APPLIED MACHINE LEARNING
LAB ACTIVITIES (LAB 3) (WITH SOLUTIONS)

17/10/19

FEATURE SELECTION AND RESAMPLING

This workbook is designed to guide you through the activities proposed for today's lab. As you will be working independently, feel free to proceed through the text at your own pace, spending more time on the parts that are less familiar to you. The workbook contains both hands-on tasks and links to learning materials such as tutorials, articles and videos. When you are unsure about something, feel free to ask our teaching assistants or use Internet resources to look for a solution. At the end of each section, there will be questions and exercises to verify your understanding of the presented information. You may need to do some research to answer the questions.

1. Feature Selection

The data features that you use to train your machine learning models have a huge influence on the performance you can achieve. Irrelevant or partially relevant features can negatively impact model performance. Feature selection is a process where you automatically select those features in your data that contribute most to the prediction variable or output in which you are interested. Having irrelevant features in your data can decrease the accuracy of many models, especially linear algorithms like linear and logistic regression. Three benefits of performing feature selection before modelling your data are:

- Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.
- Improves Accuracy: Less misleading data means modelling accuracy improves.
- Reduces Training Time: Less data means that algorithms train faster.

You can learn more about feature selection with scikit-learn in the article Feature selection. See below.

https://scikit-learn.org/stable/modules/feature_selection.html

Each feature selection recipes will use the Pima Indians onset of diabetes dataset.

Let us begin with **Univariate Selection.**

Statistical tests can be used to select those features that have the strongest relationship with the output variable. The scikit-learn library provides the $SelectKBest$ class that can be used with a suite of different statistical tests to select a specific number of features. Details of the $SelectKBest$ class is available at

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.SelectKBest

The code block below uses the chi-squared ($chi2$) statistical test for non-negative features to select *4 of the best features* from the Pima Indians onset of diabetes dataset.

```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for classification)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

```
# load data
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)
# summarize scores
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```

Verify your understanding:

(a) *Analyse the above code block and run it. Name the best 4 attributes (those with the highest scores).*

Soln.

You can see the scores for each attribute and the 4 attributes chosen (those with the highest scores): plas, test, mass and age. I got the names for the chosen attributes by manually mapping the index of the 4 highest scores to the index of the attribute names.

```
[ 111.52  1411.887   17.605   53.108 2175.565  127.669    5.393
  181.304]
[[ 148.    0.   33.6  50. ]
 [  85.    0.   26.6  31. ]
 [ 183.    0.   23.3  32. ]
 [  89.   94.   28.1  21. ]
 [ 137.  168.   43.1  33. ]]
```

Output of univariate feature selection.

Recursive Feature Elimination

The Recursive Feature Elemination (RFE) works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute. You can learn more about the *RFE* class in the scikit-learn documentation. See below.

https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html#sklearn.feature_selection.RFE

The code block below uses RFE with the logistic regression algorithm to select *the top 3 features*. The choice of algorithm does not matter too much as long as it is skillful and consistent.

```
# Feature Extraction with RFE
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = LogisticRegression()
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
```

```
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

Verify your understanding:

(b) *Analyse the above code block and run it. What are the top three features suggested by the* $support\_array$*?*

(c) *What are the features marked with the first choice in the* $ranking\_array$*?*

Soln

You can see that RFE chose the top 3 features as preg, mass and pedi. These are marked True in the support array and marked with a choice 1 in the ranking array. Again, you can manually map the feature indexes to the indexes of attribute names.

