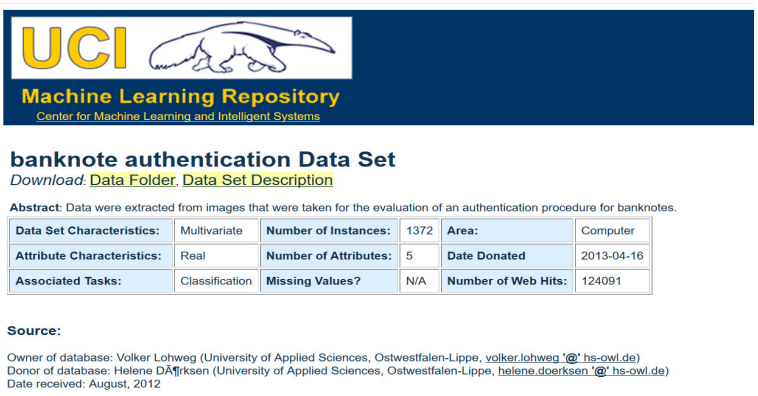

Banknote Authentication Data (Classification by Python)

Hossein Zareamoghaddam

Nov 19, 2017

1 Data Set Information

For the purpose of data classification using software programming Python, the data set called 'banknote authentication' is selected which is available at The UCI Machine Learning Repository website (http://archive.ics.uci.edu/ml/datasets/banknote+authentication). Data were extracted from images that were taken from genuine and forged banknote-like specimens. For digitization, an industrial camera usually used for print inspection was used. The final images have 400×400 pixels. Due to the object lens and distance to the investigated object gray-scale pictures with a resolution of about 660 dpi were gained. Wavelet Transform tool were used to extract features from images. The following are the attribute information of the data set:

- *variance:* variance of Wavelet Transformed image (continuous)
- *skewness:* skewness of Wavelet Transformed image (continuous)
- *curtosis:* kurtosis of Wavelet Transformed image (continuous)
- *entropy:* entropy of image (continuous)
- *class:* authenticity of the banknote of two levels
  - *1:* real banknote

– *0:* fake banknote

'pandas' and 'numpy' are the primary Python libraries packages required for importing data sets and preprocessing of the data.

```
import pandas as pd
import numpy as np

# Importing the dataset
data = pd.read_csv('banknote.csv')
```

To get more information about the features of the data set, one may use 'data.shape' or 'data.head()' commands. So, we can see that there are 1372 observations of 5 variables with four features and one classified, with two levels, where the first 5 observations of the data are shown below by making use of 'data.head()' command

```
   variance  skewness  curtosis  entropy   class
0   3.62160    8.6661   -2.8073  -0.44699      0
1   4.54590    8.1674   -2.4586  -1.46210      0
2   3.86600   -2.6383    1.9242   0.10645      0
3   3.45660    9.5228   -4.0112  -3.59440      0
4   0.32924   -4.4552    4.5718  -0.98880      0
```

and the summary of those features, by making use of 'data.describe()' command, is

```
          variance     skewness     ...          entropy         class
count  1372.000000  1372.000000     ...       1372.000000  1372.000000
mean      0.433735     1.922353     ...         -1.191657     0.444606
std       2.842763     5.869047     ...          2.101013     0.497103
min      -7.042100   -13.773100     ...         -8.548200     0.000000
25%      -1.773000    -1.708200     ...         -2.413450     0.000000
50%       0.496180     2.319650     ...         -0.586650     0.000000
75%       2.821475     6.814625     ...          0.394810     1.000000
max       6.824800    12.951600     ...          2.449500     1.000000
```

2 Data Processing and Visualization

One may be curious to see the pairwise relationship between features and the histogram plots of different variables to have better understanding of the relationship between variables. The plot at the left panel of Figure 1 shows the scatter plots and histogram of different levels of the classified variable 'class' and the right panel illustrates the corresponding pairwise correlations extracted from the following scripts:

```
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')
```

