INFO-I526 APPLIED MACHINE LEARNING – SPRING 2017
Carlos Sathler (cssathler@gmail.com)

Final Project Report

1. Project Goals

Our project analyzes the Adult dataset described at the site http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names. The prediction task associated with this dataset is to predict whether or not a person makes more than $50K a year using census data. Our project attempts to answer this question: can we implement a classifier using sklearn that will produce accuracy comparable to the best algorithm implemented by the publishers of the data? In other words, **can we implement a solution using sklearn that will yield accuracy comparable to 0.8595 obtained with a NSS Naïve Bayes classifier**, as announced by the publishers of the data?

In order to answer this question we used 7 classifiers available in sklearn, taking the time to experiment various hyper-parameter tuning options for each classifier. We also trained and tested each algorithm on 5 different versions of the dataset obtained through 5 distinct pre-processing routines. Tuning and pre-processing are described below.

The 7 sklearn classifiers we used are: Naïve Bayes, Decision Tree, K-NN, Logistic Regression, SVM, Random Forest and Adaboost. Additionally, we implemented a custom ensemble classifier that used the output of these sklearn classifiers to produce a prediction based on weighted "voting". We will show below that **our custom ensemble classifier ended up yielding the best accuracy metric** and that it was just slightly below our benchmark 0.8595 value.

Project deliverables: Final Project Report (the present document), python code (adult_final.py, plotdata.py and plotroc.py), metrics data file (adult_metrics.csv), python program report (adult_final.out). All project deliverables were uploaded to Canvas.

2. Implementation details

We used the following datasets available from http://archive.ics.uci.edu/ml/datasets/Adult: adult.data (training dataset) and adult.test (test dataset). Please refer to Appendix B for a number of plots from Weka describing the data.

The following pseudo-code describes the project main program implementation (adult_final.py).

```
Pre-process Train and Test dataset in 5 different ways, creating 5 versions of datasets.
For each sklearn classifier:
    For each dataset version:
        Create train' and test' partitions by splitting the Train dataset (80/20% split)
        Use 10 fold validation to tune algorithm hyper parameters using train' partition of Train dataset
        Calculate mean f1-score from 10 fold validation exercise for each hyper parameter setting
        Use best hyper parameter setting to retrain the model on the entire train' partition
        Test model on test' partition and calculate f1-score for dataset version
        Capture dataset version yielding the best score
    Retrain model with best hyper parameter setting on the best performing Train dataset version
    Test model on Test dataset of the same version (identical pre-processing)
    Capture results
Train and test custom ensemble algorithm that makes predictions based on weighted "voting" using prediction from the other algos
Save all metrics to file "adult_metrics.csv"
Print results
```

3. Pre-processing approach

Data cleaning and conversion

We identified missing values (column value = "?") in the following columns: workclass, occupation and native_country. These are all categorical features. Even though we could have used mode as a measure of centrality to impute missing values we opted to drop rows containing missing data to avoid adding bias to the dataset.

After removing missing values we performed the following data cleaning and conversion tasks:

- Removed rows with missing values in columns "workclass", "occupation" and "native_country"
- Dropped column "education" since it's redundant (it matches column "education-num")
- Converted all categorical columns to numbers
- Converted all columns to floats

Additional pre-processing: discretization, standardization and use of dummy variables

Initial research on the options to pre-process the data for each algorithm revealed there are no easy choices for when to discretize variables, standardize or use dummy variables for each algorithm. Some algorithms outperformed when fed data that theoretically should yield poorer results. For example, the tree classifier performed better when we used continuous variables, even though one would expect it to perform better on discretized features (since calculation of entropy require discrete features).

The sklearn man pages do not make any recommendations about how to pre-process features for each algorithm, and from our research there are no easy answers, or cookie-cutter approach to drive pre-processing of various feature types for each algorithm. It is reasonable to assume a lot will happen "under the hood" depending on how each sklearn algorithm is implemented.

With that in mind, we decided to use an empirical approach to pre-processing. After data cleaning and conversion we chose to create 5 versions of the dataset using 5 different pre-processing routines. Each algorithm was evaluated on each version of the dataset, to see which one worked best for each algorithm. The table below describes each pre-processing routine. An additional table in Appendix B describes in detail how each pre-processing version affects/transforms each feature in the dataset.

