**Assignment 2 Report for COMP5318**

**Machine Learning and Data Mining**

Comparison of Machine Learning Algorithm Performace on UCI Adult Income Dataset

Mohammad Danial Azam (460343558), Muhammad Arshad (460337384), Muhammad Farhan Munir (460238324), Geng Yuan (450621024)

May 30, 2016

# **Introduction**

In this report we will be investigating income data set from below UCI repository: http://archive.ics.uci.edu/ml/machine-learning-databases/adult/. The data provided in the above data sets contains 14 features. These features namely are: Age, Workclass, fnlwgt (final weight), Education, and Education-Num (Year of Education), Marital Status, Occupation, Relationship, Race, Sex, Capital Gain, Capital Loss, Hours per week, Country.

In this report we will be exploring the data, implementing some pre-processing methods and work towards building multiple machine learning models to try and predict if a particular person will earn more than $50K per year or less.

# **1.1 Initial Data Analysis and Pre-Processing**

We have 32561 training records and 16281 records for testing. If we exclude the missing values we get a split of 30162 to 15060 records between training and test. Our dataset contains a mixture of categorical and continuous variables. Out of our 14 features 8 are categorical features while the remaining 6 are continuous features.

**Handling Missing Values:** We have observed that around 6% of data has missing values, there are 1979 instances where one or more variables are missing in a record. An interesting observation is that whenever “Work Class” is null “Occupation” is also null with it.

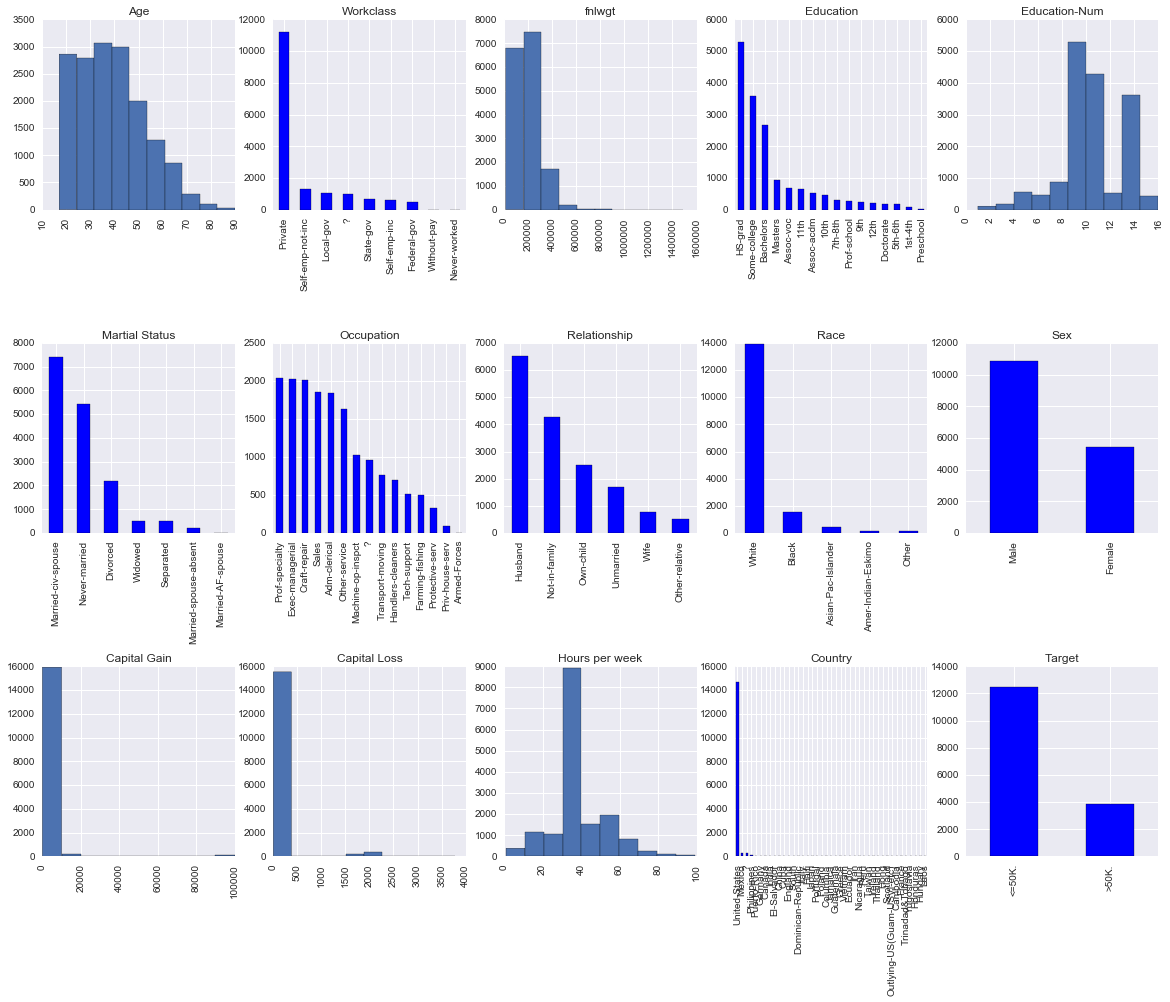
**Approaches:** To handle missing values we have gone ahead with two approaches.  
1. Approach A- Treating missing values as a separate class for categorical variables.  
2. Approach B- To use Mode of the particular category to replace the missing variable.  
Results for both approaches are displayed in the Results section for comparison.

