Data Mining Final Project - Notebook 2

Datasets :

> **Wine Quality Dataset (White)**

Authors:

> **Francisco Cunha, 76759**
>
> **João Amaral, 76460**

This notebook corresponds to the second one where the Task C of the final project, corresponding to the evaluation of the model and validation strategies for the wine dataset were implemented. Here we computed some strategies in order to select the best hyperparameters for the model in question and obtain the final accuracy results for the problem at hand.

Dataset 2 : White wine quality dataset

Pre-processing steps

In [2]:

```
import pandas as pd
from pandas import DataFrame
from sklearn.preprocessing import StandardScaler

#Disable warning
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning)
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)


dataset_white = pd.read_csv('./white_wine.csv') # numerical: https://archive.ics.uci.edu/ml

dataset_white['quality'] = dataset_white['quality'].apply(lambda value: 'bad' if value <= 5
                                                          if value<= 7 else 'good')
dataset_white['quality'] = pd.Categorical(dataset_white['quality'],categories=['bad','mediu

labels_wine = dataset_white.as_matrix(columns=[dataset_white.columns[-1]]) # Y
attributes_wine = dataset_white.as_matrix(columns=dataset_white.columns[0:11]) # X
features = list(dataset_white.columns[0:11])

sc = StandardScaler()
sc.fit(attributes_wine)
X_white_train_std = sc.transform(attributes_wine)
```

```
['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar', 'chlo
rides', 'free sulfur dioxide', 'total sulfur dioxide', 'density', 'pH', 'sul
phates', 'alcohol']
```

Task C: Model evaluation

In order to determine which one of the trained models and which of its parameters can give us the best results we implemented two different validation techniques. The first one simply being an extension to the holdout validation presented in task B except that now we trained the classifiers for other different partition fractions. However this test isn't very conclusive in the end since the results obtained did not differ much from the original 0.3 partition fraction chosen before and it does not select the best hyperparameters of the model. For the second validation technique a nested cross validation method was implemented using the most basic form of cross-validation, known as **k-fold cross-validation**. It partitions the available data into K disjoint chunks of approximately equal size and in each iteration a training set is formed from a different combination of K − 1 chunks, with the remaining chunk used as the test set used to test the classifier.

In the paper "Nested cross-validation when selecting classifiers is overzealous for most practical applications" the authors consider two procedures for selecting the best algorithm and tuning its hyperparameters via cross-validation: the first called **nested cross-validation** and the second that had no standard name, to which they called **flat cross-validation**.

Flat cross-validation

In flat cross-validation the hyperparameters of each model are tuned to minimise a cross-validation based estimate of generalisation performance. The cross-validation performance estimate, evaluated for those optimal hyperparameter values, is then used to select the best model to use in operation. This approach is computationally inexpensive, with the drawback that the selected resulting model does not correspond to the highest performing one.

Nested cross-validation

The nested cross-validation providse a performance estimate used to select the optimal model. It is based on the fact that when we used the test set to both select the values of the parameter and evaluate the model, we risk optimistically biasing our model evaluations. For this reason, if a test set is used to select model parameters, then we need a different test set to get an unbiased evaluation of that selected mode. This is where the "nested" part comes into play introducing two cross-validation loops: first, an inner cross validation used to tune the parameters and select the best model and the second outer cross validation is used to evaluate the model selected by the inner cross validation. The computational expense of nested cross-validation, however, is substantially higher.

Holdout validation

For this first validation procedure the split fractions considered were 0.15, 0.20, 0.25, 0.30 and 0.35. The SVM classifier was tested with C parameter equaling to 1 and for each different kernel type. The MLP classifiers was also considered. No hyperparameter optimization searching is done here however. As mentioned this procedure is not very conclusive simply being another preliminary process to obtain try and get closer to model the best classifier.

