Coursework_3_lyapina

March 28, 2018

1 Lab 3: Information facial age estimation

The aim of this task is to estimate age from facial images. The program built, which takes the AAM parameters as representation of human faces and learn a regression function to predict age for an unseen face.

The FG-NET Aging dataset is used (http://sting.cycollege.ac.cy/~alanitis/fgnetaging/index.htm). The dataset contains 1,002 high-resolution color or gray-scale face images of 82 multiple- race subjects with large variation of lighting, pose, and expression. The age range is from 0 to 69 years with chronological aging images available for each subject (on average, 12 images per subject). The 1,002 images are split into a training set and a test set, each of which has about half the images. An AAM feature vector is extracted from each image and used as the representation. The .mat file contains both the feature vector and the true age label for each image, but not the image itself.

In [1]:
```
import scipy.io
import numpy as np
from texttable import Texttable
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression
from sklearn.svm import SVR
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Path
database_path = './data_age.mat';
result_path = './results/';

# Initial states
absTestErr = 0;
cs_number = 0;
```

```
# Cumulative error level
err_level = 5;
```

In [3]:
```
# Preparing the data
mat_data = scipy.io.loadmat(database_path)

teData = {'label': mat_data['teData'][0][0][0], 'feat': mat_data['teData'][0][0][1]}
trData = {'label': mat_data['trData'][0][0][0], 'feat': mat_data['trData'][0][0][1]}

nTrain = len(trData['label']) # number of training samples
nTest  = len(teData['label']) # number of testing samples
# Training
xtrain = trData['feat'] # feature
ytrain = trData['label'] # labels
# Testing
xtest = teData['feat'] # feature
ytest = teData['label'] # labels
```

1.1 Regression method: linear regression

In [4]:
```
regressor = LinearRegression(fit_intercept = False)
regressor.fit(xtrain, ytrain)
w_lr = regressor.coef_[0]

yhat_test = np.matmul(xtest, w_lr)
```

Cumulative error level indicates the error (ground truth age minus predicted age) that you allow the model to make. Say, we set the level to 5, for a person of age 10, the predicted age between 10 +/- 5 is counted as a correct prediction. Other predictions violating this error range would be counted as incorrect.

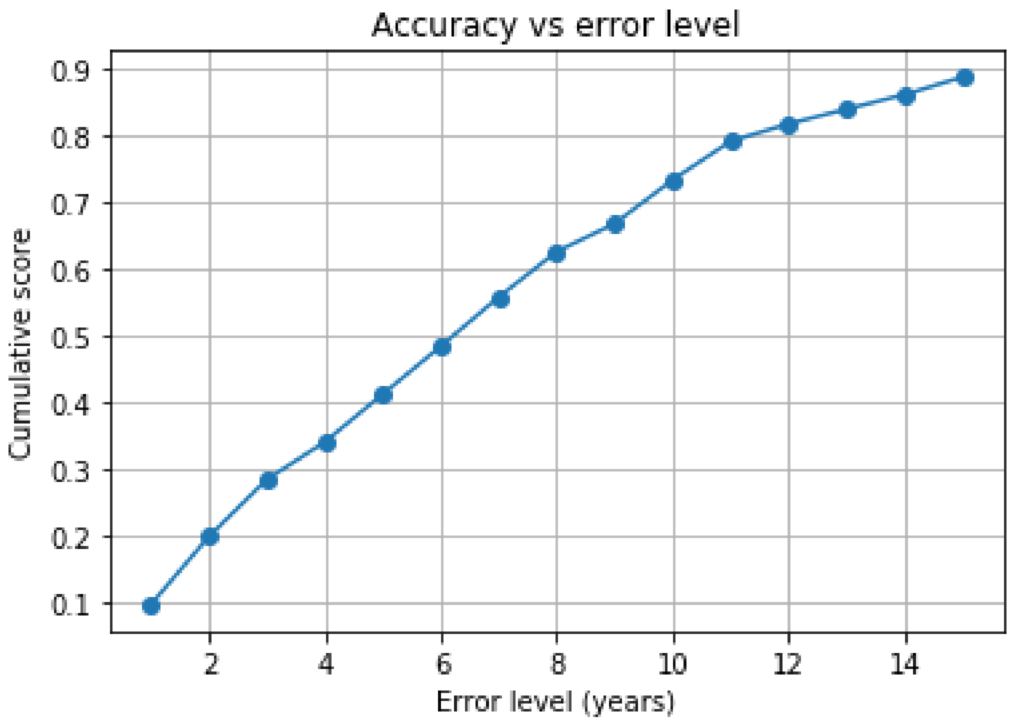
1.2 Computing the MAE and CS value (with cumulative error level of 5) for linear regression.

