DSC 540 - Topic 8- Assignment

September 28, 2021

[1]:
```
import pandas as pd
import numpy as np
import glob
from numpy import mean
from numpy import std
```

[2]:
```
folder_path = 'H:/Krishna/GCU/DSC 540/Topic 8/PAMAP2_Dataset/PAMAP2_Dataset/
↪Protocol'
```

[3]:
```
file_list = glob.glob(folder_path + "/*.dat")
```

[4]:
```
file_list
```

[4]:
```
['H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject101.dat',
 'H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject102.dat',
 'H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject103.dat',
 'H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject104.dat',
 'H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject105.dat',
 'H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject106.dat',
 'H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject107.dat',
 'H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject108.dat',
 'H:/Krishna/GCU/DSC 540/Topic
8/PAMAP2_Dataset/PAMAP2_Dataset/Protocol\\subject109.dat']
```

[5]:
```
df1 = pd.DataFrame(pd.read_csv(file_list[0],sep=' ',header=None))
```

[6]:
```
df2 = pd.DataFrame(pd.read_csv(file_list[1],sep=' ',header=None))
```

[7]:
```
df3 = pd.DataFrame(pd.read_csv(file_list[2],sep=' ',header=None))
```

```
[8]: df4 = pd.DataFrame(pd.read_csv(file_list[3],sep=' ',header=None))
```

```
[9]: df5 = pd.DataFrame(pd.read_csv(file_list[4],sep=' ',header=None))
```

```
[10]: df6 = pd.DataFrame(pd.read_csv(file_list[5],sep=' ',header=None))
```

```
[11]: df7 = pd.DataFrame(pd.read_csv(file_list[6],sep=' ',header=None))
```

```
[12]: df8 = pd.DataFrame(pd.read_csv(file_list[7],sep=' ',header=None))
```

```
[13]: df9 = pd.DataFrame(pd.read_csv(file_list[8],sep=' ',header=None))
```

Concatenate all the dataset into a single dataframe

```
[14]: series = [df1,df2,df3,df4,df5,df6,df7,df8,df9]
      df_final=pd.concat(series)
```

```
[15]: df_final.describe()
```

```
[15]:                 0             1              2             3             4  \
count  2.872533e+06  2.872533e+06  262268.000000  2.859392e+06  2.859392e+06
mean   1.834354e+03  5.466243e+00     109.872508  3.265258e+01 -4.960786e+00
std    1.105689e+03  6.331333e+00      25.870036  1.844274e+00  5.985029e+00
min    5.640000e+00  0.000000e+00      57.000000  2.475000e+01 -1.453670e+02
25%    8.931600e+02  0.000000e+00      90.000000  3.143750e+01 -9.028420e+00
50%    1.790830e+03  3.000000e+00     108.000000  3.312500e+01 -5.788145e+00
75%    2.710570e+03  7.000000e+00     125.000000  3.400000e+01 -7.829420e-01
max    4.475630e+03  2.400000e+01     202.000000  3.550000e+01  6.285960e+01

                  5             6             7             8             9  \
count  2.859392e+06  2.859392e+06  2.859392e+06  2.859392e+06  2.859392e+06
mean   3.587758e+00  3.168417e+00 -4.889420e+00  3.584267e+00  3.349479e+00
std    6.277838e+00  3.843923e+00  5.992726e+00  6.055750e+00  3.840650e+00
min   -1.043010e+02 -1.014520e+02 -6.148950e+01 -6.186800e+01 -6.193470e+01
25%    1.290268e+00  9.685818e-01 -8.933270e+00  1.284680e+00  1.164040e+00
50%    3.570830e+00  2.958415e+00 -5.737615e+00  3.613430e+00  3.132855e+00
75%    6.602720e+00  6.002930e+00 -7.249920e-01  6.601960e+00  6.257612e+00
max    1.556990e+02  1.577600e+02  5.282140e+01  6.225980e+01  6.194460e+01

       …            44            45            46            47  \
count  …  2.860784e+06  2.860784e+06  2.860784e+06  2.860784e+06
mean   …  8.635143e-03 -3.450122e-02  7.752030e-03 -3.272102e+01
std    …  1.073556e+00  5.966026e-01  1.842552e+00  1.887860e+01
min    … -2.399500e+01 -1.812690e+01 -1.401960e+01 -1.728650e+02
25%    … -1.526250e-01 -8.267093e-02 -3.084595e-01 -4.289480e+01
50%    …  4.251595e-03 -4.249850e-03 -2.216015e-03 -3.390020e+01
75%    …  9.464213e-02  8.296868e-02  6.343258e-02 -1.905920e+01
max    …  1.742040e+01  1.358820e+01  1.652880e+01  9.752550e+01
```

```
                 48            49            50            51            52  \
count  2.860784e+06  2.860784e+06  2.860784e+06  2.860784e+06  2.860784e+06
mean   1.593304e+00  1.689044e+01  3.986417e-01  2.154835e-02  3.091533e-01
std    2.161181e+01  2.030858e+01  3.034561e-01  5.691302e-01  3.237875e-01
min   -1.379080e+02 -1.092890e+02 -2.536280e-01 -9.568760e-01 -8.768380e-01
25%   -1.148540e+01  3.289347e+00  1.563440e-01 -5.839910e-01  1.087023e-02
50%    1.362615e+00  1.809105e+01  3.197555e-01  0.000000e+00  3.043820e-01
75%    1.733090e+01  3.087820e+01  5.794420e-01  6.279450e-01  6.020032e-01
max    1.233060e+02  1.469000e+02  1.000000e+00  9.595380e-01  9.923540e-01

                 53
count  2.860784e+06
mean  -1.878725e-02
std    4.731373e-01
min   -9.972810e-01
25%   -5.047580e-01
50%    0.000000e+00
75%    4.634432e-01
max    9.961050e-01

[8 rows x 54 columns]
```

Due to the Memory constraint of the machine in which this program is being executed we are limiting the pre-processing activities

[16]:
```python
df_final_na=df_final.fillna(0)
```

[17]:
```python
df_final_na.head(5)
```

[17]:
```
      0  1      2     3        4        5        6        7        8  \
0  8.38  0  104.0  30.0  2.37223  8.60074  3.51048  2.43954  8.76165
1  8.39  0    0.0  30.0  2.18837  8.56560  3.66179  2.39494  8.55081
2  8.40  0    0.0  30.0  2.37357  8.60107  3.54898  2.30514  8.53644
3  8.41  0    0.0  30.0  2.07473  8.52853  3.66021  2.33528  8.53622
4  8.42  0    0.0  30.0  2.22936  8.83122  3.70000  2.23055  8.59741

         9  …        44        45        46       47       48       49   50  \
0  3.35465  …  0.008300  0.009250 -0.017580 -61.1888 -38.9599 -58.1438  1.0
1  3.64207  … -0.006577 -0.004638  0.000368 -59.8479 -38.8919 -58.5253  1.0
2  3.73280  …  0.003014  0.000148  0.022495 -60.7361 -39.4138 -58.3999  1.0
3  3.73277  …  0.003175 -0.020301  0.011275 -60.4091 -38.7635 -58.3956  1.0
4  3.76295  …  0.012698 -0.014303 -0.002823 -61.5199 -39.3879 -58.2694  1.0

    51   52   53
0  0.0  0.0  0.0
1  0.0  0.0  0.0
2  0.0  0.0  0.0
```

```
3  0.0  0.0  0.0
4  0.0  0.0  0.0

[5 rows x 54 columns]
```

We see that the heart rate is missing infomration and has been captured every 10 to 11 seconds. So we will use the Linear interpolation method to fill the missing heart rate date and will add it as a new column

[18]:
```
# By default interpolate function uses Linear interpolation and we are filling␣
↪a maximum of 10 missing values using this function
df_final_na[54]=df_final[2].interpolate(limit=10)
# The interpolated data is being added as a new column to our dataframe.
```

