**Project report**

**Konshina Anastasia**

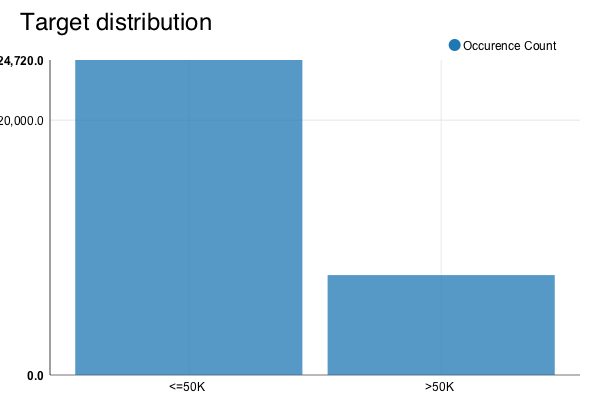
**Savickiy Eugeney**

**Problem statement**

In this project we are going to solve a classification problem of adults income dataset (LINK) using a number of social-demographic features and a number of other related features. That is whether an adult has less than 50k dollars income or greater or equal than 50k dollars income. Obviously, the target variable has unbalanced levels of income since there are always less people with small income compare to people with high income. In the next section we are going to check this assumption and show that this is a case. Also, our primary outcome needs clarification. We are more interested in correctly predicting the probability of a high outcome, therefore the primary outcome is target = '>50k' and the secondary outcome is target = '<=50k'.

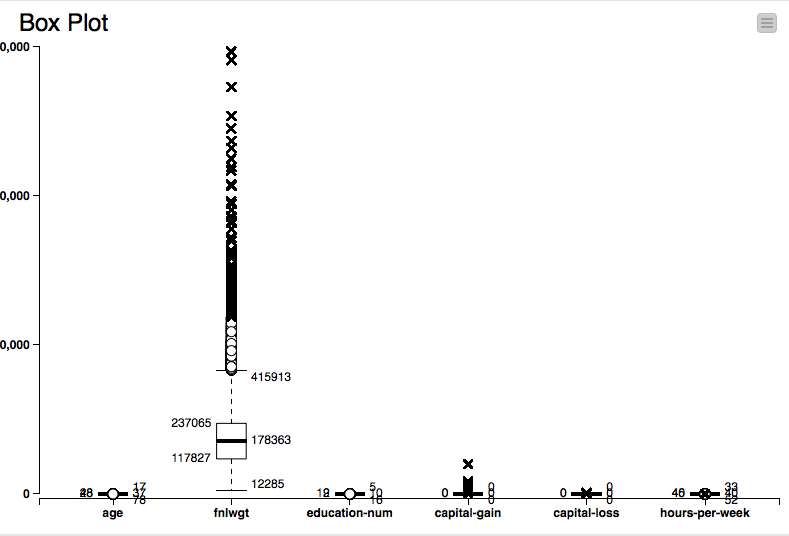
**Dataset summary with basic statistics and respective plots**

Let us begin with a graphical summary of a given dataset. We start with checking the above assumption about the levels of a target variable.