In [5]:
```
print("MAE: %s" %(mean_absolute_error(ytest, yhat_test)))
a =  abs(ytest[:,0] - yhat_test)
print("CS value: %s" %(len(np.where(a <= err_level)[0])/len(a) *100))
```

```
MAE: 7.70435906664
CS value: 41.235059760956176
```

In [6]:
```
# Generating a cumulative score (CS) vs. error level plot by varying the error level fro
values = []
errors = []
for i in range(1, 16):
    cs = len(np.where(a <= i)[0])/len(a)
    values.append(cs)
    errors.append(i)
```

```
In [7]: plt.plot(errors, values, marker='o')
        plt.title("Accuracy vs error level")
        plt.xlabel("Error level (years)")
        plt.ylabel("Cumulative score");
        plt.grid()
        plt.show()
```

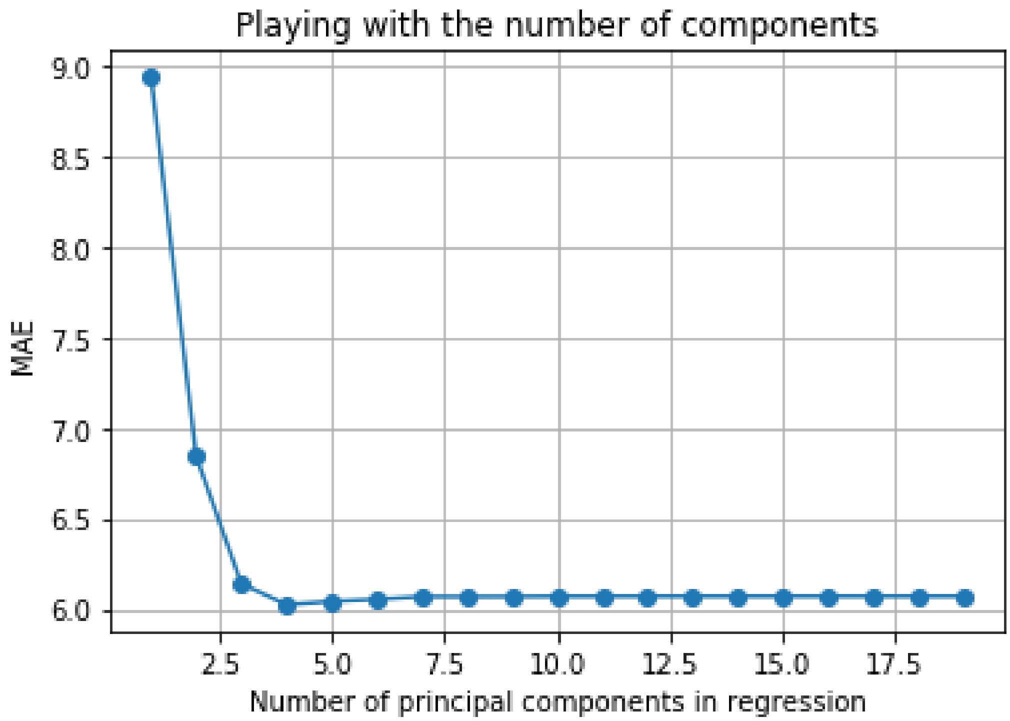


1.3 Regression method: least square regression model with various number of principal components (which will be used as regressors) to identify the best