**Scaling and Normalization:** Before implementing our classifier, we would also like to scale and normalize all our features with a constant mean of 0 and variance of 1. We have used Scikit-Learn’s Standard Scalar module to implement scaling of our variables.

**Removal of DOTs in Target Column:** The target column containing the labels and the native country column both contain dots which need to be striped since they could cause problems while parsing the data.

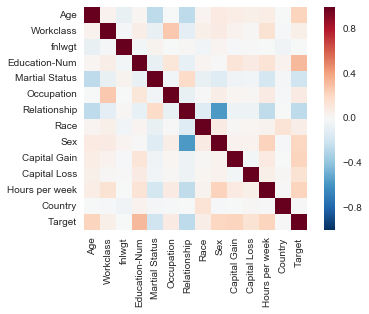
# **1.2 Distribution**

We have plotted the distribution of all of our variables using a histogram for each continuous as well as categorical features (figure 1). This will help us better understand our data. As we can see that 89% of the records are of residents of the United States while the next country on the list comes to around 2 %.



# **1.3 Correlation Analysis**

We have observed that there is a high correlation between the variables ‘Education’ and Education-Num. On further investigation we have found that both these variables represent the same thing. ‘Education’ is stored as a string variable while ‘Education-Num’ is stored as numbers. We would prefer to use ‘Education-Num’ as number have the property of being ordered; implying that higher the number the higher the education a person has. Similarly there is a low correlation between sex and relationship attributes.



# **Data Files Processing**

Anaconda’s Jupyter notebook environment is used for Python implementation.

Below is the process used to load the data files in Python:

• Download the Training and Test Data from the UCI repository using the URI

• Load the Training data and Test Data including missing values into data frames using the Pandas library. The training data is contained in ‘training\_data’ and test data in ‘test\_data’

• Load a separate Training and Test data set with empty values replaced with NaNs.

• Apply Machine learning models using training\_data for training and test\_data for predicting labels.

# **Methods and Approach**

We have use Logistic Regression, SVM(C-Support Vector kernel=rbf), Gaussian Naive Bayes, Adaboost and Random Forest. Table below summarises the key features of these algorithms. The key reasons for using these particular algorithms is summarised in the table below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Algorithm | Average predictive accuracy | Training speed | Prediction speed | Prone to over fitting | Performance w/ small no. of data? | Ease of Explanation |
| Logistic regression | Lower | Fast | Fast | Moderate | Maybe | Somewhat |
| Naive Bayes | Lower | Fast (excluding feat. extraction) | Fast | High | Yes | Somewhat |
| Adaboost | Higher | Slow | Fast | Low | No | No |
| SVM | Higher | Slow | Moderate | Low | No | Yes |

## **Logistic Regression Classifier**

Lots of ways to regularize your model, and you don’t have to worry as much about your features being correlated, like you do in Naive Bayes. You also have a nice probabilistic interpretation. We used GridSearchCV for fine tuning of our parameters for Logistic Regression. GridSearchCV ensured we used optimal values for our parameter calculations. The parameters were as follows:

Penalty = ‘l1’, C= ‘1’.