[19]:
```
df_final_na.head(11)
```

[19]:
```
       0  1      2     3        4        5        6        7        8  \
0   8.38  0  104.0  30.0  2.37223  8.60074  3.51048  2.43954  8.76165
1   8.39  0    0.0  30.0  2.18837  8.56560  3.66179  2.39494  8.55081
2   8.40  0    0.0  30.0  2.37357  8.60107  3.54898  2.30514  8.53644
3   8.41  0    0.0  30.0  2.07473  8.52853  3.66021  2.33528  8.53622
4   8.42  0    0.0  30.0  2.22936  8.83122  3.70000  2.23055  8.59741
5   8.43  0    0.0  30.0  2.29959  8.82929  3.54710  2.26132  8.65762
6   8.44  0    0.0  30.0  2.33738  8.82900  3.54767  2.27703  8.77828
7   8.45  0    0.0  30.0  2.37142  9.05500  3.39347  2.39786  8.89814
8   8.46  0    0.0  30.0  2.33951  9.13251  3.54668  2.44371  8.98841
9   8.47  0    0.0  30.0  2.25966  9.09415  3.43015  2.42877  9.01871
10  8.48  0  104.0  30.0  2.29745  8.90450  3.46984  2.39736  8.94335

          9  …        45        46       47       48       49   50   51  \
0   3.35465  …  0.009250 -0.017580 -61.1888 -38.9599 -58.1438  1.0  0.0
1   3.64207  … -0.004638  0.000368 -59.8479 -38.8919 -58.5253  1.0  0.0
2   3.73280  …  0.000148  0.022495 -60.7361 -39.4138 -58.3999  1.0  0.0
3   3.73277  … -0.020301  0.011275 -60.4091 -38.7635 -58.3956  1.0  0.0
4   3.76295  … -0.014303 -0.002823 -61.5199 -39.3879 -58.2694  1.0  0.0
5   3.77788  … -0.016024  0.001050 -60.2954 -38.8778 -58.3977  1.0  0.0
6   3.73230  … -0.053934  0.015594 -60.6307 -38.8676 -58.2711  1.0  0.0
7   3.64131  … -0.039937 -0.000785 -60.5171 -38.9819 -58.2733  1.0  0.0
8   3.62596  … -0.010042  0.017701 -61.2916 -39.6182 -58.1499  1.0  0.0
9   3.61081  … -0.013923  0.014498 -60.8509 -39.0821 -58.1478  1.0  0.0
10  3.53551  …  0.002283  0.020352 -61.5302 -38.7240 -58.3860  1.0  0.0

     52   53     54
0   0.0  0.0  104.0
1   0.0  0.0  104.0
2   0.0  0.0  104.0
3   0.0  0.0  104.0
4   0.0  0.0  104.0
```

```
5   0.0  0.0  104.0
6   0.0  0.0  104.0
7   0.0  0.0  104.0
8   0.0  0.0  104.0
9   0.0  0.0  104.0
10  0.0  0.0  104.0

[11 rows x 55 columns]
```

[20]:
```python
# Let's split the Predictor variables and the response variables
X = df_final_na.iloc[:,3:55]
y=df_final_na[1]
```

[21]:
```python
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
```

[22]:
```python
# Let us standardize the input variables that is used to predict the activity
X_trans=pd.DataFrame(StandardScaler().fit_transform(X))
```

[23]:
```python
# The standardized data post the standardization will be as follows
X_trans.head(5)
```

[23]:
```
         0         1         2         3         4         5         6   \
0 -0.87198  1.222319  0.802375  0.092828  1.220186  0.858942  0.005339
1 -0.87198  1.191577  0.796768  0.132220  1.212738  0.824073  0.080217
2 -0.87198  1.222543  0.802427  0.102851  1.197741  0.821697  0.103854
3 -0.87198  1.172576  0.790854  0.131809  1.202774  0.821660  0.103846
4 -0.87198  1.198431  0.839145  0.142168  1.185285  0.831780  0.111708

          7         8         9   …        42        43        44        45  \
0 -0.068393  0.028049 -0.006698  …  0.073248 -0.013759 -1.508894 -1.879963
1 -0.016011  0.017820  0.008705  …  0.049921 -0.003998 -1.438154 -1.876811
2 -0.041940  0.000662 -0.000586  …  0.057960  0.008035 -1.485012 -1.901009
3  0.001032  0.000928  0.001651  …  0.023614  0.001933 -1.467761 -1.870857
4  0.012327 -0.015462 -0.032564  …  0.033688 -0.005734 -1.526362 -1.899808

          46        47        48        49        50        51
0 -3.693656  1.984157 -0.037784 -0.951077  0.039626 -0.226739
1 -3.712453  1.984157 -0.037784 -0.951077  0.039626 -0.226739
2 -3.706274  1.984157 -0.037784 -0.951077  0.039626 -0.226739
3 -3.706062  1.984157 -0.037784 -0.951077  0.039626 -0.226739
4 -3.699844  1.984157 -0.037784 -0.951077  0.039626 -0.226739
```

```
[5 rows x 52 columns]
```

