CS 4372

Machine Learning Using Trees

Ryan Sharp (RXS180092), Timothy Cox (TMC190004)

Project and Dataset Selection - Preprocessing

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   work-class      32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education num   32561 non-null  int64
 5   marital status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital-gain    32561 non-null  int64
 11  capital-loss    32561 non-null  int64
 12  hours-per-week  32561 non-null  int64
 13  native-country  32561 non-null  object
 14  class           32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Our dataset is called the “Adult” dataset. The objective of this data set is to predict whether an adult makes more than $50,000 per year based on the given predictors. It has 14 predictor features and one response variable. We can see that 8 of the predictors are categorical variables and 6 are continuous. The response variable “class” is a categorical variable with two classes, “1” for >50k and “0” for <=50k.

To make it easier to identify correlation between predictors and response, we converted all our categorical predictors into indicator variables.

After checking for correlation between predictors and response, we decided to keep only predictors with at least +-0.2 correlation. This includes if any indicator showed significant correlation, we must keep all the other indicators that belong to that same parent predictor. The correlation matrix is 34x34 and is included in the google collab, but due to its size will not be included in this report.

After eliminating low correlation predictors, we are left with the **reduced dataset**:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   education num   32561 non-null  int64
 2   marital status  32561 non-null  object
 3   occupation      32561 non-null  object
 4   relationship    32561 non-null  object
 5   sex             32561 non-null  object
 6   capital-gain    32561 non-null  int64
 7   hours-per-week  32561 non-null  int64
 8   class           32561 non-null  object
dtypes: int64(4), object(5)
memory usage: 2.2+ MB
```

Model Construction

We will construct 4 classifiers: Decision Tree, Random Forest, AdaBoost and XGBoost. First, we will set aside a randomly selected 20% of our data as a final test set. The other 80% of our data will be for training and cross-validation. We will use the cross-validation to fine tune our hyper-parameters. When we establish the best set of hyper-parameters for each model, we will then test our models with the test data and analyze the results.

Decision Trees and especially ensemble methods are not sensitive to variance in data therefore we do not need to scale the data. Using min-max scalar on the DT provided the same results as without the scaling.

Hyper-parameter Tuning – For each model, we created a GridSearchCV (With CV=5). To determine which hyper-parameters were "best", we tried to include every option for that hyper-parameter. If the hyper-parameter was continuous, we tried to give it the option to go up or down and made sure it selected a value in the middle.

Decision Tree Parameter Tuning	
HYPER-PARAMETERS	**VALUES**
Max Depth	[3, 8, 15, 30]
Criterion	['gini' , 'entropy']
Max Features	['auto', 'sqrt', 'log2']
Min Sample Leaf	[1, 2, 3, 5, 8]

Example: These are the parameter choices for our DT model. If the best "Max_depth" came back as 3 or 30, we would re-run the parameters with some more options smaller than 3 or larger options than 30 to make sure that the best parameter was selected, not just one of the upper/lower bounds.

Decision Tree best parameters

```
{'dt__criterion': 'entropy', 'dt__max_depth': 15, 'dt__max_features': 'sqrt', 'dt__min_samples_leaf': 3}
```

Random Forest best parameters

```
{'max_depth': 12, 'max_features': 'log2', 'n_estimators': 100}
```

AdaBoost best parameters

```
{'algorithm': 'SAMME.R', 'learning_rate': 1, 'n_estimators': 500}
```

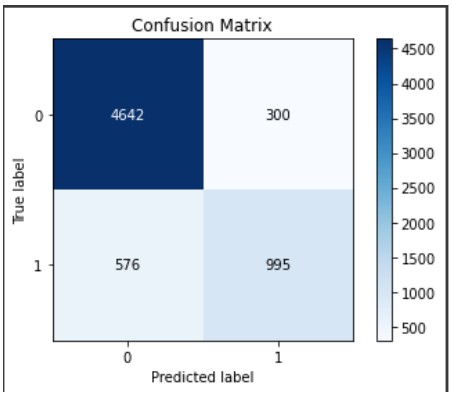
XGBoost best parameters

```
{'colsample_bytree': 0.6, 'eta': 1e-05, 'gamma': 0, 'max_depth': 9, 'min_child_weight': 6, 'subsample:': 0.6}
```

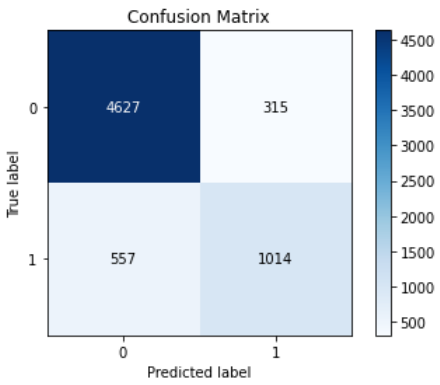
Results Analysis

Test metrics – Using the best parameters selected from our parameter tuning we tested each model with our test data and calculated these metrics.

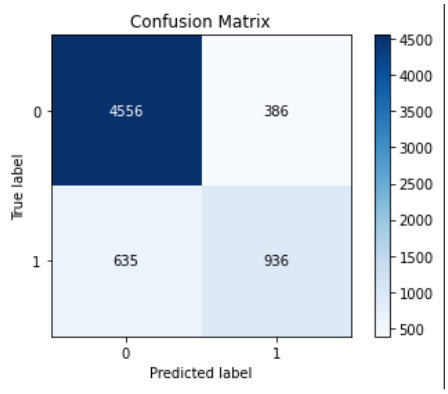
AdaBoost CM



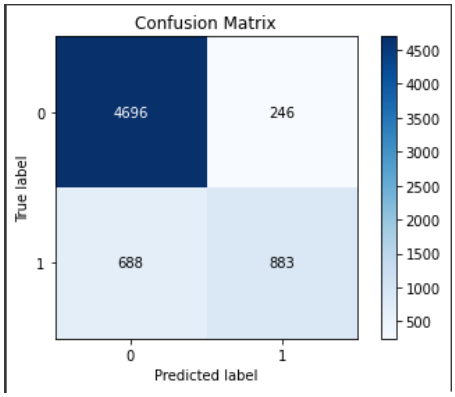
XGBoost CM



Decision Tree CM



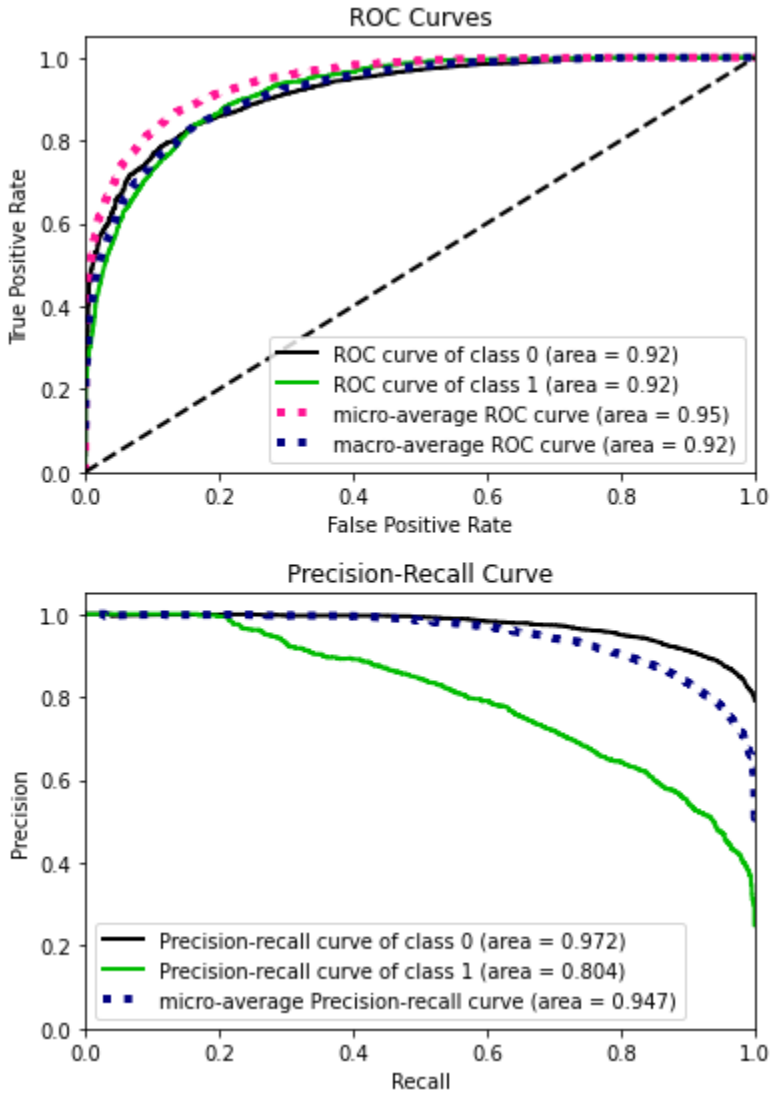
Random Forrest CM



MODEL	ACCURACY	PRECISION	RECALL	F-STAT
Decision Tree	**84%**	**70%**	**59.5%**	**64.3%**
Rand Forest	**86%**	**78.2%**	**56.2%**	**67%**
AdaBoost	**87%**	**76.8%**	**63%**	**69.9%**
XGBoost	**87%**	**76.2%**	**64.5%**	**70.3%**

AdaBoost and XGBoost both perform well on the test data. XGB has a slight edge in recall and AdaBoost is slightly better in terms of precision. Because the boosted methods are performing better, we will look at their ROC, precision-recall curve and a breakdown of their confusion matrices.

AdaBoost ROC/Precision Recall graphs



Remember that class 0 is less than $50,000 annual salary and 1 is greater than $50,000 annual salary.

The Adaboost does an amazing job at predicting class 0 correctly. This is consitent throughout all models. Predicting class 1 seems to be the more challenging objective.

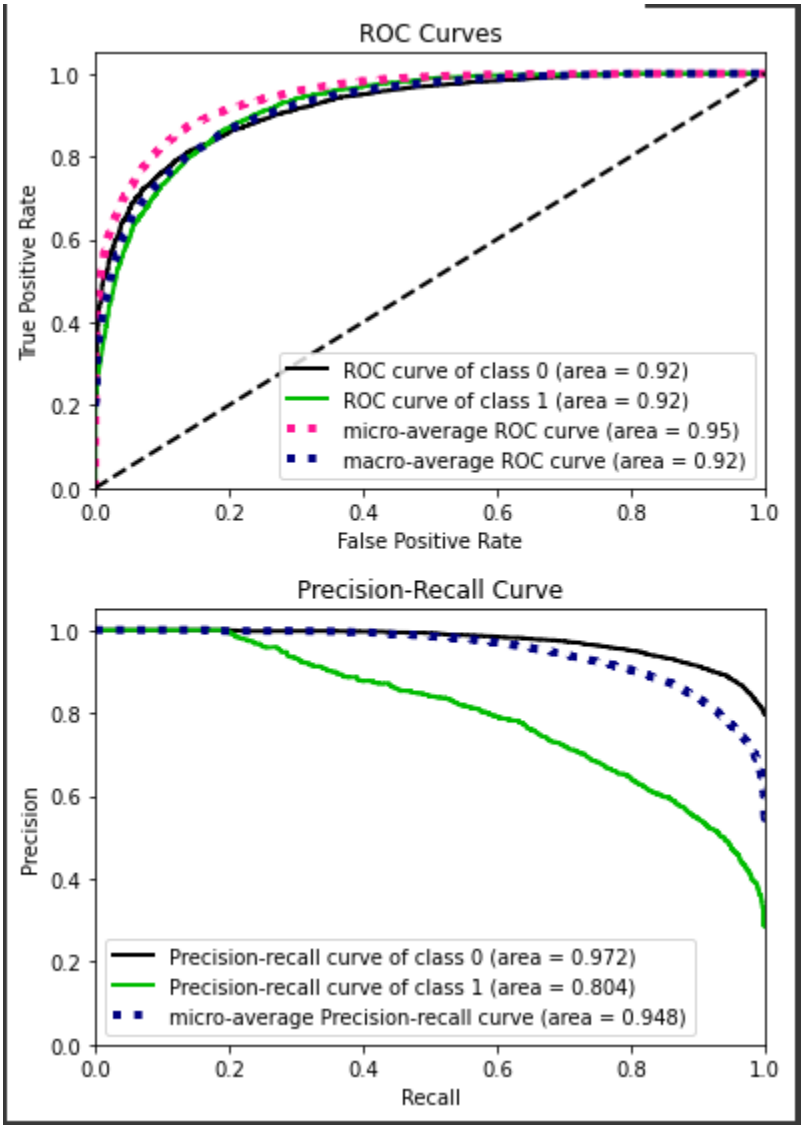
ROC

ROC is very useful in our use case. As stated above, the more challenging problem is correctly identifying class 1. ROC compares TPR vs FPR. Neither of these metrics depends on True negatives. The area under the ROC curve for both classes is 0.92.

Precision-recall curve

Looking at the area under the P-R curves we can see our model is much better at identifying class 0 correctly than it is at identifying class 1.

XGBoost ROC/Precision-Recall graphs



The ROC curves and P-R curve for XGB is almost exactly the same as AdaBoost.

Conclusion

We can see by the confusion matrices and our models' performance metrics that all four models predict class 0 very well. After looking into the results, we identified that predicting class 1 was a more difficult task. With that in mind, we could see that XGBoost and AdaBoost were able to correctly predict class 1 at a higher rate than our decision tree and our random forest. Both AdaBoost and XGBoost as such similar performances that either or could be used on this type of data effectively.

Improvements – Given more time / computing power, we could increase our grid search to further identify the best parameters to potentially improve our models.

Decision Tree Visualization – This Tree's max depth is 3. Our actual best tree depth is 15. For visualization purposes we reduced the depth.