI repository (originally from the US Census Bureau). The problem we address is a *binary classification problem* to predict if a person makes over $50,000 per year with the help of additional information about said person. This includes information such as age, sex, race, origin, type of employment etc. The description of the data set is as follows.

- Number of attributes: 39
- Number of training instances: About 200,000

- Number of test instances: About 100,000
- Number of positive instances in training set: 11607 (6.51%)
- Number of positive instances in test set: 6186 (6.61%)

We have used Support Vector Machine, Decision Tree, k-Nearest Neighbor and the Naive-Bayes algorithms for the classification task making use of the `scikit-learn` python package.

3 Pre-processing

We considered R and Python as our options for the project. After some experimentation, we found that while R was easier for statistics and analysis of the data, the lack of uniformity among the various ML packages made Python our preference.

The ML algorithms provided by the `scikit-learn` package do not function if the input data has missing values. Hence we either had to impute data at the missing slots or remove the instances that had these missing fields. Upon examining the data, we realized that 4 of the 39 features had missing values for a majority of the instances. Additionally, the statistical correlation between those features (for instances where they were available) and the Class label appeared to be nearly non-existent. Hence, we dropped those features.

In addition to this, sampling was necessary for certain algorithms where the machines at our disposal were not sufficiently powerful to efficiently process the entire sets. We performed stratified sampling to the maximum feasible ratio.

We performed $one-of-k$ encoding for the categorical features where required. Normalization of the numerical features was also performed.

4 Learning Algorithms

In this section, we discuss the stand-alone learning algorithms that we used and the variations of parameters that we tried in each case. In Section 6, we describe our observations in detail.

4.1 Decision Tree

`scikit-learn`, the Python package that we used, implements an optimised version of the CART algorithm. The parameters that we tried to vary were the following.

- `max_leaf_nodes` - The maximum number of leaf nodes allowed in the tree.
- `min_split` - The minimum number of instances at a node to consider splitting it.

It is worth noting that the two parameters are related and cannot be arbitrarily set independently.

4.2 Support Vector Machine

- `kernel` - The kernel used for the decision function

In addition to this, we also used feature selection to prune the perceived less-relevant features.

4.3 k-Nearest Neighbors

The following were the parameters that we investigated for k-NN.

- The parameter k - the number of neighbors to consider
- The sample size

Although `scikit-learn` does not provide a way to vary sample size, we implemented training with different sample sizes with stratified sampling. Since k-NN is a lazy learner, we expected the training size to be of a reasonable impact on the performance of the algorithm.

4.4 Naive Bayes

We were interested in the accuracy of Naive Bayes, but observed that the precision of Naive Bayes deteriorated rapidly with increasing recall.

5 Ensemble Methods

In addition to the above mentioned learning algorithms, we also made use of ensemble methods in order to improve the results. Specifically, we used random forests and Adaboost to solve the classification problem.

5.1 Random Forest

We investigated the impact of the following parameters on our precision and recall.

- `num_estimators` - The total number of decision tree estimators in the ensemble
- `max_leaf_nodes` - Similar to decision tree - the maximum number of leaf nodes in each estimator in the random forest.

5.2 AdaBoost

In the case of AdaBoost, we were able to achieve a much better running time and we investigated the variations in the following parameters.

- `num_estimators` - The total number of estimators in the ensemble
- `learning_rate` - Learning rate of the algorithm

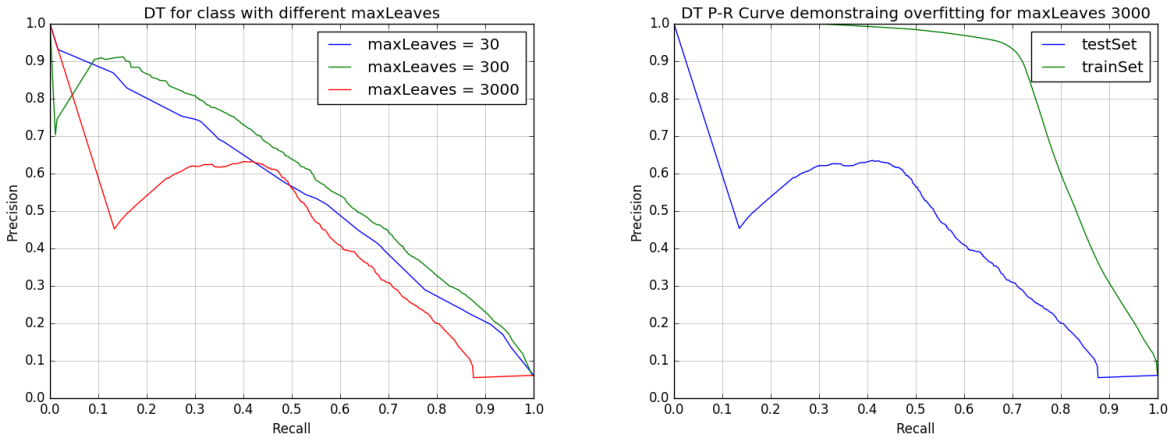

Figure 1: Precision-Recall curves for decision trees. In the first figure, we have plotted the precision-recall for the test set using decision trees with `max_leaf_nodes`=30, 300 and 3000. In the second plot, we have plotted the training and test set precision-recall curves for a decision tree with `max_leaf_nodes`=3000

6 Evaluation

We now present the results that we obtained on trying the aforementioned algorithms. In order to quantify the performance, we chose to plot Precision-Recall curves for each of the cases.

6.1 Decision Tree

We observed that changing the `max_leaf_nodes` had a significant impact on the P-R curve without much effect on the actual running time of the algorithm.

We observed that when the `max_leaf_nodes` is in the order of the number of features in the transformed data, we obtain a good precision recall curve and it degrades when we move away in either direction (i.e.) increase or decrease `max_leaf_nodes`.

Also, as the number of `max_leaf_nodes` grew, we observed very strong *overfitting*. This is to be expected in the case when the number of leaves is very much larger than the actual number of features, leading to the decision tree to split based just on the training set. These results are shown in Figure 1

6.2 Support Vector Machine

SVM training was computationally much costlier than we expected. We trained an SVM classifier with a RBF (Radial Basis Function) and a linear kernel and observed a better

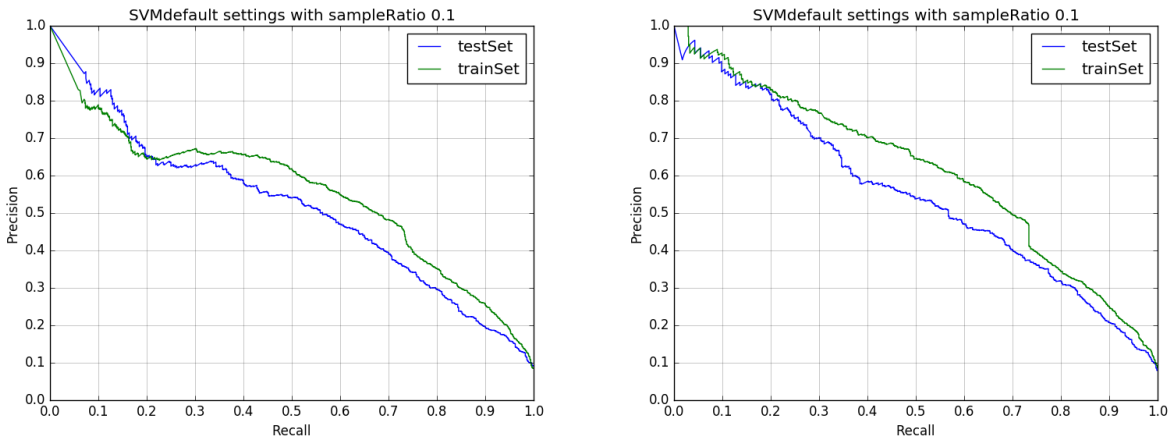

Figure 2: Precision-Recall curves for SVM. In the first figure, we have plotted the precision-recall for the test set using SVM with RBF kernel. In the second plot, we have plotted the training and test set precision-recall curves for a SVM classifier with linear kernel.

precision-recall curve near lower recall ranges in the latter case. The results are shown in Figure 2.

In addition to changing kernels, we also made use of feature selection modules from `scikit-learn`. Specifically, we made use of the `SelectKBest` feature selector which scores the features and selects the top K features for training. We observed that feature pruning lead to a loss of information leading to poor results. The lower the value of K, the lower the performance. This is shown in Figure 3.

6.3 k-Nearest Neighbors

Since there is no thresholding in k-NN, the P-R curves will be mere points. We also used a variation of k-NN which weighed each instance the same (an unweighted k-NN). In the following figure, we show the P-R curves for different values of k and also the curve for the unweighted version.

6.4 Naive Bayes

We observed that Naive Bayes did not perform very well for this problem. Due to space constraints, we are not able to include the precision recall curve. However, we would like to note that the Naive Bayes classifier exhibited a condition where increasing precision almost linearly decreased recall and vice versa.

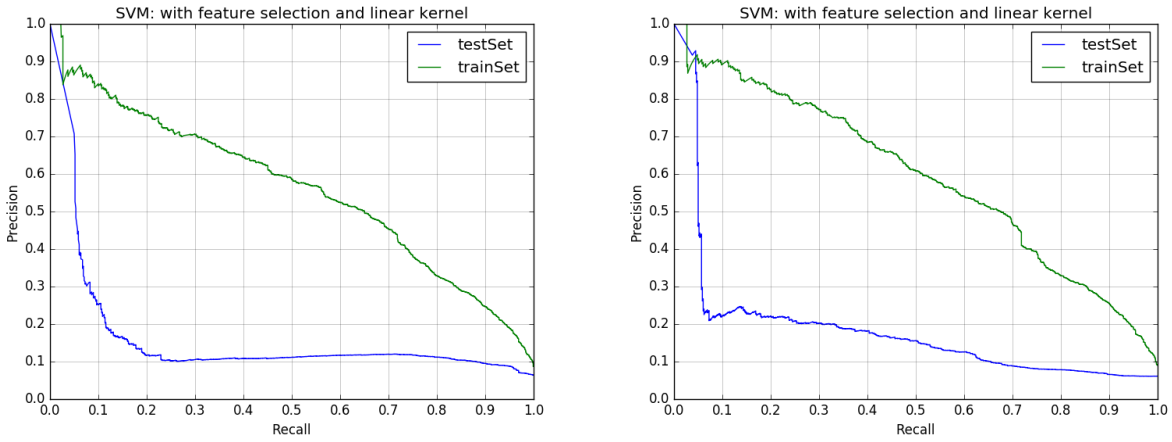

Figure 3: Precision-Recall curves for SVM. In the first figure, we have plotted the precision-recall for the test set using SVM with 100 features selected. In the second plot, we have plotted the training and test set precision-recall curves for a SVM classifier with 200 features selected.

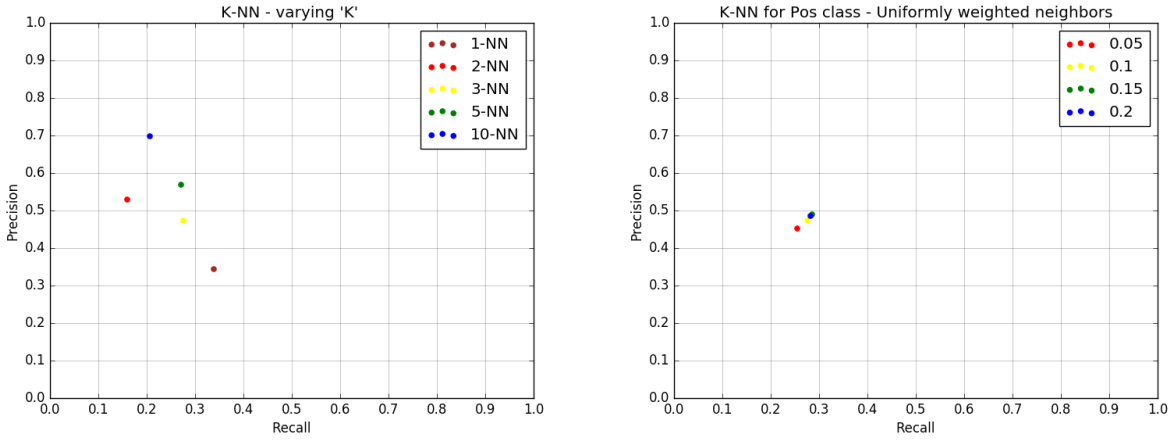

Figure 4: Precision-Recall curves for k-NN. In the first figure, we have plotted the precision-recall for the test set using k-NN with $k = \{1, 2, 3, 5, 10\}$. In the second plot, we have plotted the test set precision-recall curve for a k-NN classifier with different sample ratios.

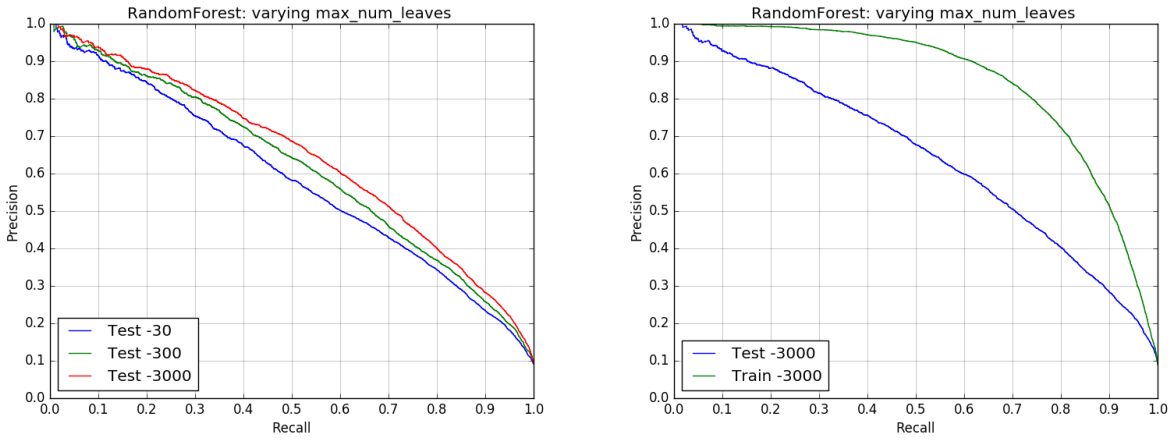

Figure 5: Precision-Recall curves for RF. In the first figure, we have plotted the precision-recall for the test set using RF with different number of max leaves. In the second plot, we have plotted the training and test set precision-recall curves for a RF with 3000 max leaves.