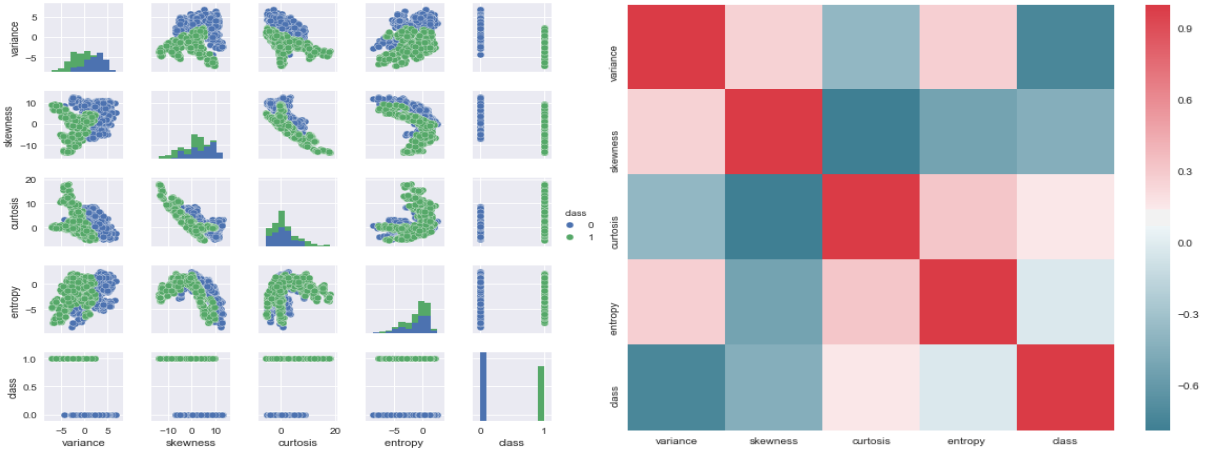

Figure 1: Different histograms and scatter plots of bankenote data set

```
import seaborn as sns
sns.set()
sns_plot = sns.pairplot(data, hue='class', size=1.5)

corr = data.corr()
plt.subplots(figsize=(10,8))
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True), square=True)
```

The data set with 1372 observations of five variables including the categorical variable *class* assigned as y and the first four continuous variables are considered as independent vector $\mathbf{x} = (x_1, x_2, x_3, x_4)$. Therefore, the data set is divided into a vector y and a matrix X. Here, some classification algorithms are studied. In order to evaluate the accuracy of each algorithm, the data set is divided into two randomly assigned *train* and *test* data sets with 1029 and 343 observations, respectively. The relevant Python command codes are here

```
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Selecting test and training sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,
                                                    random_state = 50)
```

Some of the classification techniques depend on the distance between points (features). In some situations, there are significant gaps between features where the difference between one feature may dominate the difference between another feature and therefore the classification technique will not be efficient for features with small values in presence of features with large values. To avoid such problems and to get more accurate results, the *train* and *test* sets are usually normalized as follows:

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

According to the summary of the data sets and the knowledge of having no missing value in the data, one may fit a classification techniques of interest on the training set and evaluate its performance by making use of this model to predict the levels of y values in the test set.

3 Classification

In this section, some classification techniques are employed to compare their performances for predicting the levels of y in test set for 'banknote' data. Using the fitted model computed by the training set through each algorithm, the predicted values of y in the test set are compared to the true values. The number of correct predictions out of 343, which is the test set sample size, are accounted as a measure of accuracy of the corresponding algorithm.

