10/10

DATA201 - Assignment 4

Please use this page http://apps.ecs.vuw.ac.nz/submit/DATA201 (http://apps.ecs.vuw.ac.nz/submit/DATA201) for submitssion and submit only this single Jupyter notebook with your code added into it at the appropriate places.

The due date is **Saturday 30th May, before midnight**.

The dataset for this assignment is file **whitewine.csv** which is provided with this notebook.

Please choose menu items *Kernel => Restart & Run All* then *File => Save and Checkpoint* in Jupyter before submission.

Dataset ¶

The dataset was adapted from the Wine Quality Dataset (https://archive.ics.uci.edu/ml/datasets/Wine+Quality (https://archive.ics.uci.edu/ml/datasets/Wine+Quality))

Attribute Information:

For more information, read [Cortez et al., 2009: http://dx.doi.org/10.1016/j.dss.2009.05.016 (http://dx.doi.org/10.1016/j.dss.2009.05.016)].

Input variables (based on physicochemical tests):

```
1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol
```

Output variable (based on sensory data):

```
12 - quality (0: normal wine, 1: good wine)
```

Problem statement

Predict the quality of a wine given its input variables. Use AUC (area under the receiver operating characteristic curve) as the evaluation metric.

First, let's load and explore the dataset.

In [1]:

```
import numpy as np
import pandas as pd
import time

from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC

np.random.seed = 42
```

In [2]:

```
data = pd.read_csv("whitewine.csv")
data.head()
```

Out[2]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	tot
0	7.0	0.27	0.36	20.7	0.045	45.0	
1	6.3	0.30	0.34	1.6	0.049	14.0	
2	8.1	0.28	0.40	6.9	0.050	30.0	
3	7.2	0.23	0.32	8.5	0.058	47.0	
4	7.2	0.23	0.32	8.5	0.058	47.0	

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4715 entries, 0 to 4714
Data columns (total 12 columns):
fixed_acidity           4715 non-null float64
volatile_acidity        4715 non-null float64
citric_acid             4715 non-null float64
residual_sugar          4715 non-null float64
chlorides               4715 non-null float64
free_sulfur_dioxide     4715 non-null float64
total_sulfur_dioxide    4715 non-null float64
density                 4715 non-null float64
pH                      4715 non-null float64
sulphates               4715 non-null float64
alcohol                 4715 non-null float64
quality                 4715 non-null int64
dtypes: float64(11), int64(1)
memory usage: 442.2 KB
```

In [4]:

```
data["quality"].value_counts()
```

Out[4]:

```
0    3655
1    1060
Name: quality, dtype: int64
```

Please note that this dataset is unbalanced.

Questions and Code

[1]. Split the given data using stratify sampling into 2 subsets: training (80%) and test (20%) sets. Use random_state = 42. [1 points]

In [5]:

```
train, test = train_test_split(data, test_size=0.2, random_state=42, stratify=data['quality'])

X_train = train.drop('quality', axis=1)
y_train = train['quality'].copy()

X_test = test.drop('quality', axis=1)
y_test = test['quality'].copy()
```

✓ 1/1

[2]. Use `GridSearchCV` and `Pipeline` to tune hyper-parameters for 3 different classifiers including `KNeighborsClassifier`, `LogisticRegression` and `svm.SVC` and report the corresponding AUC values on the training and test sets. Note that a scaler may need to be inserted into each pipeline. [6 points]

Hint: You may want to use `kernel='rbf'` and tune `C` and `gamma` for `svm.SVC`. Find out how to enable probability estimates (for Question 3).

Document: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC (https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC)

In [12]:

```
scaler = MinMaxScaler()

names = [
    "K-Nearest Neighbors",
    "Logistic Regression",
    "SVC"
]

classifiers = [
    KNeighborsClassifier(n_jobs=-1, weights='distance', metric='manhattan'),
    LogisticRegression(n_jobs=-1, solver='saga', max_iter = 4000, C=1),
    SVC(probability=True, kernel='rbf') # 'linear', 'poly', 'rbf', 'sigmoid'
]

parameters = [
    {
        'clf__n_neighbors':list(np.arange(5, 100, 5)),
        'clf__p':[1,2]
    },
    {
        'clf__C':[0.1, 1, 10, 100, 1000],
        'clf__penalty':['l1', 'l2']
    },
    {
        'clf__C':[0.1, 1, 10, 100],
        'clf__gamma':[0.1, 1, 10, 100]
    }
]

pipelines = []

for name, classifier, params in zip(names, classifiers, parameters):
    start = time.time()

    clf_pipe = Pipeline([
        ('scaler', scaler),
        ('clf', classifier)
    ])

