Machine Learning Learning

An algorithm comparison based on the Adult Dataset

Research Paper

by

Martin Vogel

Matriculation number: 11009471

2017-12-24

SRH University Heidelberg

School of Information, Media and Design

Degree course "Applied Computer Science"

Major field "Business Computing"

Reviewers	Dr. Simon Ziegler

Table of Contents

1 Introduction

This document has been created within the course “Applied Artificial Intelligence” taking place at SRH Hochschule Heidelberg from November 20th to December 24th 2017. It describes the process of analyzing a set of data with the algorithms introduced in the mentioned course. Any code developed for this can be found in the following git repository: https://github.com/MartinVogelSRH/Applied-AI

1.1 Objective

The objective of this document is to analyze the Adult Dataset hosted by the UCI Machine learning repository (Lichman 2013). A python script will get created which will analyze the dataset. During the development of the mentioned script, effort will be taken so that the script is as generic as possible and can be used with other datasets with minimal code adjustments.

1.2 Methodology

As mentioned beforehand, this project will aim for the generation of a python script to analyze the Adult Dataset. This script will be tested on Anaconda 4.4.0, Python 2.7 and Python 3.6. The following packages have to be available to run the script:

scipy (Jones et al. 2001-)

numpy (van der Walt et al. 2011)

matplotlib (Hunter 2007)

pandas (McKinney 2010)

sklearn (Buitinck et al. 2013)

seaborn (Michael Waskom et al. 2017)

2 Dataset Description

2.1 Getting to know the data

When starting any machine learning project, it is important to have a clear idea about the dataset to be used and the predictions that are supposed to be made. This project will deal with supervised learning. Supervised learning means that there is data available which has already been classified correctly upfront and can be used for training and for testing the used algorithms. The dataset this project will be based on is the Adult Dataset hosted by the UCI Machine learning repository (Lichman 2013).

This dataset is based on Census data and is being used in order to predict, whether a person earns more than 50 thousand US-Dollar or not. An outstanding feature in this regard is “fnlwgt”, which is described as follows: “The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US.” (UCI Machine Learning Repository: Adult Data Set). As of this, this feature contains an estimation of how many individuals are reflected by this data set. For the predictions used in this project, it will therefore be ignored and deleted from the dataset right after loading it:

```
#This code loads the dataset itself and defines the headers.
    headerNames = ['age', 'workclass', 'fnlwgt', 'education', 'education-
num', 'marital-status', 'occupation', 'relationship', 'race', 'sex',
'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'class']
    print('Downloading Data...')
    dataset = pandas.read_csv('adult.data', header=None, sep=',',
names=headerNames)

    print("Download complete. The fnlwgt feature is not needed for our
prediction. Let us delete it.")
    del dataset['fnlwgt']
    del headerNames[2]
    print('------------------------------------------')
```

Listing 1 - Loading the Dataset

2.1.1 Graphical feature Representation

In order to get an idea of how the dataset is built up and to get a rough overview on the data, a graphical representation gets created by the method `generateOverview` . This method will create an overall overview of the features and their statistics and export it into the file overview.png . In order to do so, the method uses the integration between matploplib and pandas. Numerical features will be represented as histograms, while categorical features will be represented by bar charts. This gives the user a clear indication about the type of data.

```
for i in range(0,numberOfCols):
    for j in range(0,numberOfRows):
            currentHeader = i + j + (i*(numberOfCols-1))
            if (currentHeader < len(headerNames)):
                #If we have values that are numerical, a histogram is a good
option for plotting.
                #For features that are categorical, a barchart is beneficial
                if dataset.dtypes[headerNames[currentHeader]] ==
numpy.object:
                    dataset.groupby(headerNames[currentHeader]).size().plot(
kind='bar', ax = splts[i,j])
                else:
                    dataset[headerNames[currentHeader]].hist(ax = splts[i,j]
)
                    plt.xticks(rotation="vertical")
                splts[i,j].set_title(headerNames[currentHeader])
                splts[i,j].set_xlabel('')
#This makes the output a little bit nicer by adjusting the borders of the
subplots
plt.subplots_adjust(left=0.05,  top=0.95,  bottom  =  0.2,  right  =  0.95,
hspace=1, wspace=0.3)
```

