INCOME PREDICTION - KAGGLE PROJECT

Han Ambrose
CS6350 Spring 2020 - Machine Learning
University of Utah

April 26, 2020

ABSTRACT

Machine learning is a good way to classify data and make prediction. It is also widely used in different field nowadays. In this project, we implement different machine learning methods to predict whether or not an individual makes more than $50,000/year.

1 Problem definition and motivation

This project is a competition on Kaggle (Income Level Prediction). The dataset was extracted by by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the specific conditions. Prediction task is to determine whether a person makes over $50,000/year a year given the census information. There are 14 attributes, including continuous, categorical and integer types. Some attributes may have missing values, recorded as question marks. By using different machine learning methods, we are able to compare robustness among different models and also to determine the best performing model.

2 Solution

Task modeling. The project was started using ensemble, linear classification approach and then moved to non-linear classification approach.

Construct train and test data. There are two set of data provided: the training set and the test set. We use the 'test_final.csv' only for computing prediction and submitting for score. The only data set that was used to build models is 'train_final.csv'. Thus, the training set was then split into 2 parts: train and test. Further, we also use cross validation technique to split it to k-folds (train, test, validation) to evaluate the performance of a model on unseen data.

Before jumping into running our model, we need to preprocess the data first. Any missing data or skewed dataset could have a big impact on prediction results. Data cleaning is an important step, just as important as modeling.

2.1 Data Cleaning

2.1.1 Missing Values

As we know that ['?'] indicates missing values so by running a quick check to identify the locations for these columns and counts of missing values for that column. Below is the results of missing data from both dataset provided.

Attributesl Counts	Number of missing values	Percentage of Missing Value
Workclass	1,437	5.75%
Occupation	1,442	5.77%
Native.country	427	1.71%

Table 1: Training data's missing values

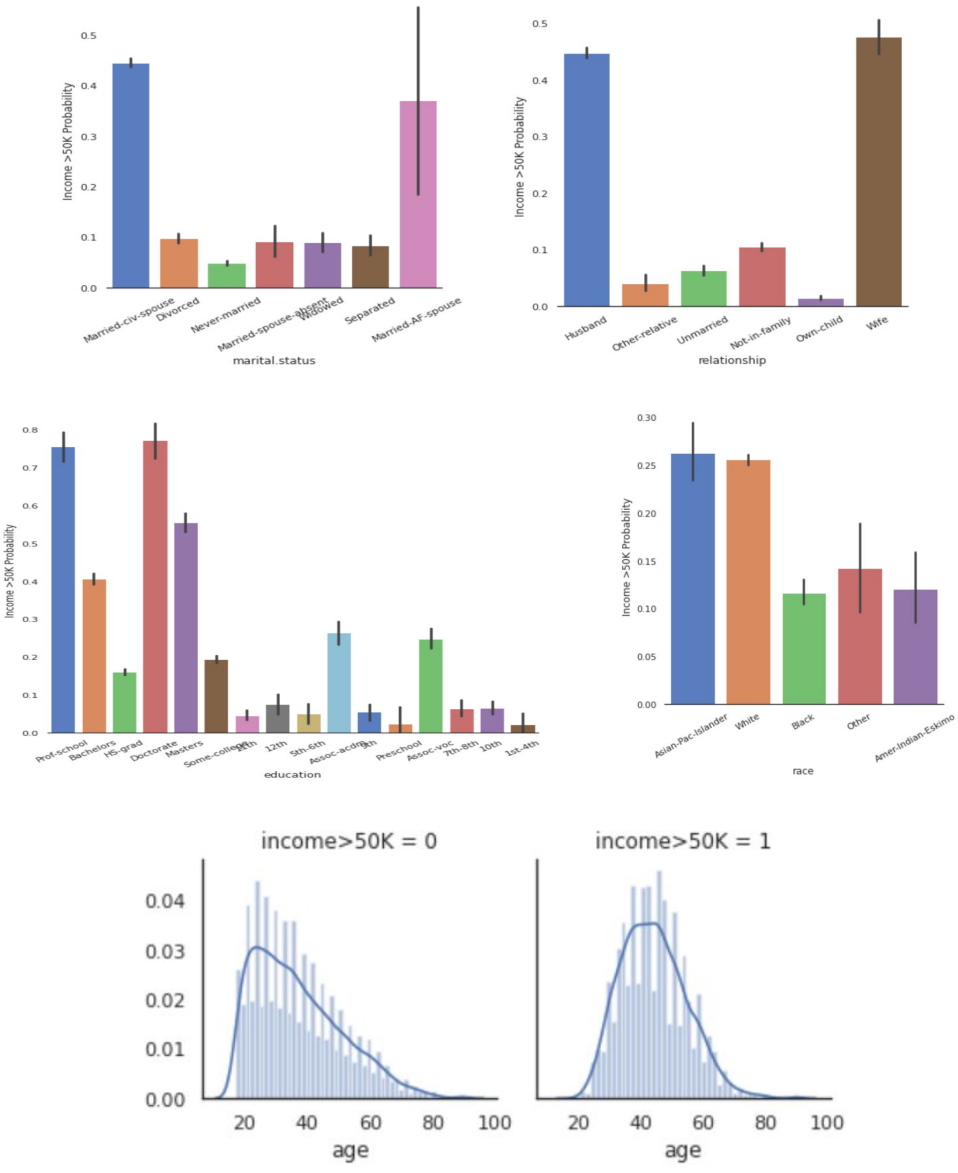
Attributesl Coutns	Number of missing values	Percentage of Missing Value
Workclass	1,362	5.71%
Occupation	1,367	5.73%
Native.country	430	1.80%