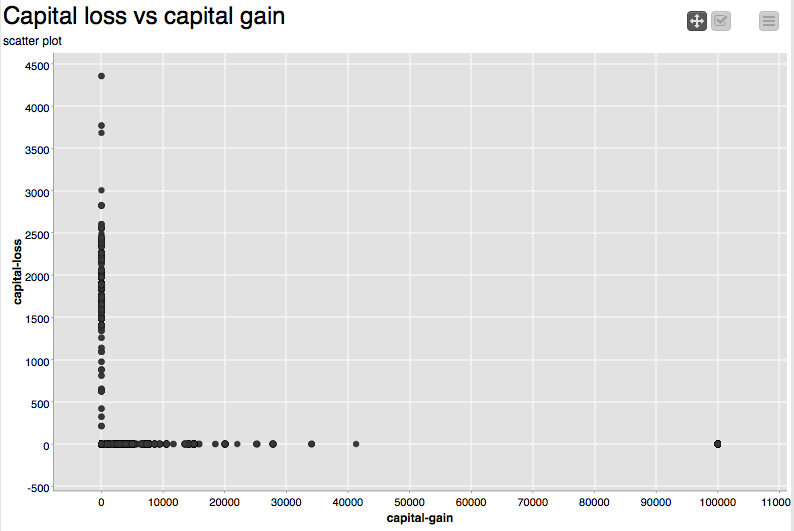


It can be seen from the graph that target distribution is indeed unbalanced. That can cause potential problems to the sensitivity of a classifier to a primary outcome (next referred to as positive class) on account of lack of information about rare class examples. Also, this is a reason to not to use classification accuracy as a quality metric.

Next lets check interval predictors. Here is a box plot of all interval variables.

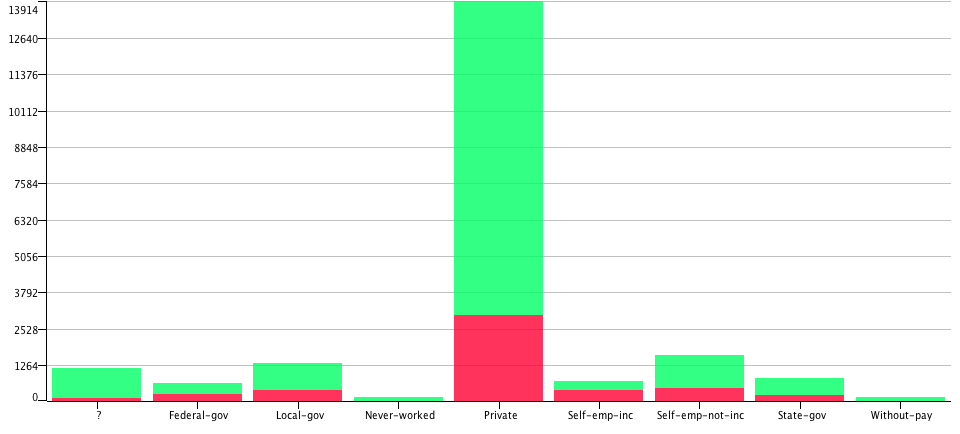


Apparently, there are a lot of outliers in fnlwgt and capital-gain features. We need to take it into account since, for instance, linear models are very sensitive to extreme observations and highly skewed distributions which is a case for fnlwgt. Apart from that we’ve discovered that there is no apparent correlation between interval variables. For example, here is a scatter plot for capital features. We are safe to assume that multicollinearity is not a case for given interval features.

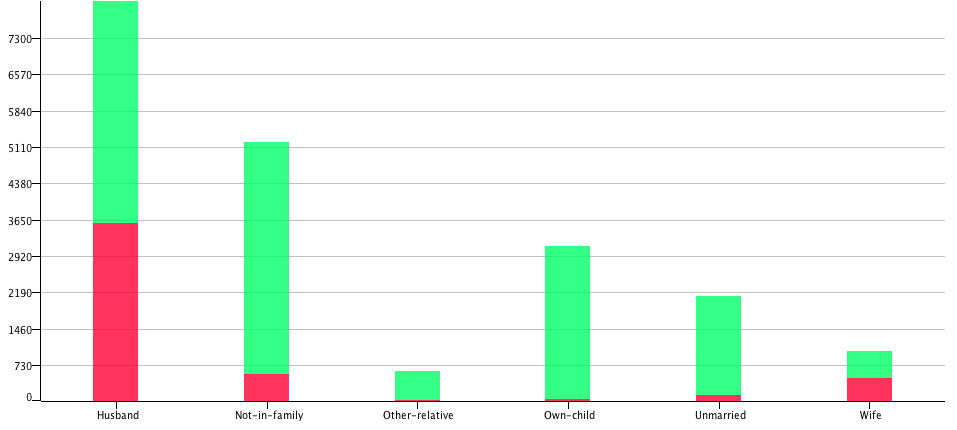


Now lets take a look on some of the interesting and potentially useful nominal attributes.

Here is a stacked bar chart for workclass variable. Apparently most of the people didn’t reveal their workclass, so there is basically no information in this one except that people who never worked all have smaller income. Note: reds are positives, greens are negatives.

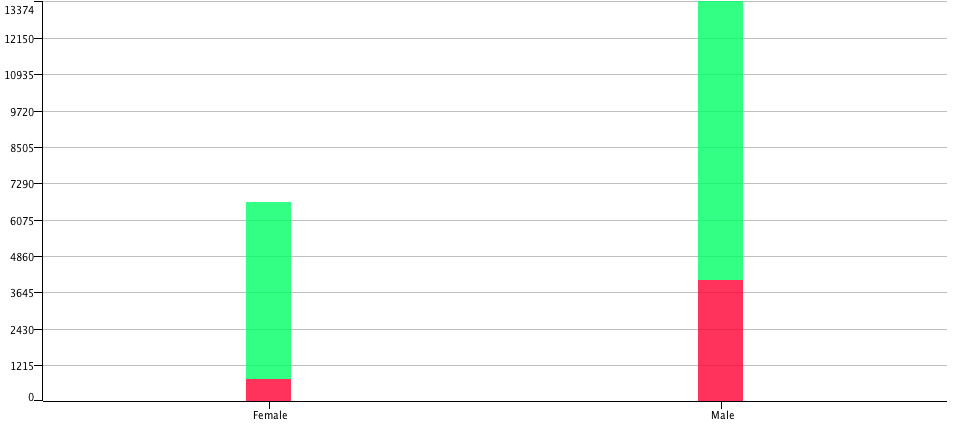


And here is a stack bar chart for relationship variable.



There is a pattern in this one in that people with high income are mostly either husbands or wives.

And the last one of our interest.



As we would expect the larger fraction of males has high outcome compare to the fraction of females.

**Methodology**

We’ve seen some of the crucial properties of this dataset above. There are outliers, there are redundant variables, there are missing values, there are extremely skewed interval distributions and there is an imbalance in target levels. So first of all we would want to define a list of models we are going to try and compare as well as software we are going to use. As for the software: KNIME analytics platform. It is open source, it is very powerful and flexible. It has nodes that allow to integrate R code, python code and java code into a workflow. Also it includes most of the predictive models from another open source project called Weka. So basically it has unlimited capabilities in a way. As for the models, see below.

1. A simple pruned decision tree learner, because it allows to look at some patterns in the data. It is robust to outliers, it has a way to handle missing values even without imputation. It is a nonlinear classifier in case our data turns out to be non-separable in linear sense. It has it’s own feature selection mechanism.
2. Random forest learner. The reasons are the same as above, but also this ensemble allows to combine weak learners such as unpruned trees into a powerful model. Also forests are very good when it comes to unbalanced data.
3. Gradient boosting based on trees. The benefits of using this model are basically the same.
4. Multilayer perceptron. Multilayer perceptron can approximate virtually any function, but it lacks missing values handling and feature selection and is sensitive to outliers. It may perform well on this dataset, because there may be complicated relationship between the target and the attributes.
5. Logistic regression with L1 regularization. We also want to try simple logistic regressions with regularization because we have enough rows in this dataset even though the dimensionality is huge on account of nominal variables after converting into dummies.
6. Logistic regression with L2 regularization.

That’s about it for models, we’ve covered powerful nonlinear models and lesser linear models.