Dataset Version	Continuous Feature	Ordinal Features	Categorical Features
Pre-processing Version 0	Leave unchanged	Leave unchanged	Leave unchanged
Pre-processing Version 1	Discretize	Leave unchanged	Leave unchanged
Pre-processing Version 2	Standardize	Standardize	Leave unchanged
Pre-processing Version 3	Discretize	Leave unchanged	Create dummy vars
Pre-processing Version 4	Standardize	Standardize	Create dummy vars

As we will report in Section 5 below most algorithms performed best with dataset version 4, i.e., standardized numeric features (continuous and encoded ordinal education number) and dummy variables for categorical features. Exceptions were Naïve Bayes, which performed best with dataset version 1, and Adaboost, which performed best with dataset version 2. We attribute these results to sklearn classifier implementation details whose explanation is beyond the scope of our project. Additionally, we noticed different results for successive runs of Adaboost and Random Forest due to the randomness inherent to these algorithms (we opted not to use the parameter "random_state").

4. Hyper-parameter tuning

The following table lists hyper parameters tuned for each algorithm/dataset version along with settings that yielded best performance:

Algorithm	Parameter	Best Hyper-Parameter Setting
Naïve Bayes	None in sklearn	No hyper-parameter tuning
Decision Tree	Tree Depth	Max Tree Depth = 15
KNN	K neighbors	Neighbors = 21
Logistic Regression	Penalty (l1, l2) and C	Penalty = l1; C = 1.0
SVM	C	C = 1
Adaboost	Max tree depth Number of trees	Max Tree Depth = 6; Estimators = 10
Random Forest	Max tree depth Number of trees	Max Tree Depth = 12; Estimators = 50

Each sklearn classifier was tuned with 10 kfold validation and tested on the train dataset validation partition (20% of training data). Instead of accuracy we used F1 Score for performance evaluation since the data is skewed (only ≈ 29% of the population makes more than $50K). After identifying the best hyper-parameter value for each classifier we retrained the model in the entire training dataset and captured results to analyze overfitting problems. As explained in Section 3, we tuned each classifier against 5 versions of the data, each version defined by how the data was pre-processed. So, we evaluated not only parameter setting for each algorithm, but also what pre-processing yields the best results for each algorithm.

Training results appear in the program output posted to Canvas (adult_metrics.csv) under the title "TRAIN DATASET RESULTS"; test results are listed under the title "TEST DATASET RESULTS". The results identify best dataset version (best pre-processing) for each algorithm, along with best hyper parameter values for the algorithm/dataset.

5. Performance comparison

The table below summarizes our results. Best performance in each metric in the train/test dataset is highlighted in bold/pink.

For a visual display helping compare results for each classifier please refer to the plots in Appendix A. We attribute performance differences among the classifiers to undetermined characteristics of the dataset features and their relationship to each other and the target label.

Algo	Best Dataset	Accuracy		Precision		Recall		F1 Score		ROC
		Train	Test	Train	Test	Test	Test	Train	Test	Test
KNN	4	0.8549	0.8452	0.7407	0.7120	0.6417	**0.6214**	0.6876	0.6636	0.8926
LOGIT	4	0.8489	0.8481	0.7369	0.7309	0.6113	0.6041	0.6683	0.6614	0.9030
SVM	4	0.8514	0.8527	0.7588	0.7582	0.5911	0.5881	0.6645	0.6624	**0.9050**
DECISION TREE	4	**0.8996**	0.8386	0.8277	0.6914	**0.7536**	0.6200	**0.7889**	0.6537	0.8505

Algo	Best Dataset	Accuracy		Precision		Recall		F1 Score		ROC
		Train	Test	Train	Test	Test	Test	Train	Test	Test
NAIVE BAYES	1	0.7947	0.7960	0.5778	0.5823	0.6504	0.6003	0.6119	0.5912	0.7930
ADABOOST	2	0.8767	0.7886	0.7874	0.6410	0.6913	0.3170	0.7362	0.4242	0.7795
RANDOM FOREST	4	0.8694	0.8558	**0.8312**	0.7924	0.5967	0.5600	0.6947	0.6562	0.8856
ENSEMBLE (CUSTOM)	N/A	0.8667	0.8564	0.7891	0.7731	0.6337	0.5884	0.7029	0.6682	N/A

Our custom ensemble classifier yielded the best accuracy and best F1 Score. This custom ensemble algorithm predicts target value by performing a weighted vote count using the results obtained from the 7 sklearn classifiers. The parameter used to weigh votes across the classifiers is the F1 score metric obtained during tuning of each algorithm. In terms of overall performance measured by area under the ROC curve the SVM algorithm was the best. ROC curves are included in Appendix A.

Note that while the custom ensemble algorithm produced the best accuracy and F1 Score, Random Forest outperformed all algorithms on the precision metric and KNN was the best algorithm for recall. In a production setting algorithm selection would call for thoughtful consideration of the problem at hand. For example, if the purpose of our project were to identify potential charity donors, KNN would be the best choice. For problems where cost of False Positives may be an issue, Random Forest will be the best option.