Table 2: Test data's missing values

Then, we replaced ['?'] with the most frequent/count value for that column to complete the dataset.

2.1.2 Features Engineering

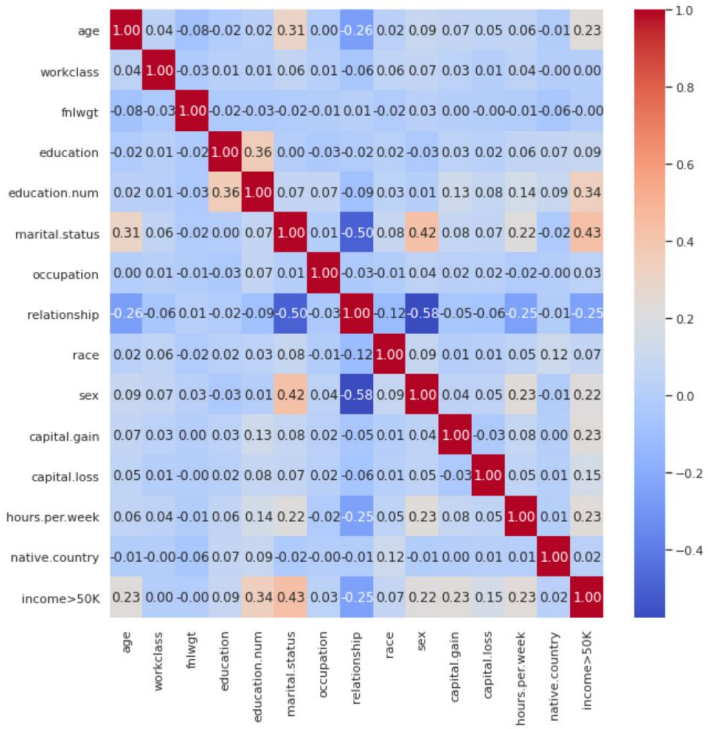
Below are some graphs showing a closer look into each individual attributes



As we can see, some attributes such as 'education' has many values within itself, they could be group together. For examples, they could be group as 'college' and 'no college'. On the other hand, some attributes are more indicative or highly correlated with income than others. It is always useful to know which ones has higher correlation with the target.

Below is the tables which shows the correlations from highest to lowest. Features that are more correlated with income are ‘education.num’, ‘relationship’, ‘hours.per.week’, ‘capital.gain’, ‘age’, ‘sex’, ‘marital.status’,’capital.loss’ Other features such as ‘fnlwgt’, ‘workclass’, ‘native.country’, ‘occupation’ does not seem to be highly correlated and can be drop.

We also notice that features such as (education and education.num), (relationship and marital status) are highly correlated with one another. Correlated features in general don’t improve models. Hence, we should keep one and drop the other to simplify the model. This can be viewed as a special case of Occam’s razor. A simpler model is preferable, and, in some sense, a model with fewer features is simpler.



As we can see, some features are also repetitive. For example, marital status has values such as ’Never-married’, ’Divorced’, ’Separated’, ’Widowed’. They can be group into one group called ’Single’. ’Married-civ-spouse’, ’Married-spouse-absent’, ’Married-AF-spouse’ can be grouped into one group called ’Married’. Then we map the first group to 0 and the second group to 1.

2.1.3 Imbalance class

Another observation here is that we have imbalance class of income. This is something we have to take into consideration when training the model.

Income LevelCounts	Number of outcomes	Percentage of outcomes
Income <50K	18,984	76%
Income >50K	6,016	24%

Table 3: Imbalance class in the dataset

The model can return high accuracy score when trained with imbalanced class but fail when test with new data. This is because our model looks at the data and cleverly decide that the best thing to do is to always predict the "majority class". This can done by trying out couple of different techniques:

- Resampling the dataset: We can add copies of instances from the under-represented class, called over-sampling or we can delete instances from the over-represented calss, called under-sampling.