Listing 2 - Generating graphical overview

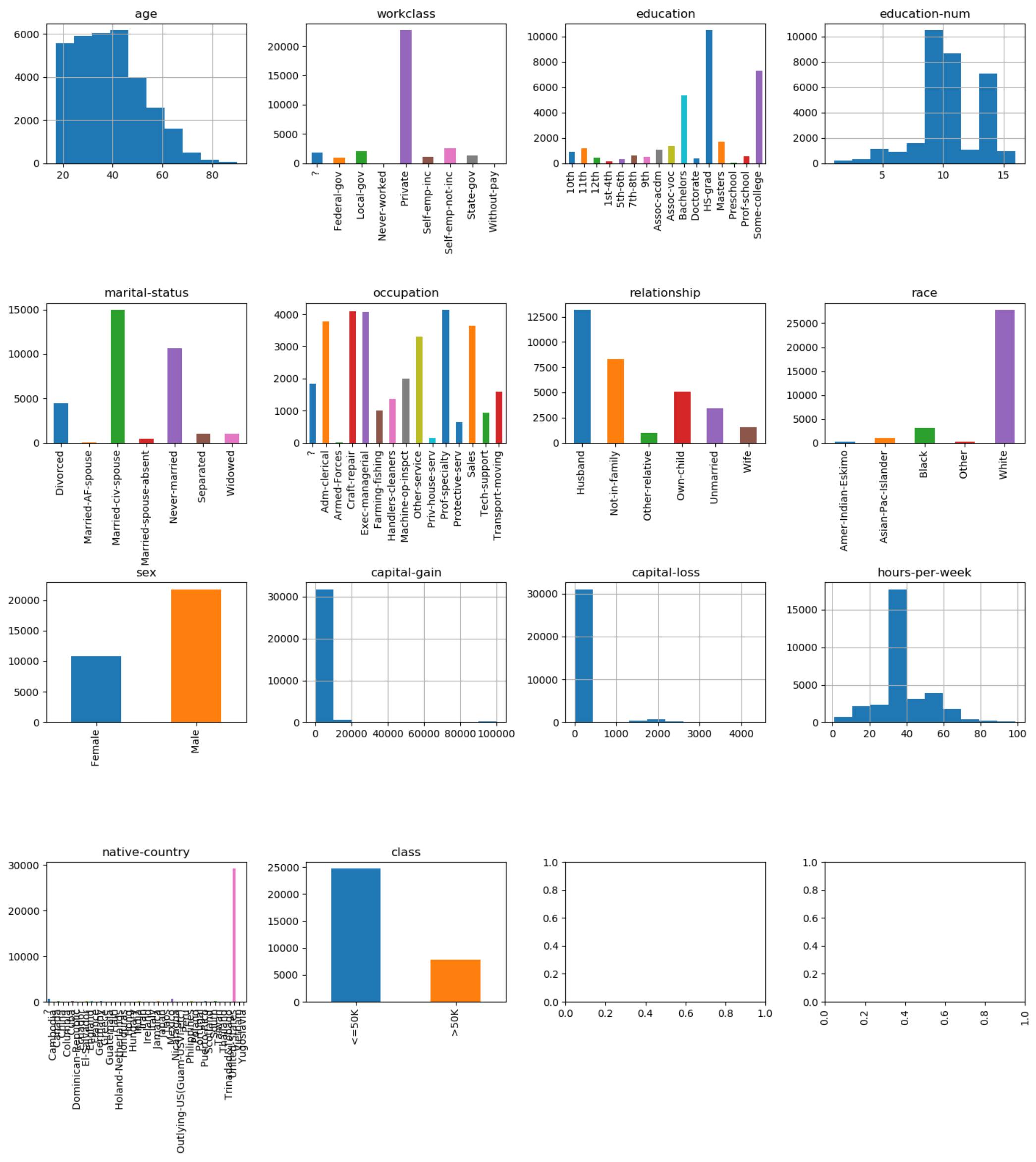


Figure 1 - Graphical feature representation

Each of these figures can also be exported as single images if the user wants to take a closer look. However, the overview already indicated a few things which are important in regards to machine learning:

1. Workclass, race, capital-gain, capital-loss, hours-per-week, native-country
   The charts for these features show that there are single values which represent a majority of data samples.
2. “?” Values
   Workclass, occupation and native-country have a category labeled as “?”. This means there is some missing data.
3. Class
   The feature class contains only two values. Therefore, the classification to be done in this project is a binary classification.
4. Categorical features
   Many of the features contain categorical values. These will have to get encoded. In this project, the Data will get Label Encoded and OneHot Encoded.

2.1.2 Linear correlations of the features

For a further overview on the data, a good option is to plot a correlation matrix. Since a lot of the features mentioned previously are categorical features, this correlation matrix will be generated with label encoded data and with OneHot encoded data. The correlation matrix itself is generated by the function built in pandas and afterwards handed over to seaborn in order to plot a heatmap:

```
print('Generating the Heatmap of %s encoded data' % (encoding))
fig = plt.figure(1,figsize=(20,20))
sns.heatmap(dataset.corr(), square=True, center=0, linewidth=0.5,
cmap='seismic')
plt.title('Heatmap of %s Encoded Data' % (encoding))
plt.savefig('Heatmap_%s.png' % (encoding))
print('Heatmap of %s encoded data done, please see file Heatmap_%s.png' %
(encoding,encoding))
```