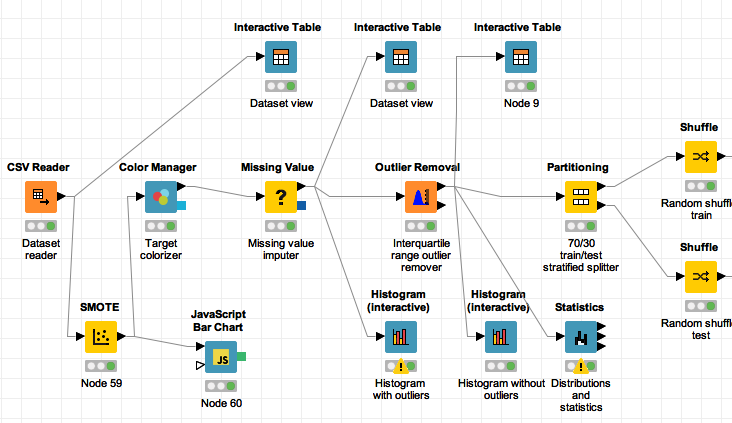
First of all, we want to try these models on a raw dataset, meaning we want to simply impute missing values with medians for interval attributes (median because it is robust to skewed non-symmetric distributions) and with the most frequent values for nominal attributes because it seems reasonable to impute in such a way. Then we want to remove outliers based on interquartile range, meaning if an observation lies outside of Q1 – 1.5\*(Q3-Q1) or Q3 + 1.5\*(Q3-Q1) that it is considered an outlier. Here Q means quartile.

Further we don’t need to do any variable selection by ourselves (as well as scaling), because chosen algorithms do it by themselves, except multilayer perceptron. Trees based algorithms are not sensitive to distributions or scaling properties. Next we partition the whole dataset into train and test subsets with 70/30 proportion in a stratified way so that we keep target proportions in both subsets to not to skew any statistics such as % cumulative response which is our main measure of quality alongside with ROC index. Then we shuffle both subsets, train our models, tweak hyperparameters for better performance and compare them.

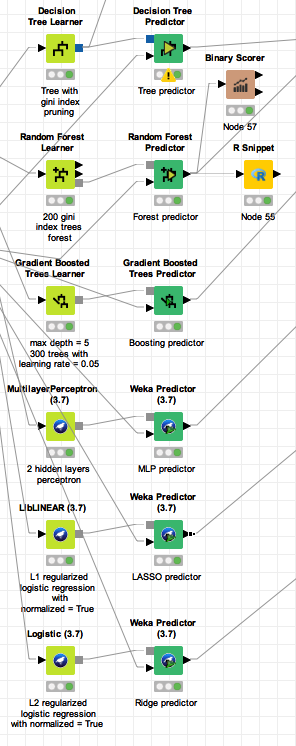
After that we repeat this procedure for the oversampled dataset to try to achieve better performance on a dataset that holds more information about the rare target level. We use SMOTE oversampling methodology to get 50/50 proportion of both target levels and even the chances of both classes to be correctly classified.

**Experiment setup**

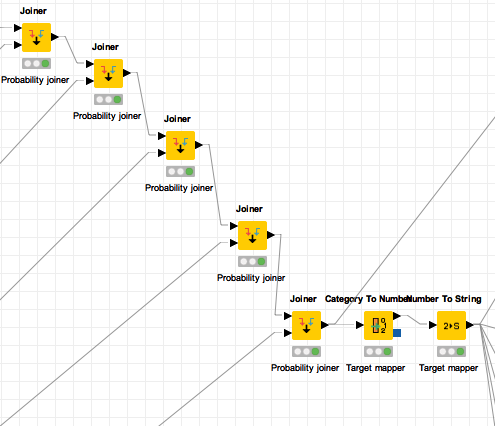
Next we introduce the whole experiment final pipeline setup in KNIME.



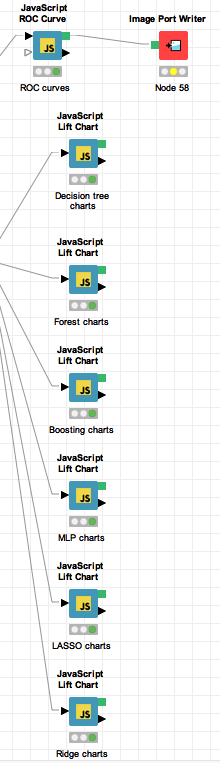
In this part we take care of oversampling within SMOTE node, missing values imputation, outlier removing and partitioning. The next part on the following page concerns the learning and predicting tasks.



The next part joins all obtained probabilities in a single dataset and recodes levels to appropriate format so we could compare them via ROC index and captured response charts.