- Use different measurement to measure accuracy score such as fraction measurement and F score instead of accuracy score
- Try penalized models: SVM is penalized algorithm that can be used in this case.

2.1.4 Normalizing data

We also normalize all numerical features by subtracting the mean and dividing by the variance. The purpose of feature normalization is to make different features in the same scale. The scaling speeds up gradient descent by avoiding many extra iterations. This is crucial espcecially training with neural net. In the case of different ranges of features, there will be problems with model training. If we don't scale features into the same ranges then features with larger values will be treated as more important in the training, which is not desired. What is more, the gradients values can explode and the neurons can saturate which will make it impossible to train.

Next, we convert categorical attributes into numerical before putting it into the model. As we are using Scikit-learn for modeling, it is important to convert these into dummy values as it mostly performs on numerical data.

2.2 Methods Used and Results

Please check out the codes here: https://github.com/hanambrose/Machine-Learning_CS6350/blob/master/Final%20Project/Income_Prediction_Main.ipynb

2.2.1 Ensembles

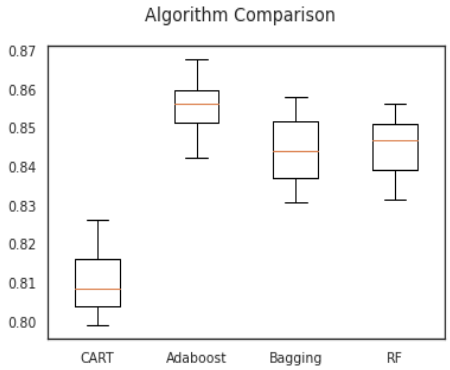
First we tried a couple of ensemble methods. The reason is because ensemble methods handle categorical data very well. Before applying any regression methods, we are unsure whether or not our data is linearly separable. The data could be non-linear and therefore using any linear classifier method would not be a good idea. On the other hand, ensemble methods allows us to capture many of these non-linear relationships, which translates into better prediction accuracy on the problem of interest

(a) Bagging
We draw m samples from the training set uniformly with replacement from the training set , denoted by X_t, Y_t. Then the model will learn a decision tree C_t from X_t, Y_t , using ID3 or CART. We then repeat this step T times to make T trees. Prediction is the vote or average T predictions for classification or regression, respectively.

(b) Random Forest
This method is similar to bagging method. However, we randomly select the number of attributes.
We draw m samples from the training set uniformly with replacement from the training set , denoted by X_t, Y_t. The difference with bagging is right here when we build a tree, we draw a subset of attributes at each split. Then the model will learn a decision tree C_t from X_t, Y_t , using ID3 or CART. We then repeat this step T times to make T trees. Prediction is the vote or average T predictions for classification or regression, respectively.

(c) AdaBoost
AdaBoost works by putting more weight on the error to classify instances and less on those already handled well.

Below is the results after using 10-fold Cross validation and obtain the accuracy scores. Adaboost seems to work best.



2.2.2 Linear to Non-Linear Classifiers

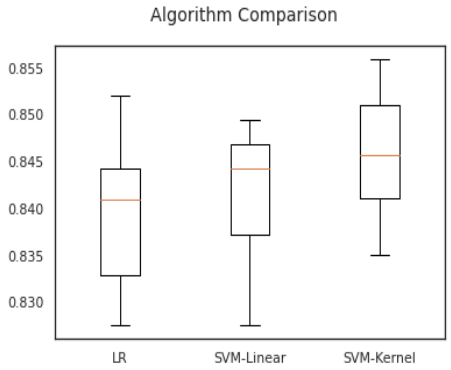
These two models are similar in a way that they are both optimization problem. They both have a regularizer and a loss function. They both penalize if the sammple is not correctly classified.

(a) Logistic Regression

Logistics Regression uses Log Loss (Cross-Entropy)

(b) Soft SVM

SVM uses hinge loss to penalize



SVM performs better than Logistic Regression and Kernel SVM is better than Linear SVM.

2.2.3 Neural Network

Please check out the codes for neural Net: https://github.com/hanambrose/Machine-Learning_CS6350/blob/master/Final%20Project/neural_net.ipynb

The model was built with 3 layers, equal width in hidden layers. The activation='relu', kernel_initializer= 'he_normal', optimizer =opt, loss = 'mean_squared_error'. Dropout was also used to reduce overfitting.