[24]:
```
# Split the data into training and test dataset
X_train, X_test, y_train, y_test = train_test_split(X_trans.fillna(0), y,␣
↪test_size=0.995, random_state=42)
```

[25]:
```
def get_models():
    models = dict()
    models['knn'] = KNeighborsClassifier()
    models['cart'] = DecisionTreeClassifier()
    models['svm'] = SVC()
    return models
```

[26]:
```
# Cross Validation Function to evaluate the model based on accuracy
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1,␣
↪error_score='raise')
    return scores
```

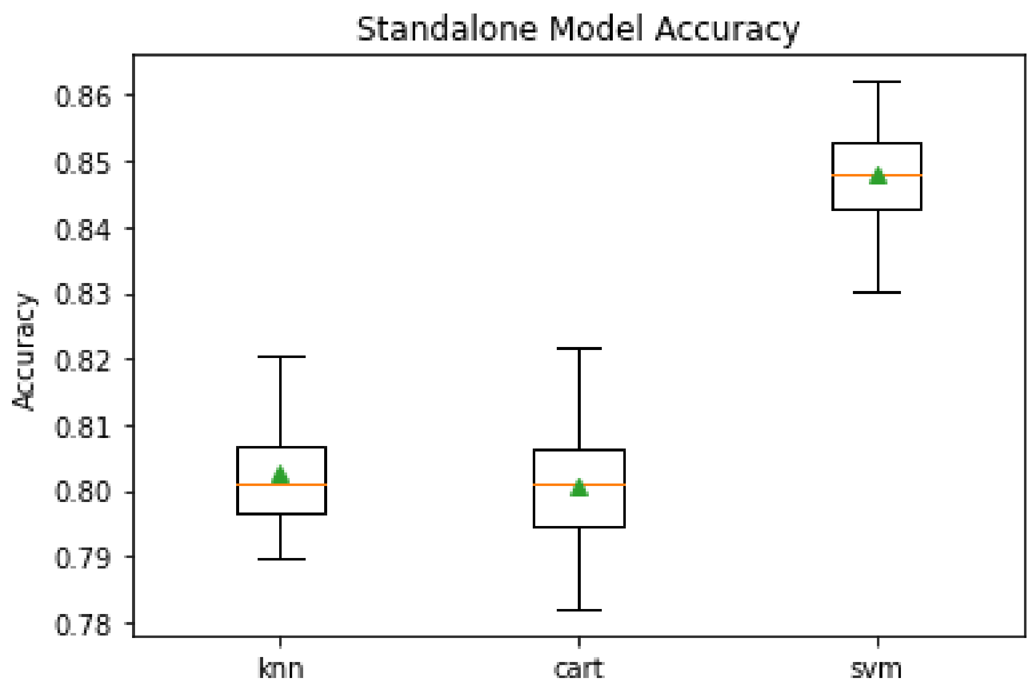
[27]:
```
models = get_models()
models.items()
```

[27]:
```
dict_items([('knn', KNeighborsClassifier()), ('cart', DecisionTreeClassifier()),
('svm', SVC())])
```

[28]:
```
results_base, names_base = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X_train, y_train)
    results_base.append(scores)
    names_base.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
```

```
>knn 0.803 (0.008)
>cart 0.801 (0.009)
>svm 0.848 (0.008)
```

[48]:
```
import matplotlib.pyplot as plt
plt.boxplot(results_base, labels=names_base, showmeans=True)
plt.title('Standalone Model Accuracy')
plt.ylabel('Accuracy')
plt.show()
```



Let us use our Voting Ensemble model (Custom) and the conventional Ensemble models to see if the accuracy score of the model has improved or deprecated. For the conventional Ensemble model we have Bagging, Random Forest and Adaboosting models.