Listing 3 - Correlation Plot

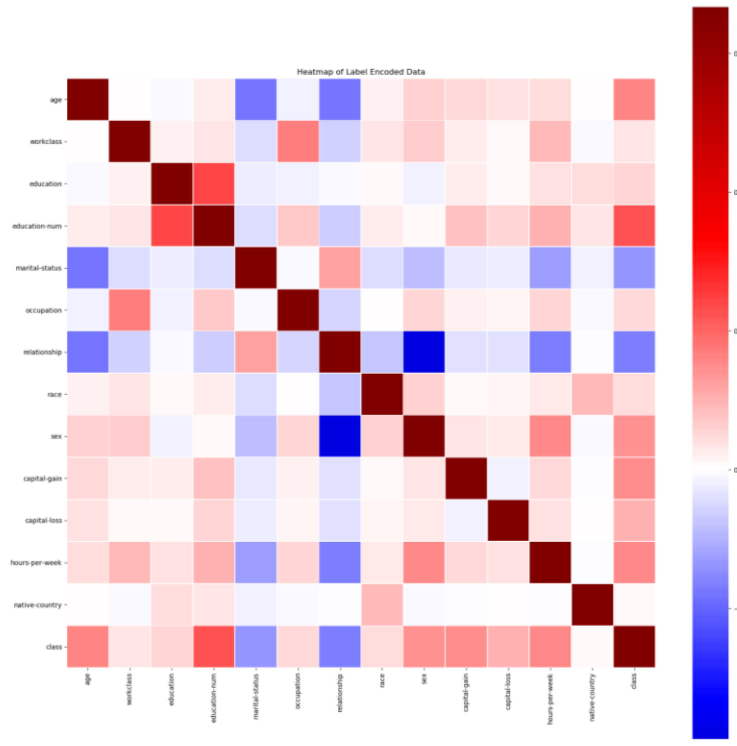


Figure 2 - Correlation heatmap - Label Encoded Data

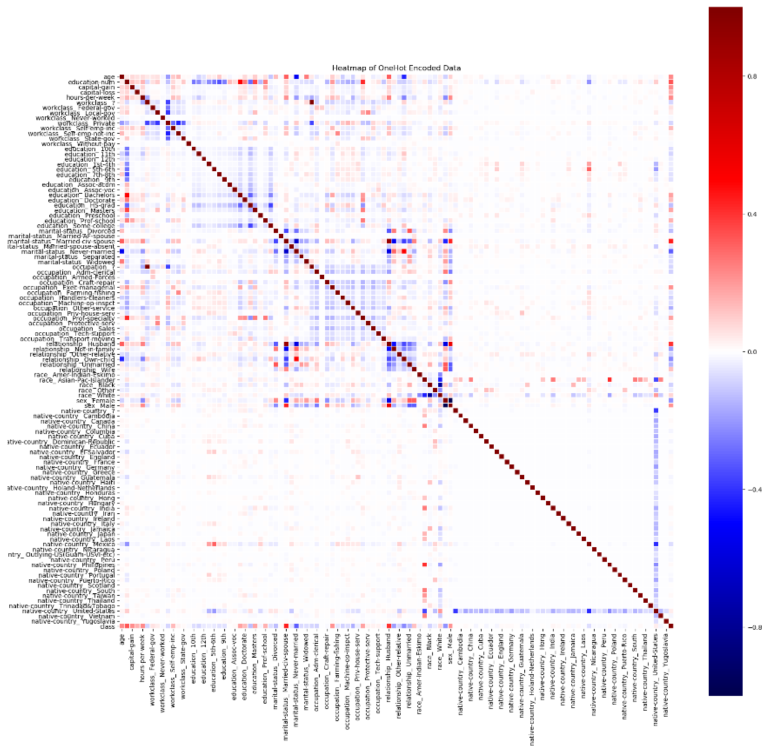


Figure 3 - Correlation Heatmap - OneHot Encoded Data

These correlation heatmaps give us the following information:

1. Relationship and sex have a high correlation
2. Education and education-num have a high correlation

This information will be relevant at a later stage of this project.

3 Machine Learning comparison

Since this project is a supervised learning project, there are several possibilities to check the applied algorithms. The scoring mechanisms used in this project are all implemented in sklearn.metrics:

Accuracy:

"Accuracy classification score.
In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true." (sklearn.metrics.accuracy_score — scikit-learn 0.19.1 documentation 2017)

F1:

"Compute the F1 score, also known as balanced F-score or F-measure

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

```
F1 = 2 * (precision * recall) / (precision + recall)
```

In the multi-class and multi-label case, this is the weighted average of the F1 score of each class." (sklearn.metrics.f1_score — scikit-learn 0.19.1 documentation 2017)

Roc_auc:

"Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.

Note: this implementation is restricted to the binary classification task or multilabel classification task in label indicator format." (sklearn.metrics.roc_auc_score — scikit-learn 0.19.1 documentation 2017)

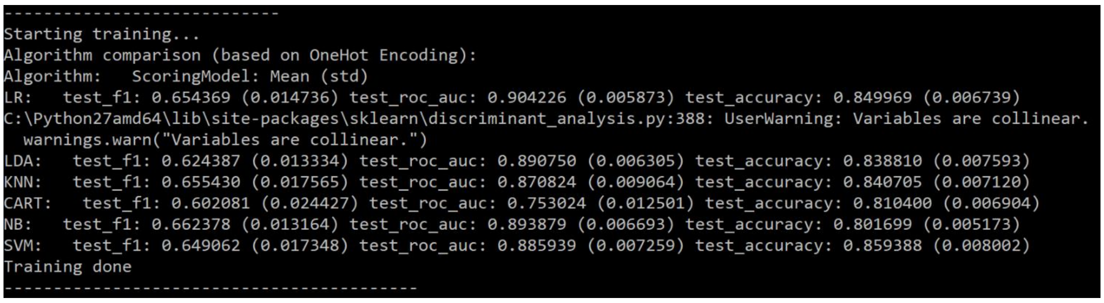
In this project, all above mentioned scores will be computed, but roc_auc will be the main score to judge the performance of the used algorithm.

3.1 Applying different algorithms

In ordert o get a first impression of different algorithms, these algorithms will be applied to the dataset with their correspondent default parameters. Method algorithmTrialDefault handles this and prints the corresponding results to the user.

