Martinez-assn-8-nb

October 27, 2020

#

Week 8 - Ensemble Methods

##

Seve Martinez

##

Grand Canyon University

###

Abstract

Ensemble methods can be a great way to improve the predictive ability of a data science model. There are several methods including bagging (bootstrap aggregating), bagging, and others. In this paper, a custom implementation is proposed using three fusion methods: Weighted Majority Voting, Behavior Knowledge Space, and Naive Bayes Combination. The data set PAMAP is used from the UCI machine learning repository.

Keywords: ensemble methods, ANN, BDT, SVM, weighted majority voting

0.0.1 Week 8 – Ensemble Methods

Ensemble methods seek to take a series of weak learners and combine their results to build a stronger and more robust model.

##

Methods

0.0.2 Week 8 – Data Set

The data set comes from the PAMAP Physical Activity Monitoring study from UCI Machine learning repository. The data contains 54 columns of various data points from three accelerometers, worn in different locations on the body. Nine subjects were measured while doing various activities of differing exertion levels. Some activities include walking, lying down, sitting, ironing, cycling, and others. A sampling rate of 100Hz was used to capture data on a 3D axis of ± 16g. While three locations were used, only the wrist location was used for the study (Chowdhary et al, 2017).

Additional features were then calculated from the three columns of wrist data which include mean, median, standard deviation, variance, skewness, and kurtosis. Chowdhary et al used a 10-second sliding window on the timestamp value, but this proved to be programatically challenging so, given the time, a random sampling of ten records was chosen for each row set of the additional features. This will produce somewhat inaccurate data because dissimilar time stamped values could be for different activities, which will skew the data. However, for the purposes of this paper, this was an acceptable compromise.

[247]:
```
import pandas as pd
import numpy as np
import os # for the file merging
from sklearn.impute import SimpleImputer # To handle the null values
from matplotlib import pyplot as plt
from sklearn.ensemble import VotingClassifier # Weighted Voting ('soft')
from sklearn.neighbors import KNeighborsClassifier # KNN
from sklearn.svm import SVC # Support vector machine
from sklearn.neural_network import MLPClassifier # ANN
from sklearn.tree import DecisionTreeClassifier # Binary decision tree
from sklearn.naive_bayes import MultinomialNB # For the Behavior Knowledge Space
from sklearn.naive_bayes import GaussianNB # Naive Bayes
import itertools
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
import pickle
from sklearn.metrics import accuracy_score
```

[31]:
```
# There are nine files, one per subject. So all will be combined in order to␣
↪simplify the process.

path = '../data/PAMAP2_Dataset/Protocol'

file_list = os.listdir(path)
for filename in sorted(file_list):
    out_filename = 'pamap.txt'
    with open(out_filename, 'a') as outfile:
        with open(path + '/' + filename, 'r') as infile:
            outfile.write(infile.read())
```

[222]:
```
# Load the combined subject dataset into a dataframe
# The study only used the wrist data, so the rest is dropped

cols = ['timestamp', 'activityID', 'heartrate', 'w1', 'w2', 'w3', 'w4', 'w5',␣
↪'w6'
        ,'w7', 'w8', 'w9', 'w10', 'w11', 'w12', 'w13', 'w14', 'w15', 'w16',␣
↪'w17']

df = pd.read_csv('pamap.txt', ' ', header=None)

df.drop(df.iloc[:, 20:], axis = 1, inplace=True)
```

```
df.columns=cols
```

[223]:
```
df.shape
```

[223]: (2872533, 20)

[224]:
```
df['activityID'].value_counts()
```

[224]:
```
0     929661
4     238761
17    238690
1     192523
3     189931
7     188107
2     185188
16    175353
6     164600
12    117216
13    104944
5      98199
24     49360
Name: activityID, dtype: int64
```

0.1 Preprocessing

[225]:
```
# Per the dataset instructions, an activityID of 0 should be removed
# The study did not explicitly state whether or not they removed it,
# but it would be prudent to do so.
# Only the data from the accelerometer is used so heartrate will also be
↪dropped.
# In addition, only values w2-4 are for the +/- 16g accelerometer
# Which is the one used in the study. The other columns will be dropped.

df.drop(df[df['activityID'] == 0].index, inplace=True)
#df.drop('heartrate', inplace=True, axis=1)

keeps = ['timestamp', 'activityID', 'w2', 'w3', 'w4']
df = df[keeps]

df.columns
```