In [13]:

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

# 5 different partitions considered to split train and test set
fractions = [0.15,0.20,0.25,0.30,0.35]
svm_kernel = ['linear','poly','rbf','sigmoid']

scores = []

for frac in fractions:
    X_train,X_test,Y_train,Y_test = train_test_split(X_white_train_std,labels_wine, test_si
    # SVM
    for k in svm_kernel:
        svc = SVC(kernel=k, C=1.0)
        svc.fit(X_train, Y_train)
        predictions = svc.predict(X_test)
    score = accuracy_score(Y_test, predictions)
    scores.append((score,"SVM " + k,frac))
    # MLP
    mlp = MLPClassifier(activation='tanh', hidden_layer_sizes=(10,5),alpha=0.01, max_iter=5
    mlp.fit(X_train, Y_train)
    predictions = mlp.predict(X_test)
    score = accuracy_score(Y_test, predictions)
    scores.append((score, "MLP", frac))

best = max(scores)

print("The best accuracy was : {:.2%} using an holdout fraction of {} and algorithm {}"\
       .format(best[0], best[2], best[1]))
```

```
The best accuracy was : 74.08% using an holdout fraction of 0.3 and algorith
m MLP
```

Nested cross validation

Non-Nested and Nested Cross Validation comparison on the white wine dataset

Scores were computed over 30 trials over nested and non-nested cross validation using K-Fold for each loop, with K=4 using the source code supplied by scikit-learn at https://scikitlearn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.htmlfbclid=IwAR2r bgNO92YoI6nOo (https://scikitlearn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.htmlfbclid=IwAR2 bgNO92YoI6nOo)

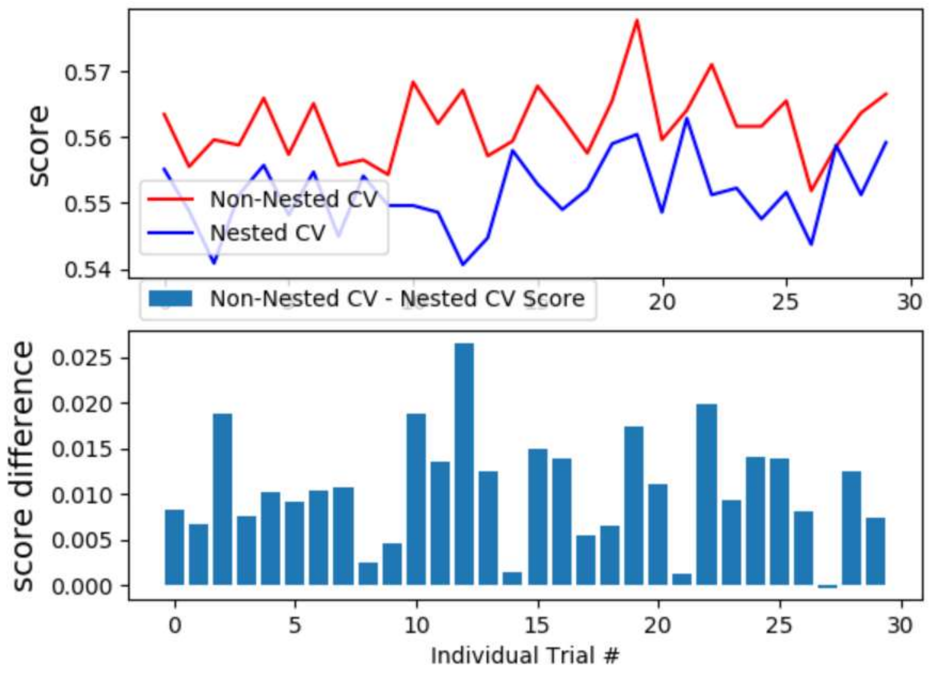
The obtained numerical output with the average of the differences between the scores obtained in the and corresponding plot is presented below:

Average difference of 0.010574 with std. dev. of 0.005965.

In [5]:

```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "./nested_vs_non_nested.png",width=500, height=500)
```

Out[5]:



From this experiment we can conclude that the score difference is not significative and it even ends up being lower in the nested cross validation. One possible reason for this behavior could be due to the fact that, as the paper authors pointed out, the non-nested approach could be biased toward the score obtained since there is no folding applied to the set used to test the classifier in question.

Nested Cross Validation code

Next we present the implement code where the nested cross validation takes place. In a first stage we considered our parameter space for the SVM classifier to be the following:

```
- Kernel: linear, poly, rbf, sigmoid
- Gamma: 1e-2,1e-3, 1e-4
- C: 1, 10, 100, 1000, 10000
```

However the processing time is was taking to calculate reached over 13 hours without until we killed the process and concluded that this approach wasnt going to be feasible with the available time left before the delivery. As such we decided to use a new approach. From the preliminary results obtained in the task B of notebook 1 we can already see a high distinction in the accuracy between the RBF kernel and the remaining. This one out-performs the rest and we can, in a sort of naive approach, assume that it would be the kernel that would give us the best solution, independent from the remaining parameters. As such we then eliminated these bad performing kernels from the parameter space and tested only for Gamma and C allowing the faster extraction of results.

In [20]:

```
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.model_selection import KFold

svm_params = [{'kernel': ['rbf'], 'gamma': [1e-2,1e-3,1e-4],'C': [1, 10,100, 1000, 10000]}]

# CV inner and outer cycles initialization
outer_cv = KFold(n_splits=4, shuffle=True, random_state=None)
# k = 5, first iteration : train on first four folds and test the fifth
inner_cv = KFold(n_splits=4, shuffle=True, random_state=None)

clf = GridSearchCV(estimator=SVC(), param_grid=svm_params, n_jobs=-1, cv=inner_cv) # inner
clf.fit(X_white_train_std, labels_wine)
nested_score = cross_val_score(clf, X=X_white_train_std, y=labels_wine, cv=outer_cv).mean()
print('Best parameters found:\n', clf.best_params_)
# All results
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

```
Best parameters found:
 {'C': 10000, 'gamma': 0.01, 'kernel': 'rbf'}
0.726 (+/-0.028) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.679 (+/-0.024) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.628 (+/-0.021) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.731 (+/-0.014) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.718 (+/-0.022) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.679 (+/-0.023) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.737 (+/-0.016) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.726 (+/-0.021) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.716 (+/-0.030) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.738 (+/-0.016) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.727 (+/-0.013) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.719 (+/-0.021) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.743 (+/-0.015) for {'C': 10000, 'gamma': 0.01, 'kernel': 'rbf'}
0.729 (+/-0.013) for {'C': 10000, 'gamma': 0.001, 'kernel': 'rbf'}
0.725 (+/-0.021) for {'C': 10000, 'gamma': 0.0001, 'kernel': 'rbf'}
```

References

Nested cross-validation when selecting classifiers is overzealous for most practical applications: https://arxiv.org/pdf/1809.09446v1.pdf (https://arxiv.org/pdf/1809.09446v1.pdf)

Nested cross validation: https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html?fbclid=IwAR2mHh8F1RfvHOPNTYoGxWVI_0K8osQrYmn3UghlW3fFS-bgNO92YoI6nOo (https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html?fbclid=IwAR2mHh8F1RfvHOPNTYoGxWVI_0K8osQrYmn3UghlW3fFS-bgNO92YoI6nOo)

Parameter estimation using grid search with a nested cross-validation: http://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/auto_examples/grid_search_digits.html (http://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/auto_examples/grid_search_digits.html)

GridSearchCV with multiple repetitions: https://stackoverflow.com/a/42230764 (https://stackoverflow.com/a/42230764)

Hyperparameter Tuning the Random Forest in Python : https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74 (https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74)

Nested Cross Validation: https://chrisalbon.com/machine_learning/model_evaluation/nested_cross_validation/ (https://chrisalbon.com/machine_learning/model_evaluation/nested_cross_validation/)