3.1 Logistic Regression Model

Logistic regression is a popular classification approach where y is a categorical variable with just two levels which is the case in 'banknote' data set. The logistic regression model is considered as

$$g(p(x)) = Log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1\, x_1 + \beta_2\, x_2 + \beta_3\, x_3 + \beta_4\, x_4,$$

where

$$p(x) = \frac{e^{\beta_0+\beta_1\, x_1+\beta_2\, x_2+\beta_3\, x_3+\beta_4\, x_4}}{1+e^{\beta_0+\beta_1\, x_1+\beta_2\, x_2+\beta_3\, x_3+\beta_4\, x_4}}.$$

For classification purposes, one may set $y = 1$ when $p \geq 0.5$ and $y = 0$ when $p < 0.5$ and unknown parameters β_i, $i = 0, \ldots, 4$ are estimated from the training set using the following code:

```
# Fitting a model to the Training set
from sklearn.linear_model import LogisticRegression
classifier_logistic = LogisticRegression()
classifier_logistic.fit(X_train, y_train)

# Predicting the Test set
y_pred_logis = classifier_logistic.predict(X_test)
```

In Python, for evaluating a classification algorithm, confusion matrix, precision, recall and f1 score are the most commonly used metrics. Here, the 'confusion-matrix' and 'classification-report' methods of the *sklearn.metrics* are used to calculate these metrics using the following code:

```
# Evaluating the Algorithm by classification_report and confusion Matrix
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred_logis))
print(classification_report(y_test, y_pred_logis))
```

and the logistic regression confusion-matrix and classification-report results are

```
[[193   5]
[  0 145]]
     precision    recall  f1-score   support

0       1.00      0.97      0.99       198
1       0.97      1.00      0.98       145

avg / total       0.99      0.99      0.99       343
```

It means that using logistic regression classification, we made 5 wrong predictions out of 343 which is a good results showing 99 percent correct prediction.

3.2 K-Nearest Neighbor (KNN)

The K-nearest neighbors algorithm is a type of supervised machine learning algorithms where it is easy to implement in its basic format, and yields good performances by applying this algorithm on some complex classification tasks. KNN is a non-parametric algorithm, using the some nearest observations of the points, in the training set, for prediction. This is an important advantage of this algorithm since most of the real world data doesn't really follow any particular assumption e.g. linear-separability, uniform distribution, etc. Here, we will see how KNN can be implemented with Python's Scikit-Learn library.

```
# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier_knn.fit(X_train, y_train)

# Predicting the Test set results
y_pred_knn = classifier_knn.predict(X_test)

# Evaluating KNN ALgorithm
print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

The KNN confusion matrix and classification report are shown below where just one wrong prediction decision was made using this algorithm. Having a simple algorithm with such a good performance for classification purposes, KNN algorithm have been widely used to find document similarity and pattern recognition. It has also been employed for developing recommender systems and for dimensionality reduction and pre-processing steps for computer vision, particularly face recognition tasks.

```
[[197   1]
[  0 145]]
     precision    recall  f1-score   support

0       1.00      0.99      1.00       198
1       0.99      1.00      1.00       145

avg / total       1.00      1.00      1.00       343
```

3.3 Support Vector Machines (SVM)

Support vector machine, as a machine learning algorithm, is a supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier using hyperplanes to divide the data into different categories. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New observations are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using kernel mapping trick, implicitly mapping the inputs into a higher dimensional space. In this section, the linear and non-linear SVM approaches are employed to classify the test set.

The following script is for linear SVM classification in Python:

```
# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier_svm = SVC(kernel = 'linear', random_state = 0)
classifier_svm.fit(X_train, y_train)

# Predicting the Test set classifications
y_pred_svm = classifier_svm.predict(X_test)

# Evaluating SVM ALgorithm
print(confusion_matrix(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

Based on the following evaluating metrics, using linear SVM classifier, just four wrong predicted decisions made which shows higher performance rather than the logistic regression.

```
[[194   4]
[  0 145]]
     precision    recall  f1-score   support
```

```
0       1.00      0.98      0.99       198
1       0.97      1.00      0.99       145

avg / total       0.99      0.99      0.99       343
```