[30]:
```
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
```

[31]:
```
# Let us define all our base model that need to be used in our ensemble method␣
 ↪and the voting method
def get_voting():
    # define the base models
    models_en = list()
    models_en.append(('knn', KNeighborsClassifier()))
    models_en.append(('cart', DecisionTreeClassifier()))
    models_en.append(('svm', SVC()))

    # define the voting ensemble
    ensemble = VotingClassifier(estimators=models_en, voting='hard')
    return ensemble
```

[34]:
```
def get_models_en():
    models_en = dict()
    models_en['voting'] = get_voting()
```

```python
    models_en['bagging'] = BaggingClassifier()
    models_en['boosting'] = AdaBoostClassifier()
    models_en['Random Forest'] = RandomForestClassifier()
    return models_en
```

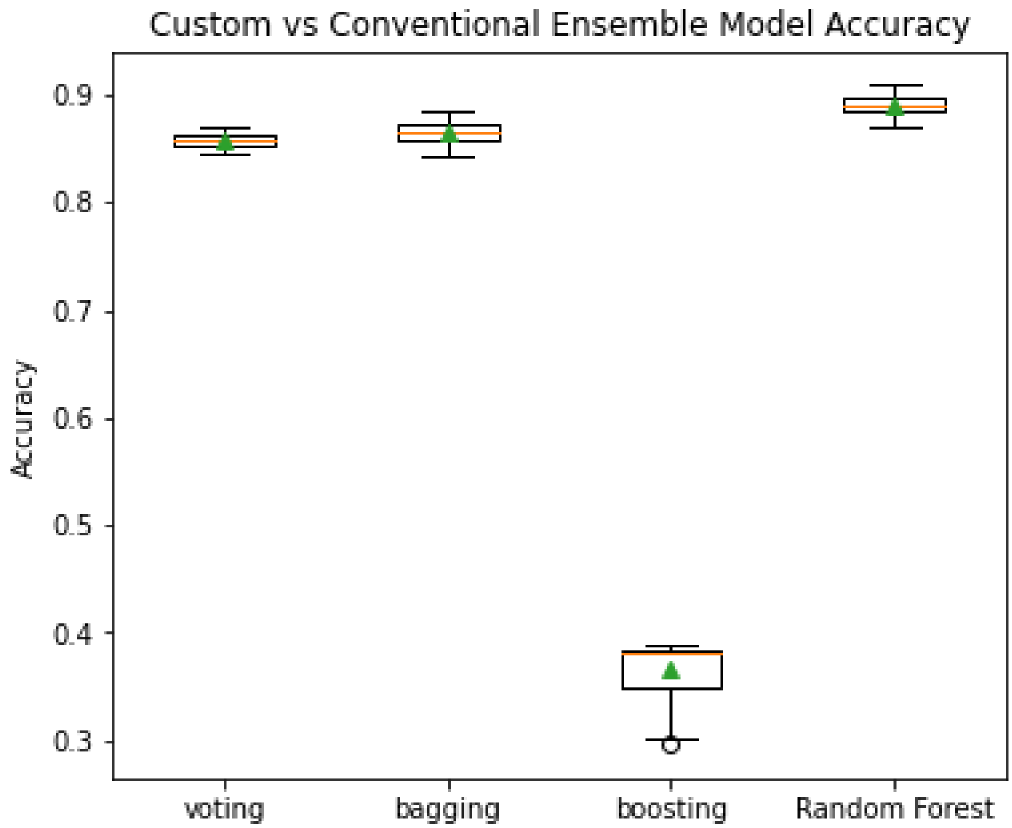
[35]:
```python
models_en = get_models_en()
models_en.items()
```

[35]:
```
dict_items([('voting', VotingClassifier(estimators=[('knn',
KNeighborsClassifier()),
                              ('cart', DecisionTreeClassifier()),
                              ('svm', SVC())])), ('bagging',
BaggingClassifier()), ('boosting', AdaBoostClassifier()), ('Random Forest',
RandomForestClassifier())])
```

[43]:
```python
models_en = get_models_en()
results, names = list(), list()
for name, model in models_en.items():
    scores_en = evaluate_model(model, X_train, y_train)
    results.append(scores_en)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores_en), std(scores_en)))
```

```
>voting 0.857 (0.006)
>bagging 0.865 (0.010)
>boosting 0.368 (0.026)
>Random Forest 0.891 (0.009)
```

[49]:
```python
fig, (ax0) = plt.subplots(1, figsize=(6, 5))
ax0.boxplot(results, labels=names, showmeans=True)
plt.title('Custom vs Conventional Ensemble Model Accuracy')
plt.ylabel('Accuracy')
plt.show()
```



From above we can see that the overall model score/accuracy has improved with the Ensemble model (Combined with weighted Voting)

```
[ ]:
```

```
[ ]:
```