    gs_clf = GridSearchCV(clf_pipe, param_grid=params, n_jobs=-1, cv=5, scoring='roc_au
c')
    clf_model = gs_clf.fit(X_train, y_train)
    train_score = clf_model.score(X_train, y_train)
    test_score = clf_model.score(X_test, y_test)
    print("{} best parameters: {}".format(name, clf_model.best_params_))
    print("{} AUC score(training set): {}".format(name, train_score))
    print("{} AUC score(test set): {}".format(name, test_score))
    print("{} Confusion Matrix(training set):\n {}".format(name, confusion_matrix(y_tra
in, clf_model.predict(X_train))))
    print("{} Confusion Matrix(test set):\n {}".format(name, confusion_matrix(y_test, c
lf_model.predict(X_test))))
    pipelines.append((name, clf_model))

    end = time.time()
    print("time: {}\n".format((end-start)/60))
```

```
K-Nearest Neighbors best parameters: {'clf__n_neighbors': 45, 'clf__p': 1}
K-Nearest Neighbors AUC score(training set): 1.0
K-Nearest Neighbors AUC score(test set): 0.9349366337144774
K-Nearest Neighbors Confusion Matrix(training set):
 [[2924    0]
 [   0  848]]
K-Nearest Neighbors Confusion Matrix(test set):
 [[701  30]
 [ 66 146]]
time: 0.13440759579340616

Logistic Regression best parameters: {'clf__C': 100, 'clf__penalty': 'l1'}
Logistic Regression AUC score(training set): 0.7867747883488629
Logistic Regression AUC score(test set): 0.7987184781767029
Logistic Regression Confusion Matrix(training set):
 [[2754  170]
 [ 605  243]]
Logistic Regression Confusion Matrix(test set):
 [[690  41]
 [158  54]]
time: 0.03498464822769165

SVC best parameters: {'clf__C': 1, 'clf__gamma': 100}
SVC AUC score(training set): 0.9991603321890405
SVC AUC score(test set): 0.9088480499703171
SVC Confusion Matrix(training set):
 [[2918    6]
 [  43  805]]
SVC Confusion Matrix(test set):
 [[718  13]
 [112 100]]
time: 0.6369452118873596
```


[3]. Train a soft `VotingClassifier` with the estimators are the three tuned pipelines obtained from [2]. Report the AUC values on the training and test sets. Comment on the performance of the ensemble model. [1 point]

Hint: consider the voting method.

Document: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html#sklearn.ensemble.VotingClassifier (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html#sklearn.ensemble.VotingClassifier)

In [13]:

```
start = time.time()

ensemble = VotingClassifier(estimators=pipelines, voting='soft', n_jobs=-1).fit(X_train
, y_train)
ensemble_train = roc_auc_score(y_train, ensemble.predict_proba(X_train)[:,1], average=
'macro')
ensemble_test = roc_auc_score(y_test, ensemble.predict_proba(X_test)[:,1], average='mac
ro')

print("VotingClassifier AUC score(training set): {}".format(ensemble_train))
print("VotingClassifier AUC score(test set): {}".format(ensemble_test))
print("VotingClassifier Confusion Matrix(training set):\n {}".format(confusion_matrix(y
_train, ensemble.predict(X_train))))
print("VotingClassifier Confusion Matrix(test set):\n {}".format(confusion_matrix(y_tes
t, ensemble.predict(X_test))))

end = time.time()
print("time: {}\n".format((end-start)/60))
```

```
VotingClassifier AUC score(training set): 0.9999903208321503
VotingClassifier AUC score(test set): 0.9399956121105748
VotingClassifier Confusion Matrix(training set):
 [[2923    1]
 [   8  840]]
VotingClassifier Confusion Matrix(test set):
 [[709  22]
 [ 84 128]]
time: 0.691833249727885
```

The ensemble model performs marginally better than K-Nearest Neighbors(the difference is 0.005 so might as well be the same performance), slightly better than SVC and significantly better than logistic regression. The ensemble model doesn't improve on the best performing estimator (KNN) in any meaningful way

1/1

[4]. Redo [3] with a sensible set of `weights` for the estimators. Comment on the performance of the ensemble model in this case. [1 point]

In [14]:

```
start = time.time()

weight_params = []

for w1 in range(1,4):
    for w2 in range(1,4):
        for w3 in range(1,4):
            weight_params.append([w1, w2, w3])

ensemble_weighted = VotingClassifier(estimators=pipelines, voting='soft', n_jobs=-1)
ensemble_gs = GridSearchCV(ensemble_weighted, param_grid={'weights': weight_params}, n_
jobs=-1, cv=3, scoring='roc_auc')
ensemble_fit = ensemble_gs.fit(X_train, y_train)
weighted_train = ensemble_fit.score(X_train, y_train)#roc_auc_score(y_train, ensemble_w
eighted.predict_proba(X_train)[:,1], average='macro')
weighted_test = ensemble_fit.score(X_test, y_test)#roc_auc_score(y_test, ensemble_weigh
ted.predict_proba(X_test)[:,1], average='macro')