```
Num Features: 3
Selected Features: [ True False False False False True True False]
Feature Ranking: [1 2 3 5 6 1 1 4]
```

Output of RFE feature selection.

Data Reduction using Principal Component Analysis

Principal Component Analysis (or PCA) uses linear algebra to transform the dataset into a compressed form. Generally this is called a data reduction technique. A property of PCA is that you can choose the number of dimensions or principal components in the transformed result. In the example below, we use PCA and select 3 principal components. Learn more about the *PCA* class in scikit-learn by reviewing the API.

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

```
# Feature Extraction with PCA
from pandas import read_csv
from sklearn.decomposition import PCA
# load data
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

Verify your understanding:

(d) *Analyse the above code block and run it. Name 3 real-life applications that PCA would be useful and discuss why.*

Feature Importance

Bagged decision trees like Random Forest and Extra Trees can be used to estimate the importance of features. In the example below we construct a $ExtraTreesClassifier$ classifier for the Pima Indians onset of diabetes dataset. You can learn more about the $ExtraTreesClassifier$ class in the scikit-learn API.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html

```
# Feature Importance with Extra Trees Classifier
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# load data
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
```

Verify your understanding:

(e) *Analyse the above code block and run it. You can see that we are given an importance score for each attribute where the larger the score, the more important the attribute. How are these scores calculated?*

(f) *How would you determine the cut-off score?*

2. Resampling

You need to know how well your algorithms perform on unseen data. The best way to evaluate the performance of an algorithm would be to make predictions for new data to which you already know the answers. The second-best way is to use clever techniques from statistics called resampling methods that allow you to make accurate estimates for how well your algorithm will perform on new data. In this section you will discover how you can estimate the accuracy of your machine learning algorithms using resampling methods in Python and scikit-learn on the Pima Indians dataset.

Verify your understanding:

(g) *Why can't you prepare your machine learning algorithm on your training dataset and use predictions from this same dataset to evaluate performance? You probably discussed this during your lecture.*

Soln:
The simple answer is overfitting. Imagine an algorithm that remembers every observation it is shown during training. If you evaluated your machine learning algorithm on the same dataset used to train the algorithm, then an algorithm like this would have a perfect score on the training dataset. But the predictions it made on new data would be terrible. We must evaluate our machine learning algorithms on data that is not used to train the algorithm.

The evaluation is an estimate that we can use to talk about how well we think the algorithm may actually do in practice. It is not a guarantee of performance. Once we estimate the performance of our algorithm, we can then re-train the final algorithm on the entire training dataset and get it ready for operational use. Next up we are going to look at four different techniques that we can use to split up our training dataset and create useful estimates of performance for our machine learning algorithms:

- Train and Test Sets
- k-fold Cross-Validation
- Leave One Out Cross-Validation
- Repeated Random Test-Train Splits.

Split into Train and Test Sets

The simplest method that we can use to evaluate the performance of a machine learning algorithm is to use different training and testing datasets. We can take our original dataset and split it into two parts. Train the algorithm on the first part, make predictions on the second part and evaluate the predictions against the expected results. The size of the split can depend on the size and specifics of your dataset, although it is common to use 67% of the data for training and the remaining 33% for testing.

This algorithm evaluation technique is very fast. It is ideal for large datasets (millions of records) where there is strong evidence that both splits of the data are representative of the underlying problem. Because of the speed, it is useful to use this approach when the algorithm you are investigating is slow to train. A downside of this technique is that it can have a high variance. This means that differences in the training and test dataset can result in meaningful differences in the estimate of accuracy. In the example below we split the Pima Indians dataset into 67%/33% splits for training and test and evaluate the accuracy of a Logistic Regression model.