Regarding overfitting, Adaboost shows significant overfitting with a performance reduction of more than 10% in accuracy. But different runs of the algorithm generated better results highlighting the randomness aspect of Adaboost. The same can be said about successive runs of the Random Forest algorithm. Even the best dataset version changed in successive runs of Adaboost and Random Forest. Due to space limitations we do not document these results herein.

The Decision Tree classifier also overfit. We were able to obtain better results for this classifier on the test dataset by using a slightly lower value for max tree depth. In a production setting we would run the algorithm in the test set using a slightly lower value for the max tree depth parameter and reasonably expect better results in the test dataset.

6. Conclusion

The answer to the project question "can we implement a solution using sklearn that will yield accuracy comparable to 0.8595 obtained with a NSS Naïve Bayes classifier?" is clearly **yes**. In fact 2 of the 7 sklearn classifiers produced accuracy greater than 85% and therefore very close to our benchmark. Our pre-processing approach seemed effective because it provided a structured framework to compare pre-processing options.

We were able to improve results from the best sklean classifiers by creating our own custom ensemble algorithm that implemented weighted voting using results from the sklearn algorithms. The NSS Naïve Bayes accuracy result published in the Adult site (http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names) is only 0.0036 (0.36%) superior to the result we obtained. It is reasonable to think that with more time we might be able to match or even beat the Adult site results.

Finally, it is important to note that because the census data for the project is somewhat skewed, precision, accuracy and F1 score are likely superior metrics to evaluate performance of classifiers on this data, depending on the problem at hand.

References:

[1] McKinney, Wes. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc.", 2012.

[2] James, Gareth, et al. An introduction to statistical learning. Vol. 6. New York: springer, 2013.

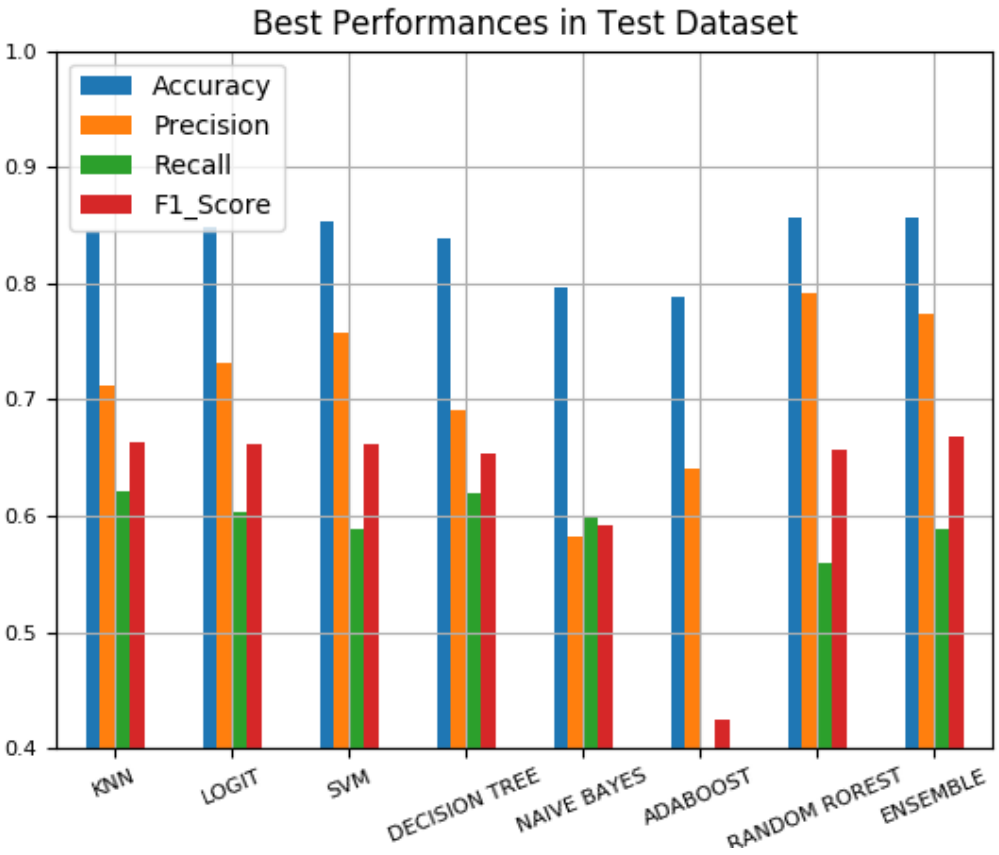
[3] Raschka, Sebastian. Python machine learning. Packt Publishing Ltd, 2015.