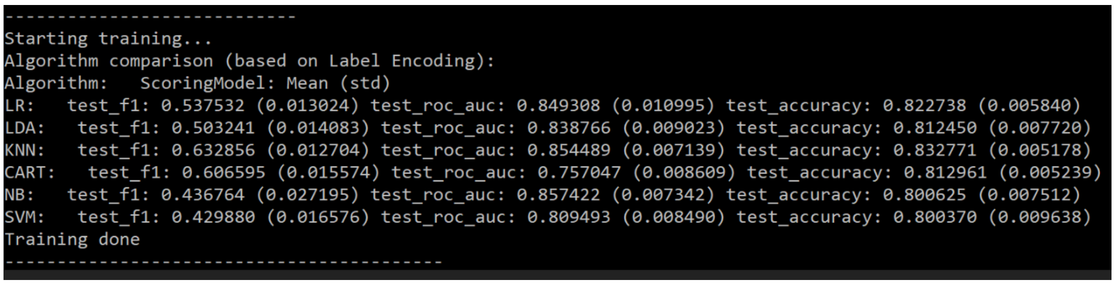
```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier(n_jobs=-1)))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC(max_iter=50)))
# evaluate each model in turn
results = []
names = []
print("----------------------------")
print("Starting training...")
print("Algorithm comparison (based on %s Encoding):" % (encoding))
print("Algorithm:   ScoringModel: Mean (std)")
for name, model in models:
    msg = "%s:  " % (name)
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    result = {'Algorithm': name}

    cv_results = model_selection.cross_validate(model, x_train, y_train,
cv=kfold, scoring=scoringMetric)
    for scoringResult in (scoringResult for scoringResult in cv_results if
scoringResult.find('test') != -1):
        msg = "%s %s: %f (%f)" %
(msg,scoringResult,cv_results[scoringResult].mean(),
cv_results[scoringResult].std() )
        result[scoringResult] = cv_results[scoringResult].mean()
        #print cv_results[scoringResult]
    cv_results['Algorithm'] = name
    results.append(result)
    names.append(name)
    #msg = "%s:  %f  (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

Listing 4 - Algorithm Trials



```
----------------------------
Starting training...
Algorithm comparison (based on OneHot Encoding):
Algorithm:   ScoringModel: Mean (std)
LR:   test_f1: 0.654369 (0.014736) test_roc_auc: 0.904226 (0.005873) test_accuracy: 0.849969 (0.006739)
C:\Python27amd64\lib\site-packages\sklearn\discriminant_analysis.py:388: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
LDA:   test_f1: 0.624387 (0.013334) test_roc_auc: 0.890750 (0.006305) test_accuracy: 0.838810 (0.007593)
KNN:   test_f1: 0.655430 (0.017565) test_roc_auc: 0.870824 (0.009064) test_accuracy: 0.840705 (0.007120)
CART:   test_f1: 0.602081 (0.024427) test_roc_auc: 0.753024 (0.012501) test_accuracy: 0.810400 (0.006904)
NB:   test_f1: 0.662378 (0.013164) test_roc_auc: 0.893879 (0.006693) test_accuracy: 0.801699 (0.005173)
SVM:   test_f1: 0.649062 (0.017348) test_roc_auc: 0.885939 (0.007259) test_accuracy: 0.859388 (0.008002)
Training done
------------------------------------------
```

Figure 4 - Algorithm Comparison - One Hot Encoded Data



```
----------------------------
Starting training...
Algorithm comparison (based on Label Encoding):
Algorithm:   ScoringModel: Mean (std)
LR:   test_f1: 0.537532 (0.013024) test_roc_auc: 0.849308 (0.010995) test_accuracy: 0.822738 (0.005840)
LDA:   test_f1: 0.503241 (0.014083) test_roc_auc: 0.838766 (0.009023) test_accuracy: 0.812450 (0.007720)
KNN:   test_f1: 0.632856 (0.012704) test_roc_auc: 0.854489 (0.007139) test_accuracy: 0.832771 (0.005178)
CART:   test_f1: 0.606595 (0.015574) test_roc_auc: 0.757047 (0.008609) test_accuracy: 0.812961 (0.005239)
NB:   test_f1: 0.436764 (0.027195) test_roc_auc: 0.857422 (0.007342) test_accuracy: 0.800625 (0.007512)
SVM:   test_f1: 0.429880 (0.016576) test_roc_auc: 0.809493 (0.008490) test_accuracy: 0.800370 (0.009638)
Training done
------------------------------------------
```

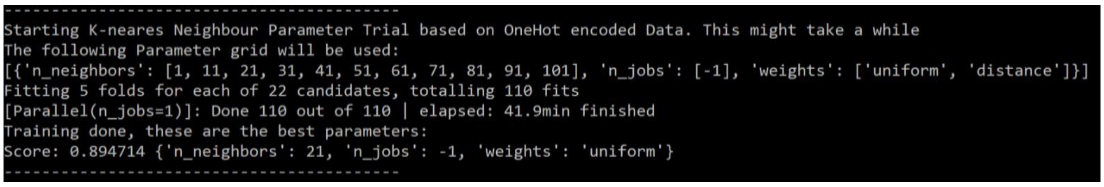
Figure 5 - Algorithm Comparison - Label Encoded Data

Based on these Results, it seems that Logistic Regression provides the best results for the given dataset (0.904 for One Hot Encoded Data). However, the K-Nearest Neighbor algorithm and Support Vector Machines are known for being highly dependent on their parameters. Therefore, these three algorithms will be checked further. For this, the `GridSearchCV` function out of sklearn.model_selection will be used. This function accepts an algorithm, a scoring mechanism and a parameter grid. It will try all combinations of the given parameters and return the parameter combination with the best results.

3.1.1 K-Nearest Neighbor

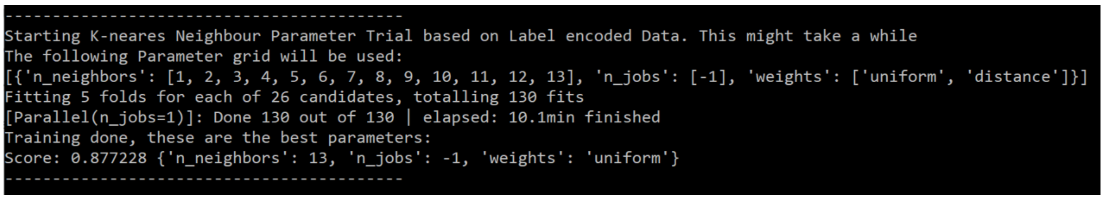
The main parameters for the K-Nearest Neighbor algorithm are k as the number of neighbors to be taken into account as well as the weighting algorithm. The following parameter grid will be used:

```
parametersgrid = [{'n_neighbors': range(1,len(dataset.columns),
(len(dataset.columns) / 10) ), 'weights': ['uniform', 'distance'], 'n_jobs':
[-1]}]
```



```
----------------------------------------
Starting K-neares Neighbour Parameter Trial based on OneHot encoded Data. This might take a while
The following Parameter grid will be used:
[{'n_neighbors': [1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101], 'n_jobs': [-1], 'weights': ['uniform', 'distance']}]
Fitting 5 folds for each of 22 candidates, totalling 110 fits
[Parallel(n_jobs=1)]: Done 110 out of 110 | elapsed: 41.9min finished
Training done, these are the best parameters:
Score: 0.894714 {'n_neighbors': 21, 'n_jobs': -1, 'weights': 'uniform'}
----------------------------------------
```

Figure 6 - K-NN Parameter Search Results - One Hot Encoded Data



```
----------------------------------------
Starting K-neares Neighbour Parameter Trial based on Label encoded Data. This might take a while
The following Parameter grid will be used:
[{'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13], 'n_jobs': [-1], 'weights': ['uniform', 'distance']}]
Fitting 5 folds for each of 26 candidates, totalling 130 fits
[Parallel(n_jobs=1)]: Done 130 out of 130 | elapsed: 10.1min finished
Training done, these are the best parameters:
Score: 0.877228 {'n_neighbors': 13, 'n_jobs': -1, 'weights': 'uniform'}
----------------------------------------
```

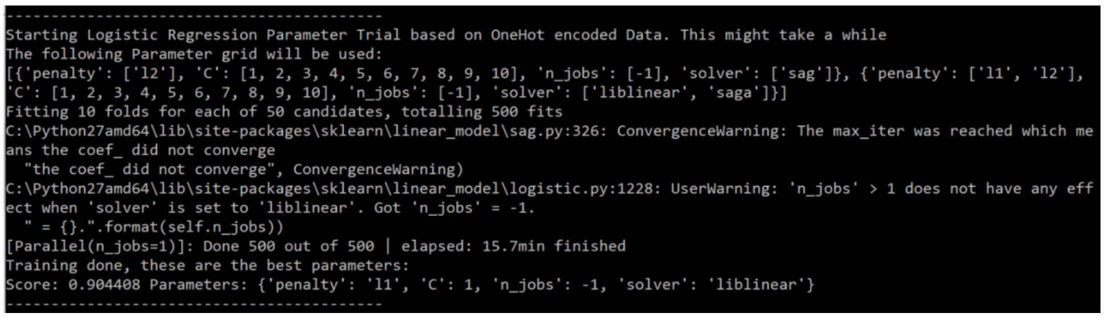
Figure 7 - K-NN Parameter Search Results - Label Encoded Data

This got the scoring results of K-Nearest-Neighbor up to 0.89 with K as 21, weighting to be uniform and the data being OneHot Encoded.

3.1.2 LogisticRegression

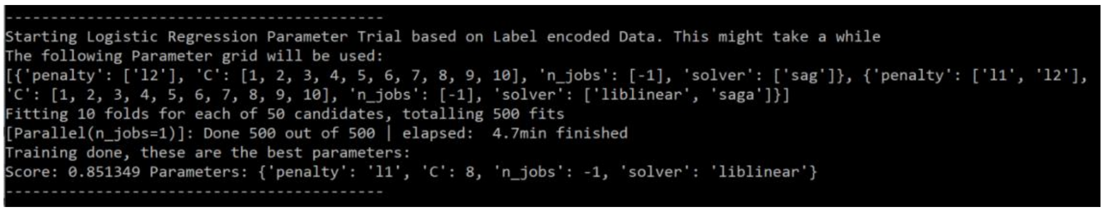
For Logistic regression, the values to be tested are C as the inverse of the regularization strength, penalty as the penalty mechanism to be applied as well as solver as the solving algorithm. With this, the following parameter grid will be used:

```
parametersgrid = ['C': range(1,11), 'penalty': ['l2'], 'solver': ['sag'],
'n_jobs': [-1]},
{'C': range(1,11), 'penalty': ['l1','l2'], 'solver': ['liblinear', 'saga'],
'n_jobs': [-1]}
```



```
----------------------------------------
Starting Logistic Regression Parameter Trial based on OneHot encoded Data. This might take a while
The following Parameter grid will be used:
[{'penalty': ['l2'], 'C': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'n_jobs': [-1], 'solver': ['sag']}, {'penalty': ['l1', 'l2'],
'C': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'n_jobs': [-1], 'solver': ['liblinear', 'saga']}]
Fitting 10 folds for each of 50 candidates, totalling 500 fits
C:\Python27amd64\lib\site-packages\sklearn\linear_model\sag.py:326: ConvergenceWarning: The max_iter was reached which me
ans the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Python27amd64\lib\site-packages\sklearn\linear_model\logistic.py:1228: UserWarning: 'n_jobs' > 1 does not have any eff
ect when 'solver' is set to 'liblinear'. Got 'n_jobs' = -1.
  " = {}.".format(self.n_jobs))
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed: 15.7min finished
Training done, these are the best parameters:
Score: 0.904408 Parameters: {'penalty': 'l1', 'C': 1, 'n_jobs': -1, 'solver': 'liblinear'}
----------------------------------------
```

Figure 8 - Logistic Regression Parameter Search Results - One Hot Encoded Data



```
----------------------------------------
Starting Logistic Regression Parameter Trial based on Label encoded Data. This might take a while
The following Parameter grid will be used:
[{'penalty': ['l2'], 'C': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'n_jobs': [-1], 'solver': ['sag']}, {'penalty': ['l1', 'l2'],
'C': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'n_jobs': [-1], 'solver': ['liblinear', 'saga']}]
Fitting 10 folds for each of 50 candidates, totalling 500 fits
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:  4.7min finished
Training done, these are the best parameters:
Score: 0.851349 Parameters: {'penalty': 'l1', 'C': 8, 'n_jobs': -1, 'solver': 'liblinear'}
----------------------------------------
```

Figure 9 - Logistic Regression Parameter Search Results - Label Encoded Data

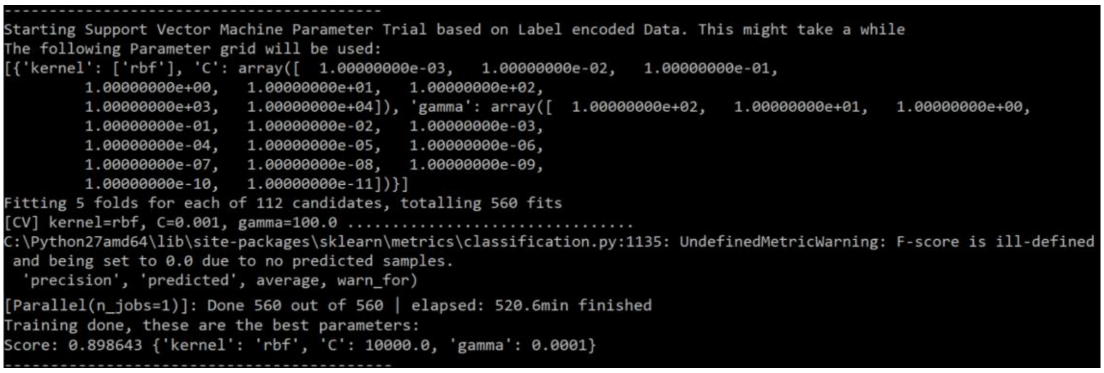
With this parameter grid, the best result is 0.904 which gets achieved by the liblinear solver, C as 1 and the l1 penalty mechanism.

3.1.3 Support Vector Machines

Support Vector Machines rely on the used kernel. Different trials during this project always returned the best results when using the rbf kernel. Since the parameter trial of Support Vector Machines is runtime intensive, only the rbf kernel will be taken into account during the final try. Support Vector machines rely heavily on c as the penalty parameter. The rbf Kernel relies on gamma as its coefficient. The following parameter grid will be used:

```
parametersgrid = [{'kernel': ['rbf'], 'gamma': np.logspace(2,-11, base=10,
num=2+11+1) ,'C': np.logspace(-3,4, base=10, num=3+4+1), 'max_iter': [-1]}]
```

The performance of the SVM parameter search drops significantly if one hot encoded data is used. Therefore, only label encoded data will be used.