## **SVM**

SVM is a robust classification method and requires minimal parameter tuning. Sklearn’s SVM.SVC was used for benchmarking. Following key parameters were used for fitting the model after parameter tuning was done using GridSearchCV:

## **Adaboost Classifier**

In some problems like this one, Adaboost can be less susceptible to the over fitting problem.  The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing the final model can be proven to converge to a strong learner. For Adaboost classifier we used the following parameters for optimising our results:

num\_estimators =50, learning rate = 1.0, random state = none.

## **Naïve Bayes Classifier**

Naive Bayes classifiers parameter tuning is limited hence we focuses on the pre processing techniques discussed in section 1.1 of our report. Our pre-processing techniques have included handling of missing values, removal of high correlated values and outlier detection

# **Evaluation**

Following metrics were used for evaluation:

|  |  |
| --- | --- |
| **Metric** | **Formula** |
| Accuracy |  |
| Precision |  |
| Recall |  |
| F1-Score |  |

**Table 1** Evaluation Metrics

We have noticed that using the imputed data the accuracy of all the algorithms has not improved. One of the underlying reasons could be that by imputing the missing values with the mode or median, we are introducing bias in the data. This can be backed up with the results in the second table.

The following tables demonstrate results from our two approaches:  
Approach A – Original Dataset with missing values.  
Approach B - Dataset with imputed values

**Approach A**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Logistic Regression** | **SVM (SVC)** | **Adaboost Classifier** | **Gaussian NB** |
| F1 score | 0.547 | 0.634 | 0.671 | 0.500 |
| Accuracy score | 0.825 | 0.851 | 0.861 | 0.833 |
| Precision score | 0.704 | 0.756 | 0.758 | 0.671 |
| Recall | 0.447 | 0.546 | 0.603 | 0.322 |
|  |  |  |  |  |
| **Approach B** |  |  |  |  |
| F1 score | 0.540 | 0.631 | 0.667 | 0.477 |
| Accuracy score | 0.826 | 0.850 | 0.861 | 0.809 |
| Precision score | 0.700 | 0.751 | 0.766 | 0.675 |
| Recall | 0.432 | 0.544 | 0.591 | 0.369 |

# **Discussion**

To clarify we will be going ahead with the Original data set and not the one with imputed values since we feel like that introduces a bias in the data set and may also result in over fitting.

As for our results in our original approach as from figure x:

Adaboost has outperformed other algorithms in terms of accuracy and precision. Adaboost is an ensemble machine learning algorithm which works on the notion of first finding a weak or base classifier. Then iteratively going through the training data set. For each iteration it assigns higher weights to records which were misclassified in the earlier iteration. This gives it the power to predict variables which are inherently difficult to learn and predict.

SVM (SVC) is closely second to Adaboost since it has a nice way of avoiding over fitting as compared to NB and Logistic Regression, It is also an iterative algorithm which works on the notion of maximising the separation between records of different classes.

Results from our other two algorithm are promising but not performing as well as Adaboost and SVM because these algorithms are prone to outliers. We have applied an outlier detection in method on the data set and found that certain classes such as Capital Gain contain substantial amount of outliers. They are also not iterative meaning that they are not suitable to learn from misclassified records.

# **Conclusion**

# **Appendix**

## **Hardware and Software Specifications for Performance Evaluation**

* Processor : Intel ® Core i7-6600U CPU @ 2.60GHz
* RAM Memory : 8 GB
* Operating System : Windows 10 64 Bit
* Implementation Language : Python 3.4
* Development Environment : Jupyter Notebook

## **Libraries Requirement**

The following libraries are required (imported) during the execution of the code:

* Numpy
* Sklearn
* Scipy
* Collections
* matpoltlib (required to produce some graphs)
* time
* Seaborn

## **Run Code**