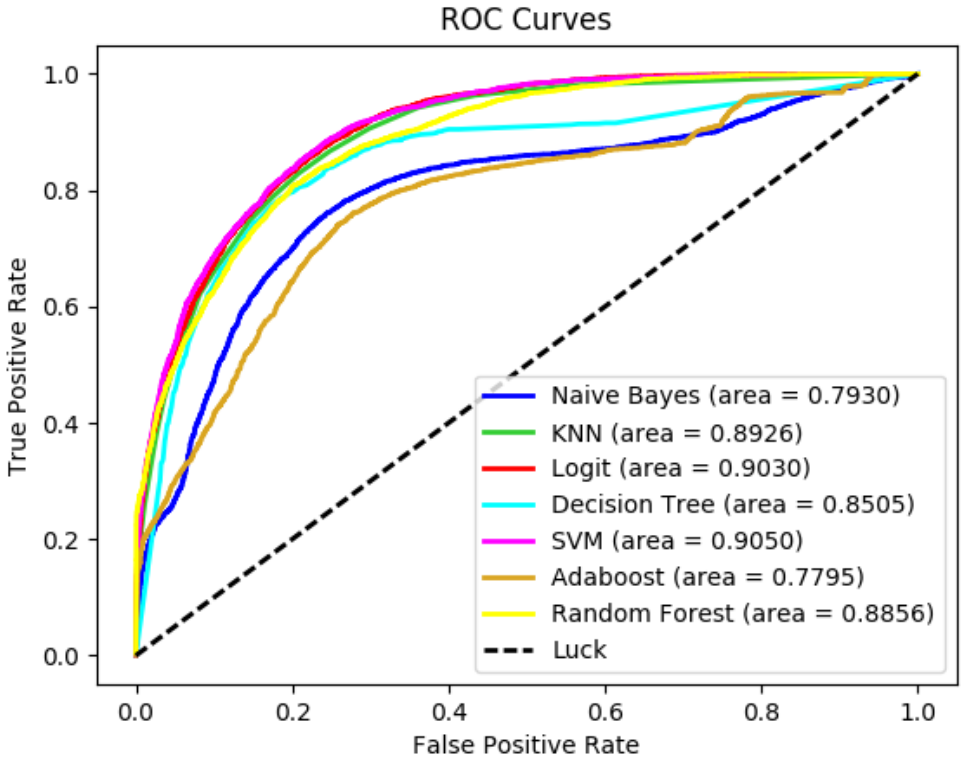
[4] Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning. Vol. 1. Springer, Berlin: Springer series in statistics, 2001.

[5] Downey, Allen. Think Python. " O'Reilly Media, Inc.", 2012.