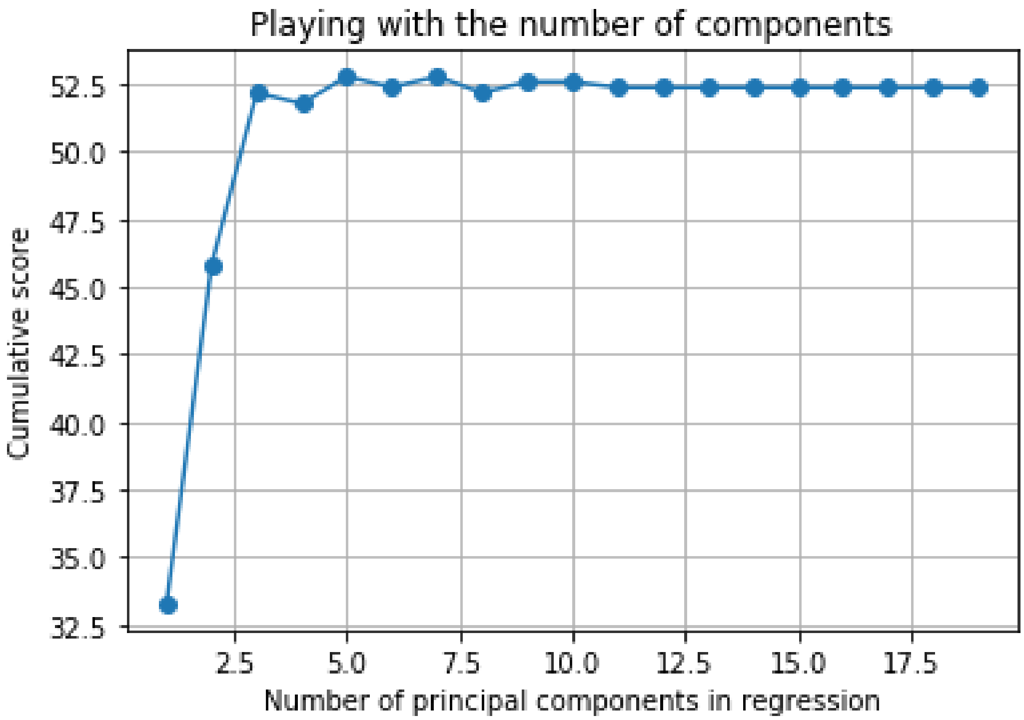
```
In [8]: mae_lsr = []
        cs_lsr = []
        for i in np.arange(1, 20):
            pls = PLSRegression(n_components=i)
            pls.fit(xtrain, ytrain)
            yhat_test_lsr = pls.predict(xtest)
            mae_lsr.append(mean_absolute_error(ytest, yhat_test_lsr))
            a = abs(ytest[:,0] - yhat_test_lsr[:,0])
            cs_lsr.append(len(np.where(a <= err_level)[0])/len(a)*100)

        # Plot results
        plt.plot(np.arange(1, 20), np.array(mae_lsr), marker='o')
        plt.title('Playing with the number of components')
        plt.xlabel('Number of principal components in regression')
```

```
plt.ylabel('MAE')
plt.grid()
plt.show()
```




```
In [9]: plt.plot(np.arange(1, 20), np.array(cs_lsr), marker='o')
        plt.title("Playing with the number of components")
        plt.xlabel("Number of principal components in regression")
        plt.ylabel("Cumulative score");
        plt.grid()
        plt.show()
```




```
In [10]: # Thus, 4 - 5 components seem to be optimal parameters.
         print("The best MAE: %s with the number of parameters %s " %(min(mae_lsr), mae_lsr.inde
         print("Corresponding CS: %s " %(cs_lsr[mae_lsr.index(min(mae_lsr))]))
```

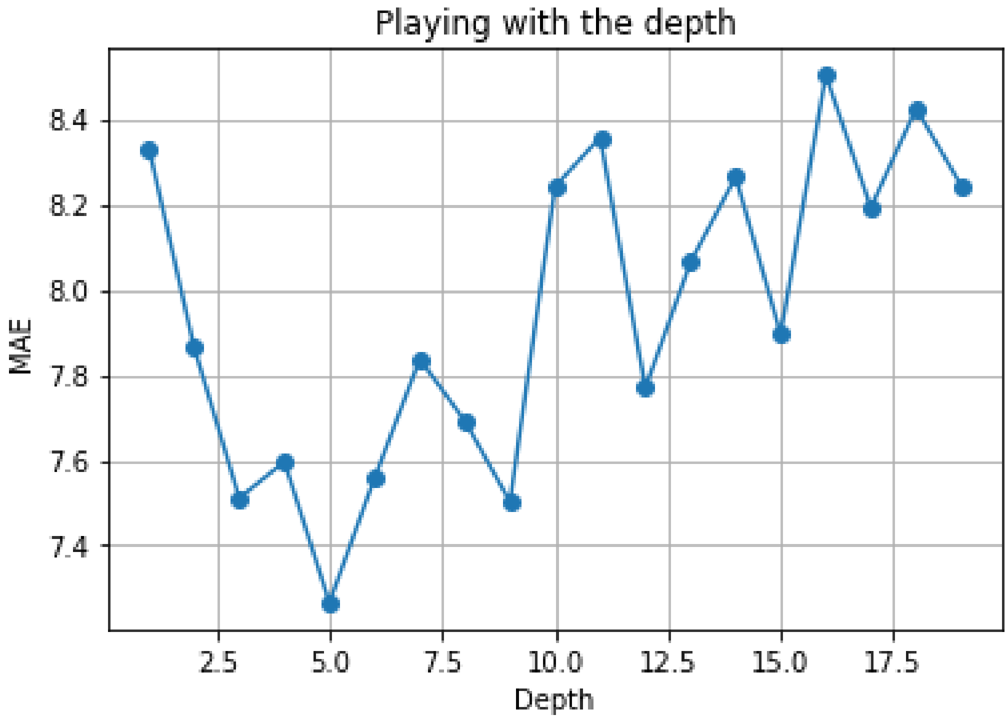
```
The best MAE: 6.02451098497 with the number of parameters 4
Corresponding CS: 51.79282868525896
```

1.4 Regression method: regression tree

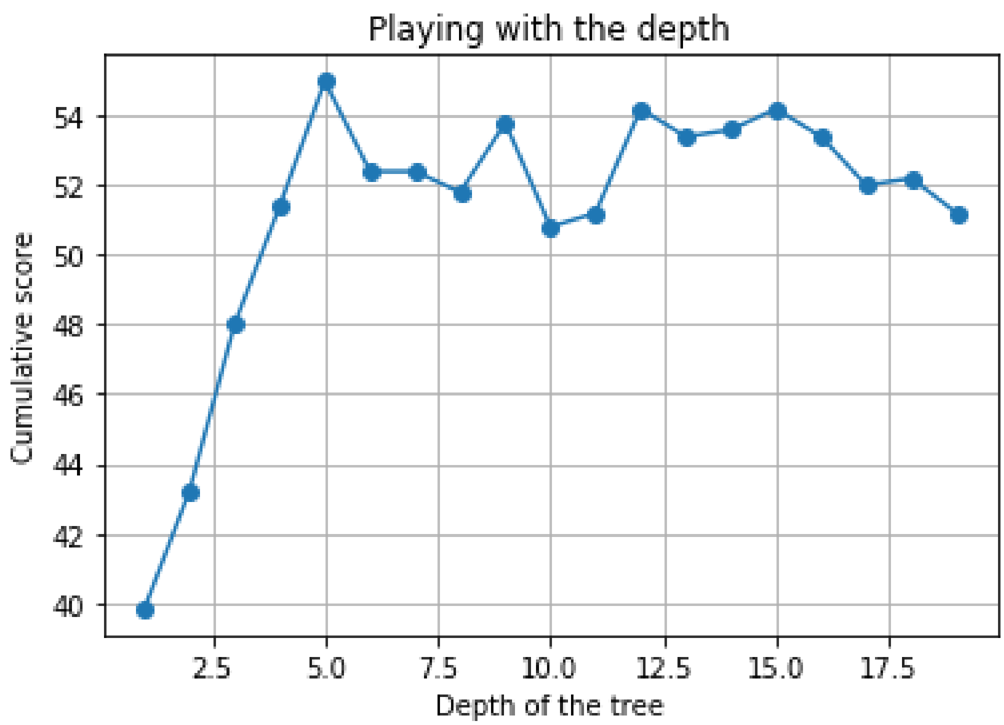
```
In [11]: mae_rt = []
         cs_rt = []
         for depth in np.arange(1, 20):
             regr = DecisionTreeRegressor(max_depth = depth)
             regr.fit(xtrain, ytrain)
             yhat_test_rt = regr.predict(xtest)
             mae_rt.append(mean_absolute_error(ytest, yhat_test_rt))
             a = abs(ytest[:,0] - yhat_test_rt)
             cs_rt.append(len(np.where(a <= err_level)[0])/len(a) *100)
```

```
In [12]: # Plot results
         plt.plot(np.arange(1, 20), np.array(mae_rt), marker='o')
         plt.title('Playing with the depth')
         plt.xlabel('Depth')
```

```
plt.ylabel('MAE')
plt.grid()
plt.show()
```




```
In [13]: plt.plot(np.arange(1, 20), np.array(cs_rt), marker='o')
         plt.title("Playing with the depth")
         plt.xlabel("Depth of the tree")
         plt.ylabel("Cumulative score");
         plt.grid()
         plt.show()
```




```
In [14]: # Thus, 5 depth seem to be optimal parameter.
         print("The best MAE: %s with the depth %s " %(min(mae_rt), mae_rt.index(min(mae_rt)) +
         print("Corresponding CS: %s " %(cs_rt[mae_rt.index(min(mae_rt))]))
```

```
The best MAE: 7.26608362676 with the depth 5
Corresponding CS: 54.980079681274894
```