```
----------------------------------------
Starting Support Vector Machine Parameter Trial based on Label encoded Data. This might take a while
The following Parameter grid will be used:
[{'kernel': ['rbf'], 'C': array([  1.00000000e-03,   1.00000000e-02,   1.00000000e-01,
         1.00000000e+00,   1.00000000e+01,   1.00000000e+02,
         1.00000000e+03,   1.00000000e+04]), 'gamma': array([  1.00000000e+02,   1.00000000e+01,   1.00000000e+00,
         1.00000000e-01,   1.00000000e-02,   1.00000000e-03,
         1.00000000e-04,   1.00000000e-05,   1.00000000e-06,
         1.00000000e-07,   1.00000000e-08,   1.00000000e-09,
         1.00000000e-10,   1.00000000e-11])}]
Fitting 5 folds for each of 112 candidates, totalling 560 fits
[CV] kernel=rbf, C=0.001, gamma=100.0 ...............................
C:\Python27amd64\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: F-score is ill-defined
 and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
[Parallel(n_jobs=1)]: Done 560 out of 560 | elapsed: 520.6min finished
Training done, these are the best parameters:
Score: 0.898643 {'kernel': 'rbf', 'C': 10000.0, 'gamma': 0.0001}
----------------------------------------
```

Figure 10 - SVM Parameter Search Results - Label Encoded Data

Even when using label encoded data, the result of support vector machines went up to a score of 0.89, when using the rbf kernel, C as 10000 and gamma as 0.0001.

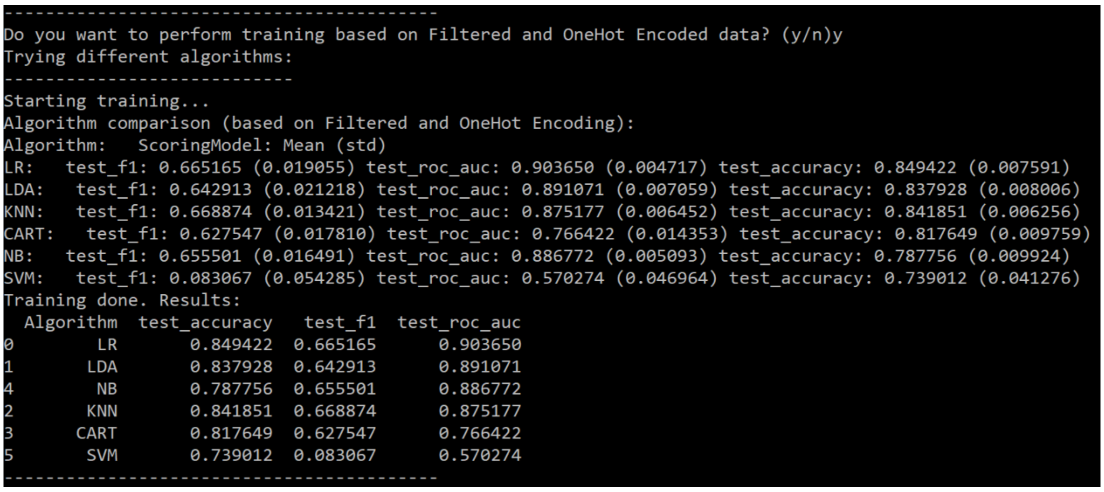
3.2 Tweaking

In order to tweak the results, the initial dataset will be manipulated a bit based on the results of chapter 2:

- The feature "education" will be deleted, since it is already reflected in "education-num"
- Missing values in classes workclass, occupation and native-country will be deleted

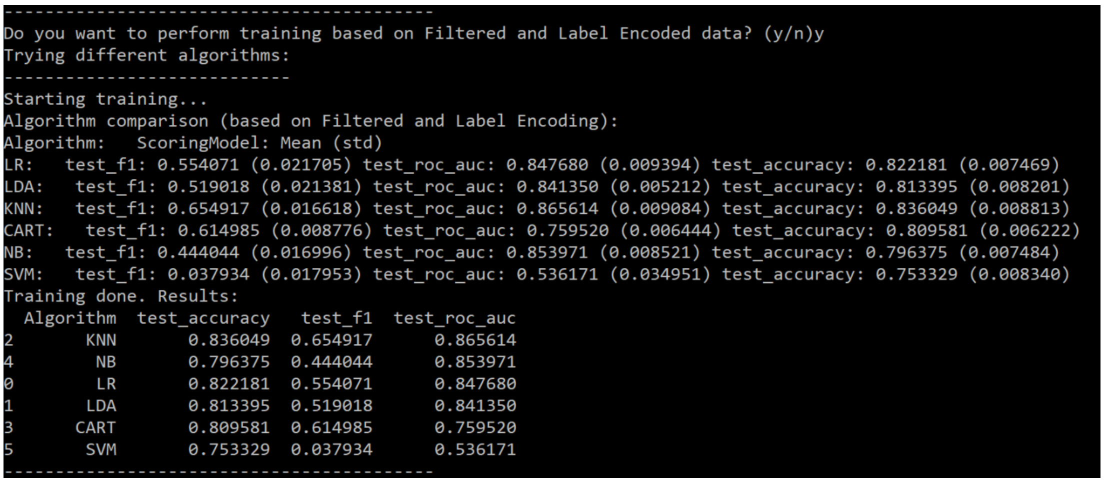
```
#We should get rid of any unknown values
dataset_flt = dataset[dataset["workclass"] != "?"]
dataset_flt = dataset_flt[dataset_flt["occupation"] != "?"]
dataset_flt = dataset_flt[dataset_flt["native-country"] != "?"]
del dataset_flt['education']
```

Listing 5 - Dataset manipulation