print("VotingClassifier best weights: {}".format(ensemble_fit.best_params_))
print("VotingClassifier(weights={}) AUC score(training set): {}".format(ensemble_fit.be
st_params_['weights'], weighted_train))
print("VotingClassifier(weights={}) AUC score(test set): {}".format(ensemble_fit.best_p
arams_['weights'], weighted_test))
print("VotingClassifier(weights={}) Confusion Matrix(training set):\n {}".format(ensemb
le_fit.best_params_['weights'], confusion_matrix(y_train, ensemble_fit.predict(X_train
))))
print("VotingClassifier(weights={}) Confusion Matrix(test set):\n {}".format(ensemble_f
it.best_params_['weights'], confusion_matrix(y_test, ensemble_fit.predict(X_test))))

end = time.time()
print("time: {}\n".format((end-start)/60))
```

```
VotingClassifier best weights: {'weights': [2, 1, 1]}
VotingClassifier(weights=[2, 1, 1]) AUC score(training set): 1.0
VotingClassifier(weights=[2, 1, 1]) AUC score(test set): 0.941073226131172
1
VotingClassifier(weights=[2, 1, 1]) Confusion Matrix(training set):
 [[2924    0]
 [   0  848]]
VotingClassifier(weights=[2, 1, 1]) Confusion Matrix(test set):
 [[710  21]
 [ 76 136]]
time: 24.4167094151179
```

KNN got a perfect 100% accuracy on the training set and the highest AUC score for the test set. It makes it sensible to have a weight of 2 for KNN and 1 for the others. I also tested it out via GridSearchCV and it also gave me 2,1,1 as the best parameters. Giving KNN more voting power gave us a 100% on the training set that we didn't get from the unweighted Voting Classifier. It also gives us a better AUC score for the test set.

1/1

[5]. Use the `VotingClassifier` with `GridSearchCV` to tune the hyper-parameters of the individual estimators. The parameter grid should be a combination of those in [2]. Report the AUC values on the training and test sets. Comment on the performance of the ensemble model. [1 point]

Note that it may take a long time to run your code for this question.

Document: https://scikit-learn.org/stable/modules/ensemble.html#using-the-votingclassifier-with-gridsearchcv (https://scikit-learn.org/stable/modules/ensemble.html#using-the-votingclassifier-with-gridsearchcv)

In [9]:

```
start = time.time()

params = {}
estimators = []
for name, classifier, param in zip(names, classifiers, parameters):
    estimators.append((name, classifier))
    for k in param:
        params[k.replace('clf', 'vote__'+name)] = param[k]

vot_ = VotingClassifier(estimators=estimators, voting='soft', n_jobs=-1)
pipe=Pipeline(steps=[('scale', scaler), ('vote', vot_)])

gs_clf_cv = GridSearchCV(estimator=pipe, param_grid=params, cv=3, n_jobs=-1, scoring='r
oc_auc')
clf_cv = gs_clf_cv.fit(X_train, y_train)
cv_train_score = clf_cv.score(X_train, y_train)
cv_test_score = clf_cv.score(X_test, y_test)

print("VotingClassifier with GridSearchCV best parameters: {}".format(clf_cv.best_param
s_))
print("VotingClassifier with GridSearchCV AUC score(training set): {}".format(cv_train_
score))
print("VotingClassifier with GridSearchCV AUC score(test set): {}".format(cv_test_score
))
print("VotingClassifier with GridSearchCV Confusion Matrix(training set):\n {}".format(
confusion_matrix(y_train, clf_cv.predict(X_train))))
print("VotingClassifier with GridSearchCV Confusion Matrix(test set):\n {}".format(conf
usion_matrix(y_test, clf_cv.predict(X_test))))

end = time.time()
print("time: {}\n".format((end-start)/60))
```

```
VotingClassifier with GridSearchCV best parameters: {'vote__K-Nearest Neig
hbors__n_neighbors': 70, 'vote__K-Nearest Neighbors__p': 2, 'vote__Logisti
c Regression__C': 1000, 'vote__Logistic Regression__penalty': 'l1', 'vote_
_SVC__C': 1, 'vote__SVC__gamma': 100}
VotingClassifier with GridSearchCV AUC score(training set): 0.999991127429
471
VotingClassifier with GridSearchCV AUC score(test set): 0.9399633482177426
VotingClassifier with GridSearchCV Confusion Matrix(training set):
 [[2923    1]
 [   8  840]]
VotingClassifier with GridSearchCV Confusion Matrix(test set):
 [[715  16]
 [ 88 124]]
time: 107.44066168467204
```

Imagine taking 100 minutes to execute and still getting a lower score than the previous two Voting Classifiers. The base Voting Classifier, Voting Classifier with GridSearchCV and SVC all yielded incredibly similiar results whilst the Voting Classifier with estimator weights of 2,1,1 seem to pull ahead by a whopping 0.001 for the test set!

1/1

In [ ]: