PROJECT REPORT

(Adult Census Dataset)

CS273A – Machine Learning
Prof. Singh, Sameer
Fall 2018

Ramtin Afshar, Martha Osegueda, Daniel C. Ruiz

1 PROBLEM STATEMENT

We will be using the “Adult” data set and aiming to predict whether an individual’s income exceeds $50k per year based on U.S. census data from 1994 [1]. Since all three members of our group have relatively little background in Machine Learning, we chose the dataset that appeared most amenable to the techniques we utilized throughout the course. Furthermore, in our estimation, the binary nature of the dataset’s target value ensured a wide array of classifier options would be at our disposal.

To explore the data and implement our classifiers, we used several libraries to explore a variety of functionality. The four core libraries we used were: matplotlib for graphing; numpy and pandas for data exploration; and scikit-learn for parameter tuning and model fitting [2][3][4][5].

2 DATA EXPLORATION & FEATURE SELECTION

The data set consists of 14 features and the target value. 75.8% of data points correspond to a person earning less than $50k. Immediately, we realized that 9 of the features were categorical. Apart from this, several of these categories seem to encode redundant information. We merged several categories that had a smaller sample size and could be categorized under a larger super-category. Some examples include grouping smaller Latin American countries together, grouping types of relationships, and grouping marital status. While considering these merges, we avoided merging over splits that provided very high information gain to prevent joining highly dissimilar categories together. We then encoded the categorical features using the one-hot encoder from scikit-learn which uses the “one-of-k” approach discussed in class.

We also found redundant and unnecessary fields. We realized “education” (which is categorical) was also modeled numerically under “education-num”, thus this categorical field was dropped in favor of the numerical one. We also dropped “fnlwgt” since it was discussed on CampusWire that it held unrelated information.

Lastly, we normalized our numerical features (centering and scaling them) through the use of scikit-learn’s StandardScalar() transformer. We included all these steps in a data pre-processing pipeline. Though we considered further rounds of feature optimization and removal, we decided against this course of action since we wanted all features available to the bagging ensembles we planned to implement later on.

3 MODEL SELECTION & HYPERPARAMETER OPTIMIZATION

3.a Logistic Regression

The first classifier we introduced to our dataset was Logistic Regression. We chose this classifier due to its strength in dealing with Boolean values, due to our encoding of categorical features. By varying the several hyperparameters available to the classifier, we discovered the L2 regularization penalty was providing the highest scores on cross-validation of the training data. This fact limited the remaining hyperparameters we could optimize to two: regularization strength and the solver algorithm utilized by the classifier (see Figure 1).

Figure 1: Logistic Regression

Using the hyperparameters previously mentioned and those optimized above (0.1 error penalty and the lbfgs solver), our logistic regression classifier achieved a mean score of **0.8492** using stratified k-fold cross validation (k = 3) on training data.

3.b Linear SVM

Our next classifier to optimize was selected with more consideration for our data set. Since we were working with labeled data to predict a binary category, and our data set was not excessively large (i.e. in excess of 100k data points), we decided to fit a linear SVM. Like logistic regression, we varied two parameters using a grid search, and produced the following results:

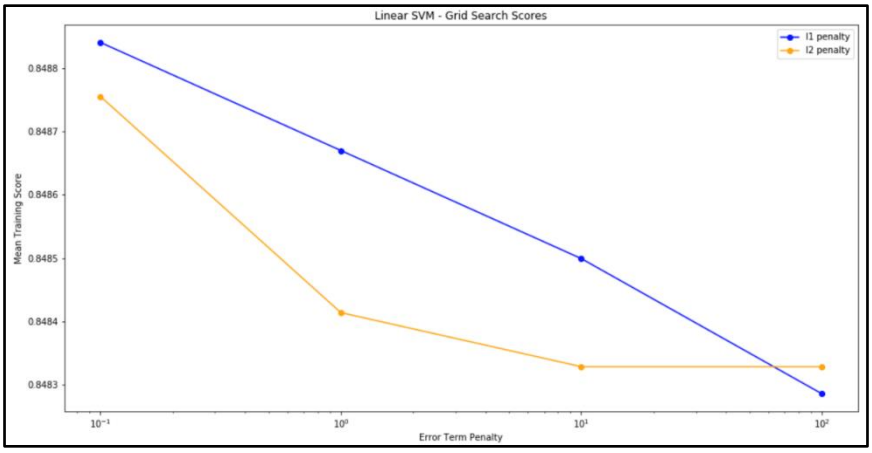

Figure 2: Linear SVM

Our Linear SVM Classifier, with hyperparameters optimized to 0.1 error penalty at L1 regularization, attained a mean score of **.8488** using stratified k-fold cross validation (k = 3) on training data. Surprisingly, these results were nearly identical to those of logistic regression.

3.c Gaussian Naïve Bayes

Next, we attempted to fit a gaussian naïve bayes classifier to our training data. Considering our data was purely numeric (after pre-processing), and not text-based as is normally preferred with Gaussian Bayes, we were not expecting stellar results. For this classifier, the only parameter we could vary was the smoothing, or the portion of the largest variance of all features that is added to variances for calculation stability (Figure 3). Notably, in order to enable this classifier to work, we converted our training data matrix into a dense matrix, since the sparsity of the original version was incompatible with this classifier.

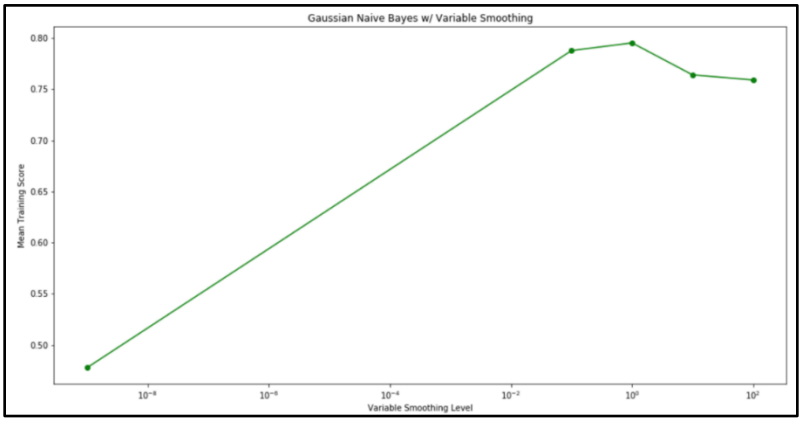

Figure 3: Gaussian Naïve Bayes

As expected, the naïve bayes classifier performance was underwhelming in comparison with our other trials so far. The highest mean score was recorded at **0.7953** with the variable smoothing parameter set to 1.0. Like the previous trials, this was achieved using stratified 3-fold cross validation.

3.d K-Nearest Neighbors

We also tested K-nearest neighbors, hoping to gain an increase in prediction power due to the numeric nature of our data. Like logistic regression and linear SVM, we optimized two hyperparameters (see Figure 4). Unfortunately, our KNN classifier was unable to outperform logistic regression. Its mean score of **0.8462**, was recorded at 25-nearest neighbors and using uniform weights.

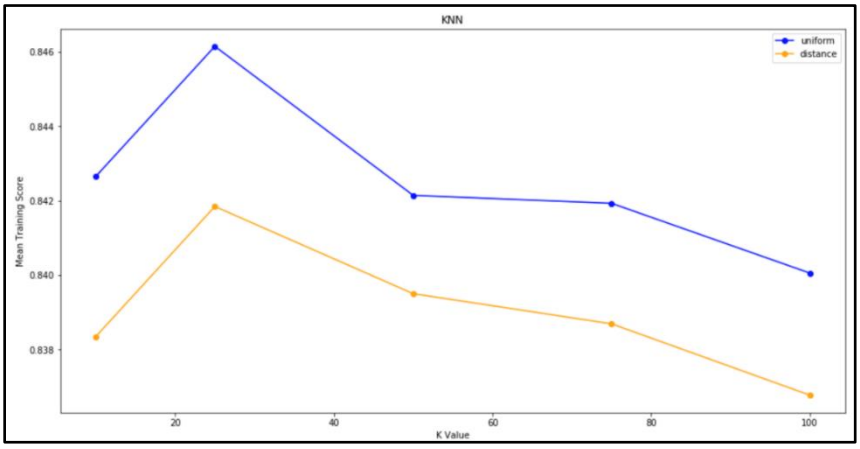

Figure 4: K Nearest Neighbors

3.e Decision Trees

We decided to test decision trees as our last simple classifier, with the realization that there existed many different ways to partition the data. We tried ranges of minParent and minLeaf values (Figure 5), and we obtained the best mean cross-validation score of **0.857.** This was tested using fractional values, the best parameters of which were 0.00681 and 0.001 (which account for around 60 and 20 data samples).

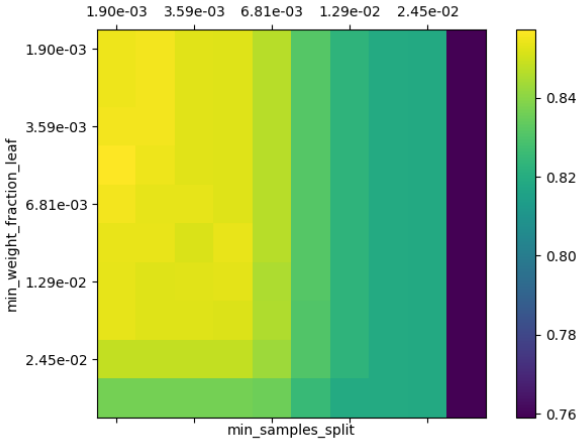

Figure 5: Decision Trees

3.f. Overview

Overall, the decision trees provided the best results for a simple classifier (Table 1), and motivated us to look into ensemble methods built on-top of decision trees.

Classifier	Mean CV Score
Logistic Regression	0.8492
Linear SVM	0.8488
Gaussian Naïve Bayes	0.7953
K-Nearest Neighbors	0.8462
Decision Tree	0.8573

Table 1: Single Classifier Results

4 ENSEMBLE METHODS

4.a Voting & Bagging Classification

We next turned our attention to more complex classifiers with the hopes of improving upon our results above. Firstly, we sought to create a classifier that combined two ensemble methods discussed in class: voting and bagging. We began by selecting the three of the highest performing single classifiers above (logistic regression, linear SVM, and KNN), and replaced the classifiers within their pipelines with bagged versions. These bagging classifiers used 10 estimators each, and sampled 50% of both the data points and features (both with replacement, via bootstrapping). These three new pipelines were then rolled into a hard (majority) voting classifier, and used to classify our test data with 10-fold stratified cross validation. The results were promising, though not quite as high as expected, with **0.8321** the highest score achieved.

4.b Random Forests Classification

Due to the great performance of decision trees in the previous section, we decided to try other ensemble methods built over them. One of which was random forests. The amount of hyperparameters we could optimize were very large. The parameters we optimized were the following:

```
RndF_param_grid = {'classifier__n_estimators': [5, 10, 25, 40, 50],
    'classifier__criterion': ['gini', 'entropy'],
    'classifier__max_depth': [5, 10, 15, 20, 25],
    'classifier__min_samples_split': [2, 5, 10, 15, 20],
    'classifier__min_samples_leaf': [2, 5, 10, 15, 20] ,
    'classifier__max_features': ['auto', 'log2', None]
}
```

Using small subsamples of our training data, we found an optimal combination of parameters that increased our mean predictive scores by roughly .005 over any prior classifiers. We trained this model on all the training data and predicted the test data to get a test error of **0.8652.**

The parameters we selected are listed below:

- 40 Estimators
- Entropy as Criteria
- Max Depth 20
- MinParent 15
- MinLeaf 2
- Max Features: “auto”

4.c Gradient Boosting Classification

We also decided to try Gradient Boosting Trees due to Decision Trees’ performance. Gradient Boosting seemed like a good addition since it seemed like it would help ensure all of the exceptions to early versions of the tree are accurately predicted. Originally, we decided to find the optimal loss function (‘exponential’ or ‘deviance’), maximum features to consider at each split and MinParent.

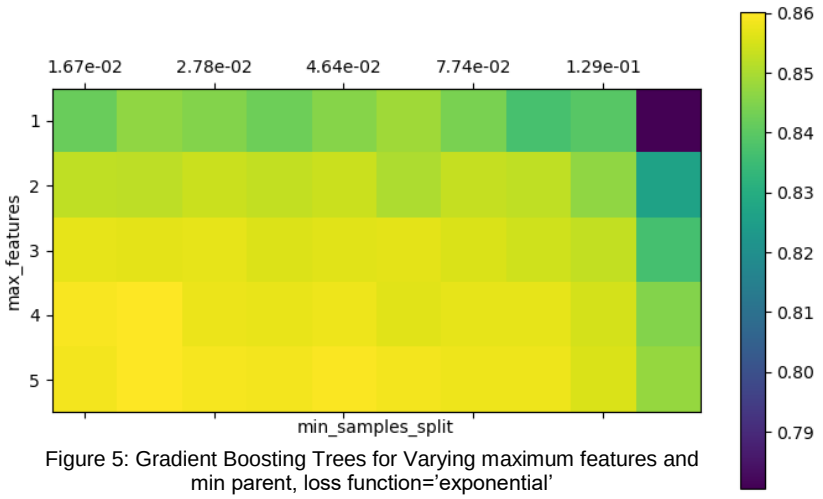

Figure 5: Gradient Boosting Trees for Varying maximum features and min parent, loss function=’exponential’

We found that the best parameters were using exponential loss, 5 features, and a fraction of 0.0167 (around 160) of the number of elements as MinParent. This achieved a cross-validation error of 0.860. We also realized that 5 features were on the upper bound of features to consider, so we unbound this parameter while calculating the test error.

We trained this model on the training data then predicted the test data, which gave us a test error of **0.8663,** barely out-performing random forests.

5 CONCLUSION & MODEL RECOMMENDATION

The dataset lends itself to performing nicely under decision tree-based classifier. This was evident by the performance of decision trees when we compared simple classifiers, but only became more evident when we performed Gradient Boosting and Random Forest Classification.

Table 2: Ensemble Classifiers

Ensemble	Testing Error
Voting and Bagging	0.8321
Random Forest	0.8652
Gradient Boosting	0.8663

The parameters we optimized for our Gradient Boosting Tree classifier achieved a **0.866** test error, and serves as our official recommendation for this dataset. We feel these parameters represent a decent balance between bias and variance, as the trees were never allowed to get to deep or overfitted due to the 160 min split/parent.

Given more time, we would have enjoyed applying further ensemble techniques to the dataset, to build a more tailored ensemble for the data. Similarly, we would like to improve the tuning of the bagging parameters on the voting.

6 DISTRIBUTION OF WORK

Each member of the team contributed significantly to the overall direction of the project and decisions that were made throughout. As a whole, the team put ideas together by discussing topics that came up in class and applying them to the project.

Ramtin Afshar contributed to the development of classifiers throughout the project and was overall responsible for the implementation of the random forest classifier.

Martha Osegueda was responsible for data exploration and feature selection. She was responsible for implementing decision trees, gradient boosting trees and evaluating testing error of finalized parameters. She also coordinated set up of the group repository and finalized the report draft.

Daniel Ruiz was responsible for developing the pipelines for preprocessing and building the project's notebook where we optimized hyper-parameters. He also coded classifiers for logistic regression, linear SVM, Gaussian naïve Bayes, KNN and bagging along with their associated plots. He also drafted the project report.

7 REFERENCES

1. https://archive.ics.uci.edu/ml/datasets/adult - Dataset Source and Documentation
2. https://matplotlib.org/contents.html - Matplotlib Documentation
3. https://docs.scipy.org/doc/ – Numpy Documentation
4. https://pandas.pydata.org/pandas-docs/stable/ – Pandas Documentation
5. https://www.scikit-learn.org – Scikit Learn Documentation