1.5 Computing the MAE and CS value (with cumulative error level of 5) for Support Vector Regression

C adjusts how hard or soft a large margin classification should be. C parameter tells the SVM optimization how much we want to avoid misclassifying each training example. Parameters: kernel and C are selected using Grid search approach. The kernel coefficient gamma is 1/n_features by default.

```
In [15]: # Set the parameters to try
         tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                              'C': [1, 10, 100, 1000]},
                             {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
         scores = ['precision', 'recall']

         for score in scores:
             print("# Tuning hyper-parameters for %s" % score)
```

```
            svr = svm.SVC()
            clf = GridSearchCV(svr, tuned_parameters, cv=5, scoring='%s_macro' % score)
            clf.fit(xtrain, ytrain)

            print("\nBest parameters set found on development set: %r" % (clf.best_params_))
            print("Grid scores on development set:\n")

            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))
            print("\n Detailed classification report:\n")
            print("The model is trained on the full development set.")
            print("The scores are computed on the full evaluation set.\n")

            y_pred = clf.predict(xtest)
            print(classification_report(ytest, y_pred))
            print()
```

```
# Tuning hyper-parameters for precision

Best parameters set found on development set: {'C': 1, 'kernel': 'linear'}
Grid scores on development set:

0.001 (+/-0.001) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.001 (+/-0.001) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.001 (+/-0.001) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.001 (+/-0.001) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.028 (+/-0.013) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.001 (+/-0.001) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.054 (+/-0.051) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.030 (+/-0.018) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.057 (+/-0.046) for {'C': 1, 'kernel': 'linear'}
0.050 (+/-0.051) for {'C': 10, 'kernel': 'linear'}
0.050 (+/-0.051) for {'C': 100, 'kernel': 'linear'}
0.050 (+/-0.051) for {'C': 1000, 'kernel': 'linear'}

 Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

          0       0.57      0.50      0.53        26
          1       0.11      0.12      0.11        17
          2       0.09      0.04      0.05        27
          3       0.17      0.15      0.16        26
```

4	0.20	0.10	0.14	29
5	0.06	0.10	0.07	20
6	0.06	0.03	0.04	31
7	0.06	0.09	0.07	22
8	0.22	0.10	0.13	21
9	0.33	0.06	0.10	17
10	0.00	0.00	0.00	28
11	0.00	0.00	0.00	18
12	0.07	0.04	0.05	24
13	0.00	0.00	0.00	19
14	0.07	0.06	0.06	17
15	0.09	0.06	0.07	17
16	0.00	0.00	0.00	14
17	0.00	0.00	0.00	16
18	0.07	0.18	0.10	17
19	0.10	0.11	0.11	9
20	0.00	0.00	0.00	8
21	0.00	0.00	0.00	5
22	0.00	0.00	0.00	5
23	0.00	0.00	0.00	7
24	0.00	0.00	0.00	4
25	0.00	0.00	0.00	7
26	0.00	0.00	0.00	2
27	0.00	0.00	0.00	6
28	0.00	0.00	0.00	3
29	0.00	0.00	0.00	2
30	0.00	0.00	0.00	4
31	0.00	0.00	0.00	1
32	0.00	0.00	0.00	2
33	0.00	0.00	0.00	4
34	0.00	0.00	0.00	1
35	0.00	0.00	0.00	2
36	0.00	0.00	0.00	0
37	0.00	0.00	0.00	0
38	0.00	0.00	0.00	1
39	0.00	0.00	0.00	1
40	0.00	0.00	0.00	2
41	0.00	0.00	0.00	2
42	0.00	0.00	0.00	1
43	0.00	0.00	0.00	2
44	0.17	1.00	0.29	1
45	0.00	0.00	0.00	4
46	0.00	0.00	0.00	1
47	0.00	0.00	0.00	1
48	0.00	0.00	0.00	1
49	0.00	0.00	0.00	1
50	0.00	0.00	0.00	1
52	0.00	0.00	0.00	1

53	0.00	0.00	0.00	1
54	0.00	0.00	0.00	2
55	0.00	0.00	0.00	1
avg / total	0.10	0.08	0.08	502

Tuning hyper-parameters for recall

Best parameters set found on development set: {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
Grid scores on development set:

0.021 (+/-0.007) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.021 (+/-0.007) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.021 (+/-0.007) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.021 (+/-0.007) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.043 (+/-0.018) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.021 (+/-0.007) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.053 (+/-0.028) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.045 (+/-0.022) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.051 (+/-0.024) for {'C': 1, 'kernel': 'linear'}
0.049 (+/-0.031) for {'C': 10, 'kernel': 'linear'}
0.049 (+/-0.031) for {'C': 100, 'kernel': 'linear'}
0.049 (+/-0.031) for {'C': 1000, 'kernel': 'linear'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

	precision	recall	f1-score	support
0	0.62	0.50	0.55	26
1	0.17	0.18	0.17	17
2	0.09	0.04	0.05	27
3	0.17	0.15	0.16	26
4	0.21	0.14	0.17	29
5	0.06	0.10	0.08	20
6	0.10	0.06	0.08	31
7	0.08	0.14	0.10	22
8	0.29	0.10	0.14	21
9	0.25	0.06	0.10	17
10	0.00	0.00	0.00	28
11	0.00	0.00	0.00	18
12	0.06	0.04	0.05	24
13	0.00	0.00	0.00	19
14	0.08	0.06	0.07	17
15	0.00	0.00	0.00	17

```
          16       0.00      0.00      0.00        14
          17       0.00      0.00      0.00        16
          18       0.12      0.24      0.16        17
          19       0.14      0.11      0.12         9
          20       0.00      0.00      0.00         8
          21       0.00      0.00      0.00         5
          22       0.00      0.00      0.00         5
          23       0.00      0.00      0.00         7
          24       0.00      0.00      0.00         4
          25       0.00      0.00      0.00         7
          26       0.00      0.00      0.00         2
          27       0.00      0.00      0.00         6
          28       0.00      0.00      0.00         3
          29       0.00      0.00      0.00         2
          30       0.00      0.00      0.00         4
          31       0.00      0.00      0.00         1
          32       0.00      0.00      0.00         2
          33       0.00      0.00      0.00         4
          34       0.00      0.00      0.00         1
          35       0.00      0.00      0.00         2
          36       0.00      0.00      0.00         0
          37       0.00      0.00      0.00         0
          38       0.00      0.00      0.00         1
          39       0.00      0.00      0.00         1
          40       0.00      0.00      0.00         2
          41       0.00      0.00      0.00         2
          42       0.00      0.00      0.00         1
          43       0.00      0.00      0.00         2
          44       0.17      1.00      0.29         1
          45       0.00      0.00      0.00         4
          46       0.00      0.00      0.00         1
          47       0.00      0.00      0.00         1
          48       0.00      0.00      0.00         1
          49       0.00      0.00      0.00         1
          50       0.00      0.00      0.00         1
          52       0.00      0.00      0.00         1
          53       0.00      0.00      0.00         1
          54       0.00      0.00      0.00         2
          55       0.00      0.00      0.00         1

avg / total       0.11      0.09      0.09       502
```

In [16]:
```python
#Training with the best parameters: {'C': 1, 'kernel': 'linear'}
clf = SVR(C = 1.0, kernel='linear')
clf.fit(xtrain, ytrain)
```

```
yhat_test_clf = clf.predict(xtest)
```

1.6 Computing the MAE and CS value (with cumulative error level of 5) for Support Vector Regression

```
In [17]: print("MAE: %s" %(mean_absolute_error(ytest, yhat_test_clf)))
         a =  abs(ytest[:,0] - yhat_test_clf)
         print("CS value: %s" %(len(np.where(a <= err_level)[0])/len(a) *100))
```

```
MAE: 5.62386672972
CS value: 56.573705179282875
```

```
In [18]: from texttable import Texttable
         t = Texttable()
         t.add_rows([['Algorithm','MAE', 'CS'], ['Linear regression', 7.704, 41.235], ['PLS regr
                     ['Regression tree', 6.969, 53.386], ['Support vector regression', 5.624, 56
         print(t.draw())
```

```
+---------------------------+-------+--------+
|         Algorithm         |  MAE  |   CS   |
+===========================+=======+========+
| Linear regression         | 7.704 | 41.235 |
+---------------------------+-------+--------+
| PLS regression            | 6.025 | 51.793 |
+---------------------------+-------+--------+
| Regression tree           | 6.969 | 53.386 |
+---------------------------+-------+--------+
| Support vector regression | 5.624 | 56.574 |
+---------------------------+-------+--------+
```

To sum up all experiments results, SVR seem to perform better in estimations the facial age.