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result*100.0))
```

Verify your understanding:

(h) *Analyse the above code block, before you run it put a comment at the end of each line. What is the accuracy?*

(i) *Why do you need to specify the random seed?*

Soln:
Accuracy: 75.591%
We specify the random seed. Because the split of the data is random, we want to ensure that the results are reproducible. By specifying the random seed we ensure that we get the same random numbers each time we run the code and in turn the same split of data. This is important if we want to compare this result to the estimated accuracy of another machine learning algorithm or the same algorithm with a different configuration. To ensure the comparison was apples-for-apples, we must ensure that they are trained and tested on exactly the same data.

K-fold Cross-Valudation

Cross-validation is an approach that you can use to estimate the performance of a machine learning algorithm with less variance than a single train-test set split. It works by splitting the dataset into k-parts (e.g. k = 5 or k = 10). Each split of the data is called a fold. The algorithm is trained on k − 1 folds with one held back and tested on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the held back test set. After running cross-validation you end up with k different performance scores that you can summarise using a mean and a standard deviation.

The result is a more reliable estimate of the performance of the algorithm on new data. It is more accurate because the algorithm is trained and evaluated multiple times on different data. The choice of k must allow the size of each test partition to be large enough to be a reasonable sample of the problem, whilst allowing enough repetitions of the train-test evaluation of the algorithm to provide a fair estimate of the algorithms performance on unseen data. For modest sized datasets in the thousands or tens of thousands of records, k values of 3, 5 and 10 are common. In the example below we use 10-fold cross-validation.

```
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

Verify your understanding:

(j) *Analyse the above code block, before you run it put a comment at the end of each line. What is the mean accuracy and the std?*

(k) *Why is the standard deviation an important evaluation measure? Explain.*

Soln:
Accuracy: 76.951% (4.841%)
Because it indicates model stability.

Leave One Out Cross-Validation

You can configure cross-validation so that the size of the fold is 1 (k is set to the number of observations in your dataset). This variation of cross-validation is called leave-one-out cross-validation. The result is a large number of performance measures that can be summarised in an effort to give a more reasonable estimate of the accuracy of your model on unseen data. A downside is that it can be a computationally more expensive procedure than k-fold cross-validation. In the example below we use leave-one-out cross-validation.

```
from pandas import read_csv
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
loocv = LeaveOneOut()
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=loocv)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

Verify your understanding:

(l) *Analyse the above code block, before you run it put a comment at the end of each line. What is the mean accuracy and the std?*

(m) *Why does the standard deviation score have more variance than the k-fold cross-validation?*

Soln:
Accuracy: 76.823% (42.196%)

Repeated Random Test-Train Splits

Another variation on k-fold cross-validation is to create a random split of the data like the train/test split described above but repeat the process of splitting and evaluation of the algorithm multiple times, like cross-validation. This has the speed of using a train/test split and the reduction in variance in the estimated performance of k-fold cross-validation. You can also repeat the process many more times as needed to improve the accuracy. A down side is that repetitions may include much of the same data in the train or the test split from run to run, introducing redundancy into the evaluation. The example below splits the data into a 67%/33% train/test split and repeats the process 10 times.

```
# Evaluate using Shuffle Split Cross Validation
from pandas import read_csv
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
n_splits = 10
test_size = 0.33
seed = 7
kfold = ShuffleSplit(n_splits=n_splits, test_size=test_size, random_state=seed)
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

Verify your understanding:

(n) *Analyse the above code block, before you run it put a comment at the end of each line. What is the mean accuracy and the std?*

(o) *Why does the standard deviation score have less variance than other methods? Compare pros and cons of each method.*

Soln:
Accuracy: 76.496% (1.698%)

- Generally k-fold cross-validation is the gold standard for evaluating the performance of a machine learning algorithm on unseen data with k set to 3, 5, or 10.
- Using a train/test split is good for speed when using a slow algorithm and produces performance estimates with lower bias when using large datasets.
- Techniques like leave-one-out cross-validation and repeated random splits can be useful intermediates when trying to balance variance in the estimated performance, model training speed and dataset size.

The best advice is to experiment and find a technique for your problem that is fast and produces reasonable estimates of performance that you can use to make decisions. If in doubt, use 10-fold cross-validation.

3. References

[1]. Géron, A., 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

[2]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.

[3]. Grus, J., 2019. Data science from scratch: first principles with python. O'Reilly Media.

[4]. Müller, A.C. and Guido, S., 2016. Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".

[5]. Brownlee, J., 2014. Machine learning mastery.

[6]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.

[7]. SAS, 2018, Advanced Predictive Modelling using SAS