[225]: Index(['timestamp', 'activityID', 'w2', 'w3', 'w4'], dtype='object')

[226]:
```
# So we have quite a few null values that will have to be handled.
# There are a consistent number of nulls that comprise only 0.5% of the entire
↪dataset
```

```
# The easiest thing would be to drop them, but in the spirit of the study, they␣
↪will
# be imputed.

# The authors used linear interpolation to determine the values, but for␣
↪simplicity,
# mean will be used here.
df[df['w2'].isnull()]
```

[226]:

```
         timestamp  activityID  w2  w3  w4
19193       200.31           1 NaN NaN NaN
19194       200.32           1 NaN NaN NaN
19195       200.33           1 NaN NaN NaN
34152       349.90           2 NaN NaN NaN
45999       468.37           2 NaN NaN NaN
…              …           …  ..  ..  ..
2843922    3884.86          24 NaN NaN NaN
2843923    3884.87          24 NaN NaN NaN
2843950    3885.14          24 NaN NaN NaN
2871906      93.97          24 NaN NaN NaN
2871907      93.98          24 NaN NaN NaN

[11124 rows x 5 columns]
```

[227]:

```
# The authors used linear interpolation to determine the values, but for␣
↪simplicity,
# mean will be used here.
# Use the simpleimputer class from sklearn and leverage 'mean' as the fill value
fill_NaN = SimpleImputer(missing_values=np.nan, strategy='mean')
imputed_df = pd.DataFrame(fill_NaN.fit_transform(df))

# Rebuild the dataframe
imputed_df.columns = df.columns
imputed_df.index = df.index

# Verify it worked
imputed_df.head().isnull().any()
```

[227]:

```
timestamp     False
activityID    False
w2            False
w3            False
w4            False
dtype: bool
```

[228]:

```
imputed_df.shape
```

[228]: (1942872, 5)

[229]:
```
# Remove the first ten seconds and the last ten seconds
# to ensure steady state data

indexNames = imputed_df[(imputed_df['timestamp'] <= imputed_df['timestamp'].
↪min()+10) | (imputed_df['timestamp'] >= imputed_df['timestamp'].max()-10)].
↪index

# Remove the values
imputed_df.drop(indexNames , inplace=True)


# Reset the index after doing all this
imputed_df.reset_index(inplace=True)
imputed_df.drop('index', inplace=True, axis=1)
```

[230]:
```
imputed_df
```

[230]:

	timestamp	activityID	w2	w3	w4
0	41.21	1.0	-1.34587	9.57245	2.83571
1	41.22	1.0	-1.76211	10.63590	2.59496
2	41.23	1.0	-2.45116	11.09340	2.23671
3	41.24	1.0	-2.42381	11.88590	1.77260
4	41.25	1.0	-2.31581	12.45170	1.50289
…	…	…	…	…	…
1940510	95.06	24.0	4.99466	6.01881	5.59830
1940511	95.07	24.0	5.02764	5.90369	5.48372
1940512	95.08	24.0	5.06409	5.71370	5.48491
1940513	95.09	24.0	5.13914	5.63724	5.48629
1940514	95.10	24.0	5.00812	5.40645	5.02326

[1940515 rows x 5 columns]

[231]:
```
imputed_df = imputed_df.sample(n=200000)
imputed_df.reset_index(inplace=True)
imputed_df.drop('index', inplace=True, axis=1)
t = imputed_df.sample(n=10)

new_serie = t.agg(['sum'
                  ,'mean'
                  ,'var'
                  ,'std'
                  ,'skew'
                  ,'kurt'
                  ,'median'
                  ,'min'
```

```
                  ,'max']).unstack()

new_df = pd.concat([imputed_df, new_serie.set_axis([f'{x}_{y}'
                               for x, y in new_serie.index])
                                 .to_frame().T], axis=1)
```

[232]:
```
# loop back through and update the dataset
for i in range(0, len(imputed_df)):
    t = imputed_df.sample(n=10)
    new_serie = t.agg(['sum'
                       ,'mean'
                       ,'var'
                       ,'std'
                       ,'skew'
                       ,'kurt'
                       ,'median'
                       ,'min'
                       ,'max']).unstack()


    #if new_df already exist:
    new_df.loc[i, :] = new_serie.set_axis([f'{x}_{y}' for x, y in new_serie.
 ↪index])