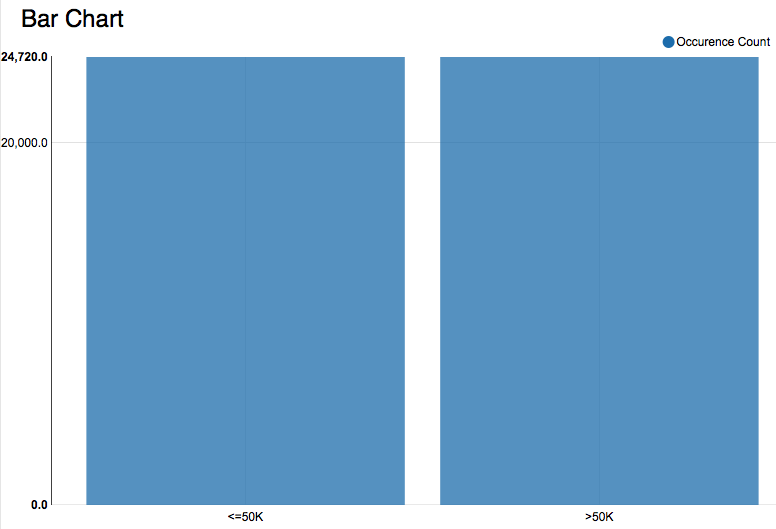
And the next final part takes care of graphical reports.



**Results and discussion**

After the first run (which is not interesting in details, really) on the unbalanced dataset we’ve discovered that the best performing algorithm is gradient boosting based on decision trees. We’ve obtained about 0.83 ROC index and cumulative captured response about 70% positive cases within the first 40% of rows, sorted by the probability of a target being positive. It is not bad, but as it turned out the no information rate for this unbalanced dataset is approximately 0.81 and classification accuracy is about 0.87 (note that this means that there are 19% of positive outcomes and we beat only by 6%, which proves that accuracy is a bad measure in this case). Even though we beat guessing algorithm, we still were not far from it, because the contingency table showed that within 2400 positive test cases we classified correctly only 1250 positive cases and also misclassified about 2000 negative cases into positive (false positive).

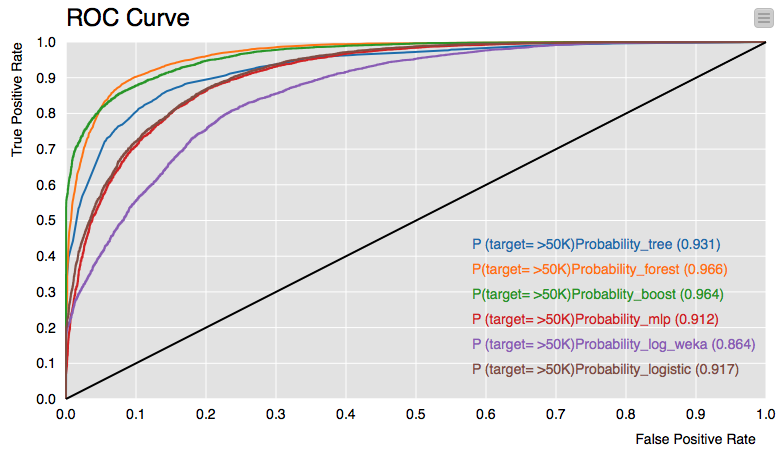
Now lets take a closer look at the results obtained after oversampling. But before that lets also take a look at target distribution after oversampling.