[6] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

APPENDIX A

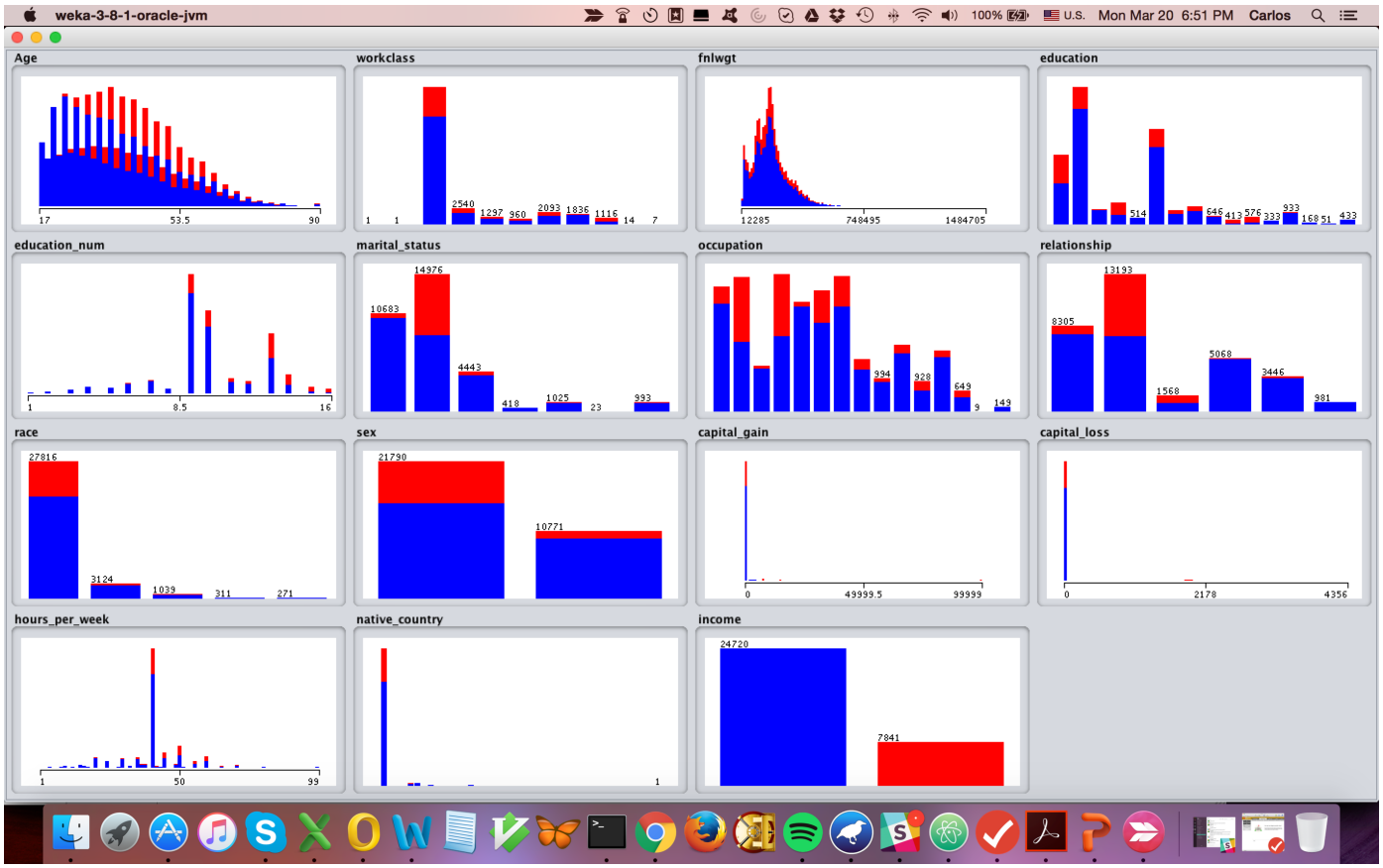

Best Performances in Test Dataset
Accuracy
Precision
Recall
F1_Score
1.0
0.9
0.8
0.7
0.6
0.5
0.4
KNN
LOGIT
SVM
DECISION TREE
NAIVE BAYES
ADABOOST
RANDOM RORES
ENSEMBLE


ROC Curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Naive Bayes (area = 0.7930)
KNN (area = 0.8926)
Logit (area = 0.9030)
Decision Tree (area = 0.8505)
SVM (area = 0.9050)
Adaboost (area = 0.7795)
Random Forest (area = 0.8856)
Luck

INFO-I526 APPLIED MACHINE LEARNING – SPRING 2017

Carlos Sathler (cssathler@gmail.com)

APPENDIX B



Pre-processing transformations to dataset features

Feature	Type	General preprocessing	Pre-Pro 0	Pre-Pro 1	Pre-Pro 2	Pre-Pro 3	Pre-Pro 4
age	Numeric (Continuous)	Convert to float	No change	Discretize	Standardize	Discretize	Standardize
workclass	Categorical	Convert to float	No change	No change	No change	Dummy vars	Dummy vars
fnlwgt	Numeric (Continuous)	Convert to float	No change	Discretize	Standardize	Discretize	Standardize
education	Categorical	Remove (redundant)	N/A	N/A	N/A	N/A	N/A
educational-num	Numeric (Ordinal)	Convert to float	No change	No change	Standardize	No change	Standardize
marital-status	Categorical	Convert to float	No change	No change	No change	Dummy vars	Dummy vars
occupation	Categorical	Convert to float	No change	No change	No change	Dummy vars	Dummy vars
relationship	Categorical	Convert to float	No change	No change	No change	Dummy vars	Dummy vars
race	Categorical	Convert to float	No change	No change	No change	Dummy vars	Dummy vars
gender	Categorical	Convert to float	No change	No change	No change	Dummy vars	Dummy vars
capital-gain	Numeric (Continuous)	Convert to float	No change	Discretize	Standardize	Discretize	Standardize
capital-loss	Numeric (Continuous)	Convert to float	No change	Discretize	Standardize	Discretize	Standardize
hours-per-week	Numeric (Continuous)	Convert to float	No change	Discretize	Standardize	Discretize	Standardize
native-country	Categorical	Convert to float	No change	No change	No change	Dummy vars	Dummy vars
income	Binary flag	Convert to float	N/A	N/A	N/A	N/A	N/A