```

[234]:
```
# for some reason it clobbers the original rows, so drop them and rejoin to the␣
 ↪old one
new_df.drop(['timestamp', 'activityID', 'w2', 'w3', 'w4'], inplace=True, axis=1)
final_df = imputed_df.join(new_df)
```

[236]:
```
final_df
```

[236]:
```
        timestamp  activityID        w2         w3        w4  timestamp_sum  \
0         2721.29         7.0  -6.71878  14.914300  1.467120       16541.81
1         1840.01        13.0  -6.55117   3.081560  2.202320       15757.92
2          900.84        17.0  -6.76244   2.671800  7.088440       12621.53
3          194.16         1.0   8.45717  -1.454100  4.597730       22653.49
4         3000.69         4.0 -13.20160   2.120250  3.414520       20768.58
…             …          …         …         …         …            …
199995    3010.11         6.0  -7.48967   1.103940  9.315100        9234.84
199996    3030.91         6.0 -10.28910  -0.221495  2.927220       18681.62
199997     302.91         1.0   5.08045  -0.256253  8.276380       13991.12
199998    2385.57         4.0  -5.62356   7.983600 -0.378975       20974.66
199999    2914.80         4.0 -12.45550   5.443550 -1.896780       19454.94

        timestamp_mean  timestamp_var  timestamp_std  timestamp_skew  …  \
0             1654.181   9.264337e+05     962.514282       -0.393664  …
```

```
1          1575.792  1.056256e+06   1027.743333        0.447349  …
2          1262.153  7.861352e+05    886.642656        0.309393  …
3          2265.349  1.361728e+06   1166.930937       -1.127839  …
4          2076.858  1.003401e+06   1001.699270       -0.837843  …
…               …          …            …             …  …
199995      923.484  1.020397e+06   1010.147091        2.354490  …
199996     1868.162  1.802452e+06   1342.554208        0.385985  …
199997     1399.112  1.188831e+06   1090.335146        0.680787  …
199998     2097.466  9.796520e+05    989.773724       -0.694516  …
199999     1945.494  7.896505e+05    888.622826       -0.028955  …

         w3_skew   w3_kurt  w3_median     w4_sum   w4_mean     w4_var  \
0       1.896776  5.104444   4.041105  48.114755  4.811476   9.590619
1       0.417718 -0.043661   3.638440  24.615158  2.461516  16.156464
2       0.568219  1.688135   4.994155  46.046140  4.604614   6.160259
3       0.082441  0.119479   4.502360  20.977467  2.097747  13.730057
4      -0.441202  0.091522   3.690055  37.447436  3.744744   8.568038
…            …         …         …          …         …
199995 -2.911651  8.854461   5.893685  48.164091  4.816409  10.268572
199996  1.572348  4.206410   2.572815  54.723290  5.472329   8.105623
199997 -2.739108  8.179653   4.010380  36.827149  3.682715   9.507849
199998 -0.016564 -1.363516  -1.474430  34.968217  3.496822   5.330679
199999  0.871773  0.776339   2.336750  54.638828  5.463883  11.685051

          w4_std   w4_skew   w4_kurt  w4_median
0       3.096873  0.257536 -1.661255   3.986475
1       4.019510  0.570342 -0.759230   1.827280
2       2.481987  0.259665 -1.374382   4.409425
3       3.705409 -0.713149  0.593306   2.404555
4       2.927121  0.204204 -0.517594   3.506080
…            …         …         …          …
199995  3.204461 -1.138176 -0.169199   6.001230
199996  2.847038  0.386852 -0.504066   5.113615
199997  3.083480  0.309082 -1.684113   2.860450
199998  2.308826  0.630594  0.062462   3.266260
199999  3.418340  0.108031 -1.759184   4.558575

[200000 rows x 40 columns]
```

[237]:
```
final_df.to_csv('out.csv')
```

0.2 Unit Scaling

[238]:
```
se = StandardScaler()

X = final_df.drop('activityID', axis=1)
X = se.fit_transform(X)
```

```
y = final_df['activityID']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state=69)
```