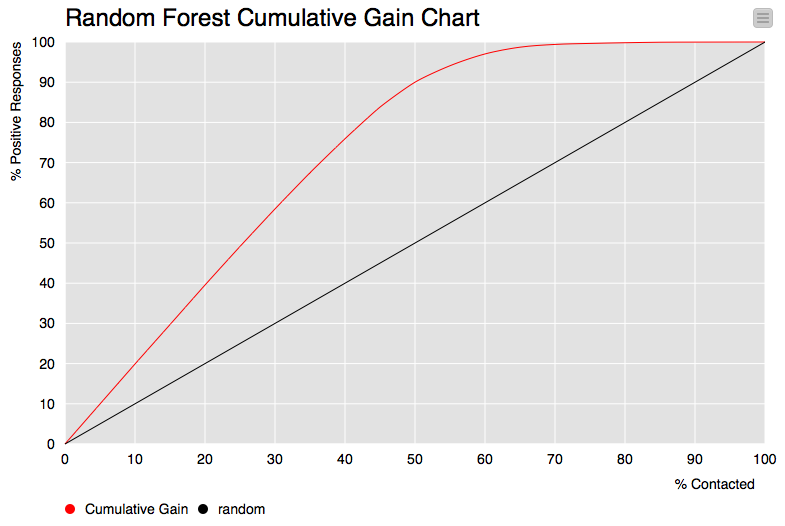
Now it looks balanced.

We tuned models hyperparameters until we reached peaks on the ROC indexes on a plot below. Later we will reveal parameters in more details.

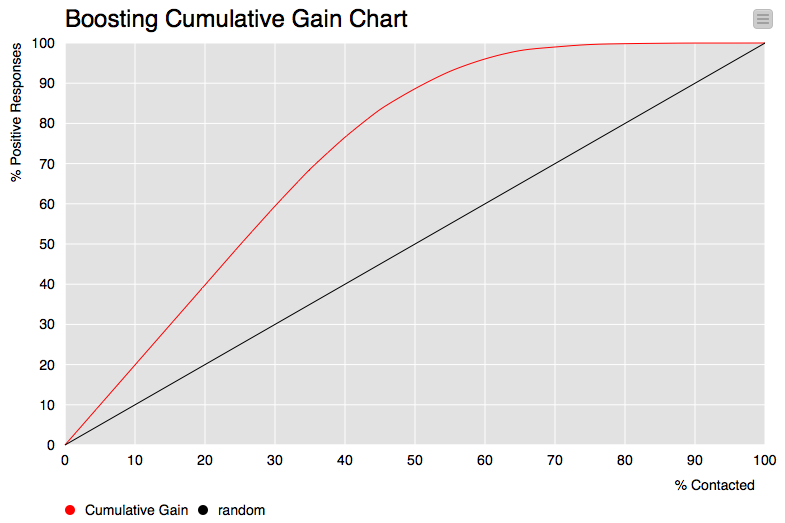
There goes the ROC curve chart and corresponding indexes.



This time around it can be seen that there is a competiton between random forest and gradient boosting models. It seems that boosting performs better on upper fractions of predicted probabilities and then falls down while random forest performs slightly worse on upper fractions but then beats boosting model in terms of TPR to FPR trade-off. Also there is a negligible difference between ROC indexes of those two models. We need to examine it in more details. Lets now take a look at cumulative gain chart which will give as better picture of performance.



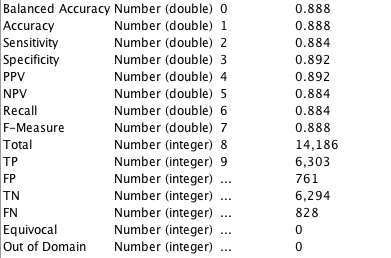
Remember that now we have 50% of positives and 50% of negatives. This chart shows that, for example, on a 50% fraction of sorted probabilities (% Contacted) we capture 90% of all positive cases (our interest) in the test data. Now this means that within this 50% fracture we captured 90% of all positive cases and only 10% negatives we might classify into false positives if we are to choose such a threshold for classification. On the other hand, around the fracture of 70% we capture 100% of positive cases. This looks very good and accurate. Lets compare this chart to the chart for the gradient boosting model.



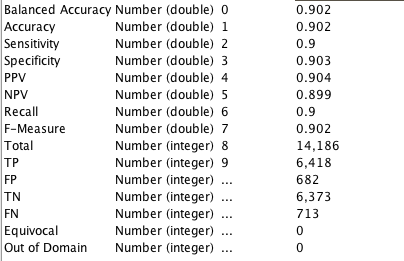
This chart also looks very good and similar, but one can notice that this chart does not show that this algorithm captures 100% about 70% fracture and captures less than 90% on a 50% fracture. Boosting performs slightly worse than random forest in terms of positive case capture.