It is hard to tune as there are many hyperparameters to consider and changing one could drastically change the outcome. Compare to other methods, it underperforms, it could be due to hyperparameters tuning techniques and it takes more experiments to get to the optimal outcomes.

Summary As we point out that income is imbalanced so it makes identification of the minority class by a learner challenging because a high-class imbalance introduces a bias in favor of the majority class. Consequently, it becomes quite difficult for the learner to effectively discriminate between the minority and majority classes, yielding a task comparable to searching for the needle in a haystack.

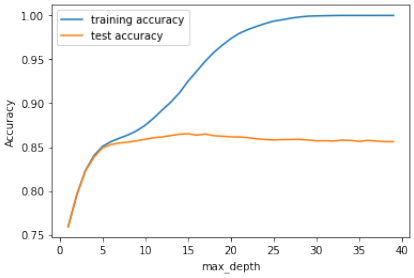
All of the methods above yield very similar results in term of accuracy. Ensembles seems to work better. However, tuning hyperparameters could improve prediction.

2.3 Tuning models

(a) Cross validation to tune hyperparameter

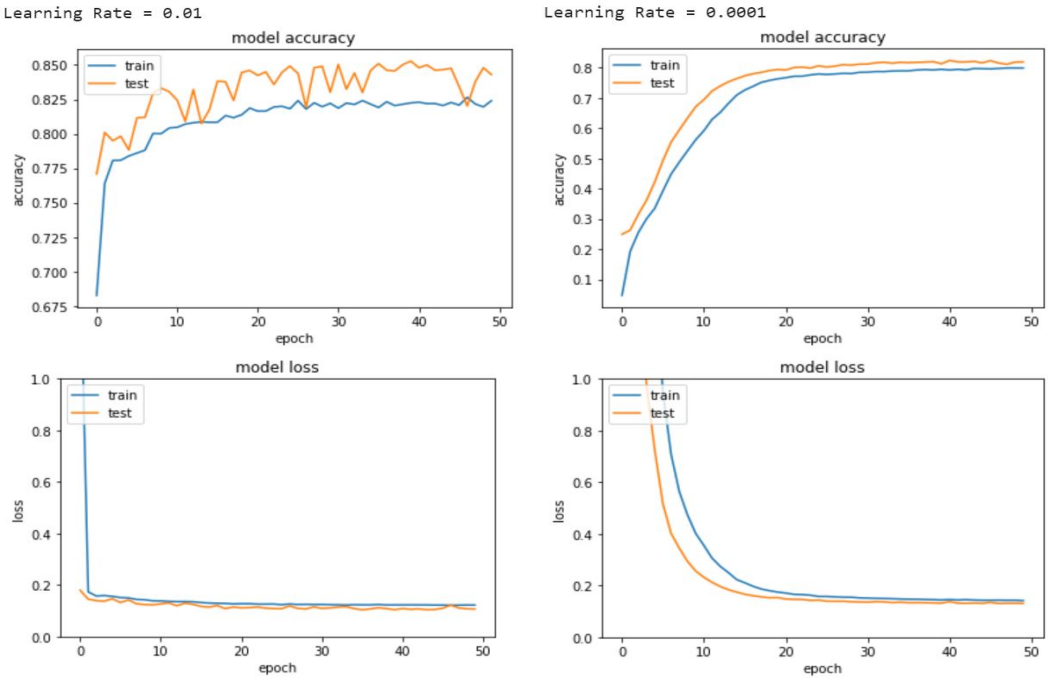
As we know decision tree tends to overfit. Another way to test this is to do cross validation. Cross validation is a good method to see how the model performs on unseen data. We split the data into K (say 5) equal parts. We then train a classifier on four parts and evaluate it on the fifth one. We repeat this using each of the K parts as the validation set. The quality of this hyper-parameter is the average of these K estimates.

Below is an example when we applied this method to select optimal maxinum tree depth for random forest classifier for 5-fold cross validation, we got the optimal max tree depth is about 5. As we can see the model overfits after maximum depth goes beyond 5.



(b) Learning Rate

Below is the comparison of different learning rates learned from neural net model using Adam as optimizer. It is important to have a good learning rate as we do not want to model to learn too fast or too slow. The left figures (top and bottom) with larger learning rate converge too fast. On the other hand, the right figures show better steady convergence.



3 Conclusion and Ideas for improvement

Overall, ensembles seems to work better. It is important to preprocess data before putting it through any models. On the other hand, tunning models using hyperparameters could significantly improve the model performance. Another idea other than tuning hyperparameters that we could try is to focus more on feature engineering. Feature engineering involves the careful selection to feed the model only the most optimal form of input. If we can give the model only the parts of the data it needs to make accurate predictions, then it does not have to deal with any extra noise that comes from the rest of the data.