0.3 The Models

0.3.1 Binary Decision Tree

[239]:
```
# 20 was the max depth the authors used.
dt_clf = DecisionTreeClassifier(random_state=69, max_depth=20)
```

0.3.2 K-Nearest Neighbors

[240]:
```
# Authors used 7 neighbors
k_clf = KNeighborsClassifier(n_neighbors=7)
```

0.3.3 Support Vector Machine

[241]:
```
# Implement a 'one-vs-rest' type SVM
svm_clf = SVC(decision_function_shape='ovr', probability=True, kernel='linear')
```

0.3.4 Artificial Neural Network

[242]:
```
# 50 neurons in the hidden layer
# linear activation 'ReLu'
# learning rate = 0.001
# 250 epochs

ann_clf = MLPClassifier(hidden_layer_sizes=(50,), max_iter=250, random_state=69)
```

0.3.5 Weighted Majority Voting

[243]:
```
# unleash the kraken!
wmv_clf = VotingClassifier(estimators=[
        ('BDT', dt_clf)
    ,('knn', k_clf)
    ,('svm', svm_clf)
    ,('ann', ann_clf)], voting='soft', n_jobs=-1)


wmv_clf.fit(X_train,y_train)
```

[243]:
```
VotingClassifier(estimators=[('BDT',
                              DecisionTreeClassifier(max_depth=20,
                                                     random_state=69)),
                             ('knn', KNeighborsClassifier(n_neighbors=7)),
```

```
                              ('svm', SVC(kernel='linear', probability=True)),
                              ('ann',
                               MLPClassifier(hidden_layer_sizes=(50,),
                                             max_iter=250, random_state=69))],
                 n_jobs=15, voting='soft')
```

[244]:
```python
y_pred = wmv_clf.predict(X_test)

# Build the confusion matrix
confusion_matrix(y_test, y_pred)
```

[244]:
```
array([[3857,   25,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           4],
       [  15, 3814,   69,    0,    0,    0,    0,    0,    0,    0,    0,
           0],
       [   0,   85, 3712,    0,    0,    0,    0,    0,    0,    0,   88,
           0],
       [   0,    0,    0, 4995,    2,   49,   49,    2,    4,    0,    0,
           0],
       [   0,    0,    0,   10, 1765,   83,  111,    0,    0,    0,    0,
          36],
       [   0,    0,    0,   89,   42, 3147,   65,    0,    0,    0,    0,
           8],
       [   0,    0,    0,   59,   88,   48, 3541,    0,    7,    0,    0,
           4],
       [   0,    0,    0,  137,    0,    0,    4, 1921,  330,  107,    0,
           0],
       [   0,    0,    0,   47,    0,    0,   25,  522, 1654,    0,    0,
           0],
       [   0,    0,    0,    0,    0,    0,    0,   74,    0, 3364,   99,
           0],
       [   0,    1,   84,    0,    0,    0,    0,    0,    0,   73, 4761,
           0],
       [  35,    0,    0,    0,   83,   56,   13,    0,    0,    0,    0,
         737]])
```

0.4 Scoring

[245]:
```python
wmv_clf.score(X_test, y_test)
```

[245]:
```
0.9317
```

[251]:
```python
y_test.value_counts()
```

[251]:
```
4.0     5101
17.0    4919
2.0     3898
```

```
1.0     3886
3.0     3885
7.0     3747
16.0    3537
6.0     3351
12.0    2499
13.0    2248
5.0     2005
24.0     924
Name: activityID, dtype: int64
```

[250]:
```python
# See how weak weak classifier did prior to the ensemble
print('Ann: ', {wmv_clf.named_estimators_['ann'].score(X_test, y_test)})
print('BDT: ', {wmv_clf.named_estimators_['BDT'].score(X_test, y_test)})
print('SVM: ', {wmv_clf.named_estimators_['svm'].score(X_test, y_test)})
print('knn: ', {wmv_clf.named_estimators_['knn'].score(X_test, y_test)})
```

```
Ann:  {0.008025}
BDT:  {0.0034}
SVM:  {0.02585}
knn:  {0.040525}
```

0.4.1 BKS Combination

[ ]:
```python
# Wasn't sure how to do this one, so used a multinomialNB to combine the␣
↪classifiers' outputs.



bks_clf = MultinomialNB().fit(nb_data, y_train)
```