Lets also take a look at classification reports of all models and see how well they perform at 0.5 probability threshold.

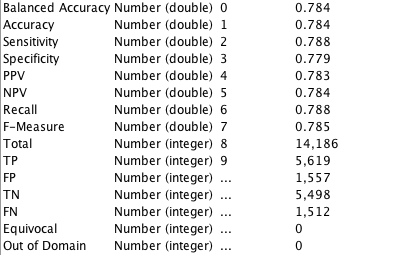
**Boosting score**



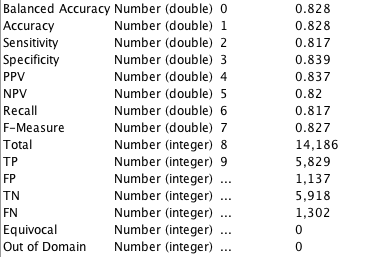
**Forest score**



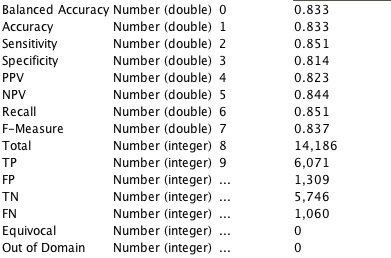
**Lasso score**



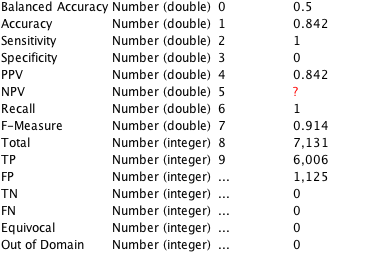
**Multilayer perceptron score**



**Ridge score**

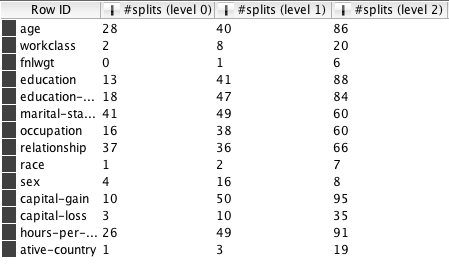


**Tree score**



As we can see, random forest shows the best performance in terms of captured response, true positives vs false positives and ROC index. Boosting does a little worse, whereas ridge logistic regression has problems with identifying positive class and tree just labels everything as positive. Also we can take accuracy into account this time, because this dataset is balanced and accuracy shows real numbers, not just some promising high value on extremely unbalanced data.

Now that we are certain that random forest shows the best performance, lets take a look at split statistics.



Those are split statistics on the first three levels of every tree. It shows how many times the model split a node on a value of each attribute. We can see that some attributes are almost uninformative, such as race, native country or fnlwgt. And some were used for splitting very frequently. It makes sense, for example, that the outcome defines by age as usually the older you are, the more experience you have and the more money you can make out of your experience. Or the more hours you work, the more money you can make. On the contrary, it doesn’t matter where you were born or what skin color you are.

At last a few words about hyperparameter tuning. We tested decision tree with different settings, including the split criterion, max depth, min number of features per node and nominal feature split strategies. Multilayer perceptron with different number of hidden layers from 1 to 4, different learning rates and a normalization option turned on. Logistic regressions with different costs and normalization turned on as well. And finally a pack of different forests with information gain, information gain ratio and gini split criterions, number of trees from 10 to 200 with 20 as increment step and bagging option set to square root. The same goes for boosted trees. Final setting for every model you can see at screenshots of project workflow in the experiment setup section.

**Conclusion**

We’ve built a number of different models with initial prior probabilities dataset and oversampled dataset and found out that models show worse performance when there is lack of information about rare class. We’ve done a full pipeline of data preparation and statistics assessment. We’ve tried different feature selection approaches for ML perceptron and logistic regression since those two lack this capability and it turned out that those models perform better on the full set of features. And finally with random forest statistics we’ve discovered that there are some interesting and reasonable patterns in the data that one would expect income to depend on.