6.5 Random Forest

In the case of Random Forest we tried using different number of estimators and the results are shown below in Figure 5.

Similar to decision trees, we observed overefitting with higher number of max leaves. This behavior is what we expected.

Upon changing the number of estimators we observed increase in performance until a number of estimators, beyond which the P-R curve converges. This is shown in Figure 6.

6.6 AdaBoost

Of all the algorithms we tried, AdaBoost was the fastest (whereas SVM was the slowest) to come up with a prediction. The performance of AdaBoost is shown in Figure 7.

We examined the effect of changing the number of estimators and as we expected, increasing the number of estimators increased the performance up until a point, beyond which the performance seems to converge.

We also examined the effect of the learning rate of AdaBoost and we determined that the algorithm performed the best at a rate of 0.5.

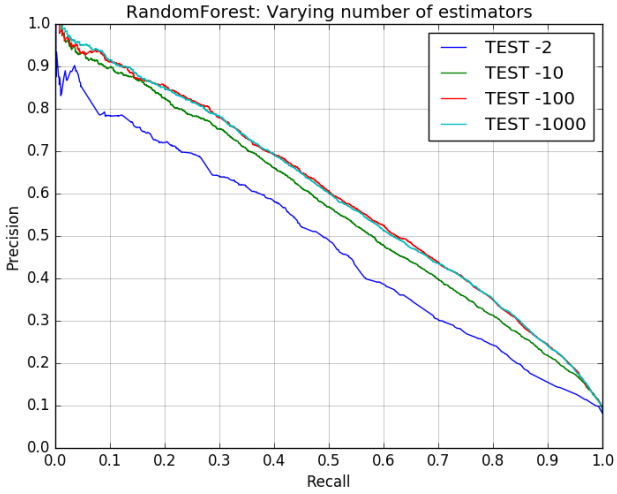

Figure 6: P-R Curve for Random Forest on the test set with different number of estimators.

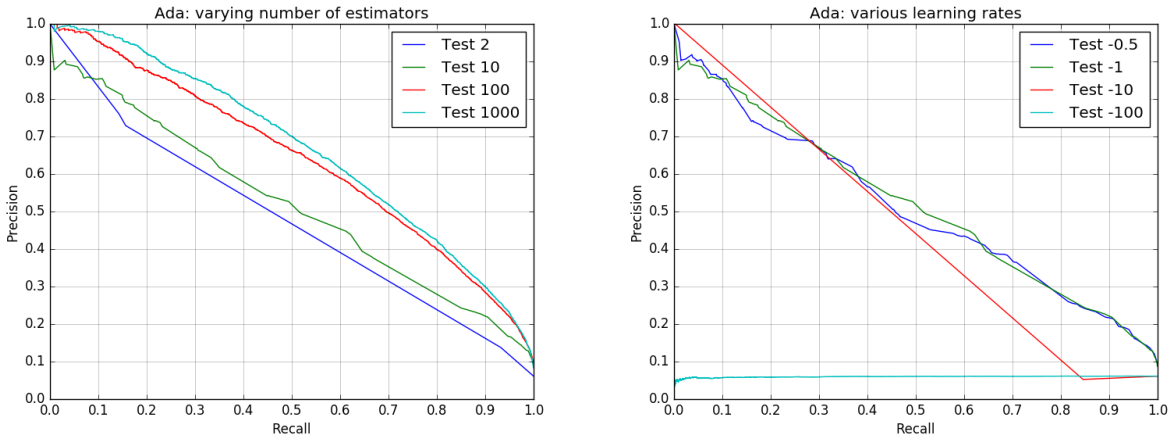

Figure 7: Precision-Recall curves for AdaBoost. In the first figure, we have plotted the precision-recall for the test set using Adaboost with different number of estimators. In the second plot, we have plotted the test set precision-recall curves for AdaBoost with different learning rates.

7 Conclusion

As mentioned earlier, our aim in this project was to get hands-on experience working on a machine learning problem and to understand the effects of different settings of parameters for the algorithms and to see the various learning algorithms in action on large volumes of data. We believe that this project has served this aim.

References

- Scikit-Learn Package Python - http://scikit-learn.org/stable/
- UCI repository Census Data - http://archive.ics.uci.edu/ml/datasets/Census-Income+(KDD)