Separation of some data using a hyperplanes is not always a right choice because the pattern of the points is perhaps nonlinear. SVM algorithm using a technique called kernel trick is an efficient approach where, first, the kernel function takes low dimensional input space and transform it to a higher dimensional space and then find out the process to separate the data based on the labels or outputs you've defined. In Python, there are various options available with kernel like, 'linear', 'rbf, 'poly' and others (default value is 'rbf') where 'rbf' and 'poly' are well-known nonlinear transformations using Gaussian and polynomial functions respectively. Here, 'rbf' is selected as the kernel function for kernel SVM algorithm.

```
classifier_svm2 = SVC(kernel = 'rbf')
classifier_svm2.fit(X_train, y_train)

# Predicting the Test set classifications
y_pred_svm2 = classifier_svm2.predict(X_test)

# Evaluating SVM ALgorithm
print(confusion_matrix(y_test, y_pred_svm2))
print(classification_report(y_test, y_pred_svm2))
```

It is a great achievement that kernel SVM made no wrong prediction decision in this experiment, proving the high level of efficiency of this technique.

```
[[198   0]
[  0 145]]
     precision    recall  f1-score   support

0       1.00      1.00      1.00       198
1       1.00      1.00      1.00       145

avg / total       1.00      1.00      1.00       343
```

3.4 Naive Bayes Method

The Naive Bayes algorithm is an intuitive method that uses the Bayes Theorem and probabilities of each attribute belonging to each class to make a prediction. It is a supervised machine learning approach for prediction aims. Naive Bayes simplifies the calculation of probabilities by assuming that the probability of each attribute belonging to a given class value is independent of all other attributes. This is a strong assumption but results in a fast and effective method. The following script is for naive Bayes classification algorithm:

```
from sklearn.naive_bayes import GaussianNB
classifier_nb = GaussianNB()
classifier_nb.fit(X_train, y_train)

# Predicting the Test set classifications
y_pred_nb = classifier_nb.predict(X_test)

# Evaluating the ALgorithm
print(confusion_matrix(y_test, y_pred_nb))
print(classification_report(y_test, y_pred_nb))
```

Comparing to other classification methods, Naive Bayes technique is not efficient for this data with making 55 wrong decisions.

```
[[174  24]
[ 31 114]]
     precision    recall  f1-score   support

0       0.85      0.88      0.86       198
1       0.83      0.79      0.81       145

avg / total       0.84      0.84      0.84       343
```

3.5 Random Forest Classification

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees where decision tree concept is more to the rule based system. Ensemble classifier means a group of classifiers. Instead of using only one classifier to predict the target, in ensemble, we use multiple classifiers to predict the target. Given the training dataset with targets and features, the decision tree algorithm will come up with some set of rules. The same set rules can be used to perform the prediction on the test data set. In this algorithm, the process of finding the root nodes and splitting the feature nodes for each tree is happened randomly. More information regarding this algorithm is available in machine learning websites.

```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifierRF = RandomForestClassifier(n_estimators = 10)
classifierRF.fit(X_train, y_train)

# Predicting the Test set results
y_pred_rf = classifierRF.predict(X_test)

# Evaluating the ALgorithm
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

Here, the prediction using random forest algorithm with 10 trees ends up to the results of making just two wrong decisions out of 343 predictions.

```
[[197   1]
[  1 144]]
     precision    recall  f1-score   support

0       0.99      0.99      0.99       198
1       0.99      0.99      0.99       145

avg / total       0.99      0.99      0.99       343
```

4 Conclusion

Interested reader might refer to some the machine learning books, websites, etc. for further information about the classification algorithms and their pros and cons. However, based on the results of this experiment, one may choose kernel SVM as the best selected algorithm for classification following by KNN and random forest classification techniques and logistic regression had an acceptable performance as well, where Naive Bayes method might be considered as a simple and intuitive approach to get an initial thought about the task.

References

- Aurelien Geron, Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent, 2017.
- https://en.wikipedia.org/wiki/Statistical-classification
- http://archive.ics.uci.edu/ml/datasets/banknote+authentication