```
-----------------------------------------
Do you want to perform training based on Filtered and OneHot Encoded data? (y/n)y
Trying different algorithms:
---------------------------
Starting training...
Algorithm comparison (based on Filtered and OneHot Encoding):
Algorithm:   ScoringModel: Mean (std)
LR:   test_f1: 0.665165 (0.019055) test_roc_auc: 0.903650 (0.004717) test_accuracy: 0.849422 (0.007591)
LDA:   test_f1: 0.642913 (0.021218) test_roc_auc: 0.891071 (0.007059) test_accuracy: 0.837928 (0.008006)
KNN:   test_f1: 0.668874 (0.013421) test_roc_auc: 0.875177 (0.006452) test_accuracy: 0.841851 (0.006256)
CART:   test_f1: 0.627547 (0.017810) test_roc_auc: 0.766422 (0.014353) test_accuracy: 0.817649 (0.009759)
NB:   test_f1: 0.655501 (0.016491) test_roc_auc: 0.886772 (0.005093) test_accuracy: 0.787756 (0.009924)
SVM:   test_f1: 0.083067 (0.054285) test_roc_auc: 0.570274 (0.046964) test_accuracy: 0.739012 (0.041276)
Training done. Results:
  Algorithm  test_accuracy   test_f1  test_roc_auc
0        LR       0.849422  0.665165      0.903650
1       LDA       0.837928  0.642913      0.891071
4        NB       0.787756  0.655501      0.886772
2       KNN       0.841851  0.668874      0.875177
3      CART       0.817649  0.627547      0.766422
5       SVM       0.739012  0.083067      0.570274
-----------------------------------------
```

Figure 11 - Algorithm Comparison - Filtered and One Hot Encoded Data



```
------------------------------------------
Do you want to perform training based on Filtered and Label Encoded data? (y/n)y
Trying different algorithms:
----------------------------
Starting training...
Algorithm comparison (based on Filtered and Label Encoding):
Algorithm:   ScoringModel: Mean (std)
LR:   test_f1: 0.554071 (0.021705) test_roc_auc: 0.847680 (0.009394) test_accuracy: 0.822181 (0.007469)
LDA:   test_f1: 0.519018 (0.021381) test_roc_auc: 0.841350 (0.005212) test_accuracy: 0.813395 (0.008201)
KNN:   test_f1: 0.654917 (0.016618) test_roc_auc: 0.865614 (0.009084) test_accuracy: 0.836049 (0.008813)
CART:   test_f1: 0.614985 (0.008776) test_roc_auc: 0.759520 (0.006444) test_accuracy: 0.809581 (0.006222)
NB:   test_f1: 0.444044 (0.016996) test_roc_auc: 0.853971 (0.008521) test_accuracy: 0.796375 (0.007484)
SVM:   test_f1: 0.037934 (0.017953) test_roc_auc: 0.536171 (0.034951) test_accuracy: 0.753329 (0.008340)
Training done. Results:
  Algorithm  test_accuracy   test_f1  test_roc_auc
2       KNN       0.836049  0.654917      0.865614
4        NB       0.796375  0.444044      0.853971
0        LR       0.822181  0.554071      0.847680
1       LDA       0.813395  0.519018      0.841350
3      CART       0.809581  0.614985      0.759520
5       SVM       0.753329  0.037934      0.536171
------------------------------------------
```

Figure 12 - Algorithm Comparison - Filtered and Label Encoded Data

4 Conclusion

During the development of this script, I learned a lot about the different machine learning algorithms. Given the dataset selected, the main improvements in regards to the scoring of the different algorithms was achieved by changing the parameters with which the algorithms were run. This was much more significant than any manual adjustments done to the dataset itself. However, for another dataset, this might change significantly.

Index of Figures

Index of Listings

Bibliography

Buitinck, Lars; Louppe, Gilles; Blondel, Mathieu; Pedregosa, Fabian; Mueller, Andreas; Grisel, Olivier et al. (2013): API design for machine learning software. Experiences from the scikit-learn project. In : ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122.

Hunter, John D. (2007): Matplotlib. A 2D Graphics Environment. In *Comput. Sci. Eng.* 9 (3), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

Jones, Eric; Oliphant, Travis; Peterson, Pearu; others (2001-): SciPy. Open source scientific tools for Python. Available online at http://www.scipy.org/.

Lichman, M. (2013): UCI Machine Learning Repository. Available online at http://archive.ics.uci.edu/ml.

McKinney, Wes (2010): Data Structures for Statistical Computing in Python. In Stéfan van der Walt, Jarrod Millman (Eds.): Proceedings of the 9th Python in Science Conference, pp. 51–56.

Michael Waskom; Olga Botvinnik@czbiohub; Drew O'KaneThe Climate Corporation; Paul Hobson@Geosyntec; Saulius Lukauskas; David C Gemperline et al. (2017): Mwaskom/Seaborn. V0.8.1 (September 2017): Zenodo.

sklearn.metrics.accuracy_score — scikit-learn 0.19.1 documentation (2017). Available online at http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score, updated on 12/22/2017, checked on 12/24/2017.

sklearn.metrics.f1_score — scikit-learn 0.19.1 documentation (2017). Available online at http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score, updated on 12/22/2017, checked on 12/24/2017.

sklearn.metrics.roc_auc_score — scikit-learn 0.19.1 documentation (2017). Available online at http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score, updated on 12/22/2017, checked on 12/24/2017.

UCI Machine Learning Repository: Adult Data Set. Available online at http://archive.ics.uci.edu/ml/datasets/Adult, checked on 12/22/2017.

van der Walt, Stéfan; Colbert, S. Chris; Varoquaux, Gaël (2011): The NumPy Array. A Structure for Efficient Numerical Computation. In *Comput. Sci. Eng.* 13 (2), pp. 22–30. DOI: 10.1109/MCSE.2011.37.