- 论文名：Classification of functional data: A segmentation approach
- 作者：Bin Li ∗, Qingzhao Yu
- Louisiana State University, 70803 Baton Rouge, LA, United States

1 论文概况

主要方法：

- FSDA(functional segment discriminant analtsis) 函数段判别分析——主要是将LDA和SVM相结合。

优点：

- 对不规则的功能数据特别有用(spatial heterogeneity"空间异构性" and spikes)
- fraction of the spectrum 减少计算量
- 可以识别重要的预测值和提取特征
- 灵活——易于合并来自其他数据源的信息或来自调查人员的先验知识

1.1 方法简介

LDA（Linear discriminant analysis）：

- 对非正态分布以及不同类别的协方差相当**稳健**
- 对高度非线性的类边界过于严格
- 对FDA中常出现的许多高度相关的预测因子这种情况，他太灵活了

SVM：

- 很容易扩展到非线性空间
- 由于一视同仁的利用所有变量，SVM存在冗余变量

FSDA（本文的方法）

- F统计量找marker,将数据切成短曲线，进行处理，方便计算与储存
- LDA作数据简化、降维(data reduction), SVM作分类器
- 对不规则的函数型数据和没有明显局部特征的数据都有很好的分类效果

1.2 论文内容(phoneme为例)

- five-fold cross-validation
- 在phoneme的例子中，随机地选取1000个样本作为训练集，剩余3509个样本作为测试集。
- 作为一个多类分类问题，本例中提取的特征采用前三个线性判别变量，即$l = 3$，因为达到了最佳的预测性能。
- 在每次复制中，随机分配的训练集用于拟合模型，其中性能在测试集上进行评估。
- 在音素的例子中，FSDA在45次试验中是最好的，平均每一次试验只比最好的高0.22%。
- 通过Wilcoxon符号秩检验(三个p值均小于0.0001)，fsda与其他三个竞争者之间的差异在统计学上也是显著的。

1.2.1 一些参数

- 交叉验证：five_fold cross-validation
- LDA中采取3个线性判别变量，$l=3$
- SVM使用高斯核函数，$kernel='rbf'$
- 函数段半径取6，marker的个数取7.即$h=6, m=7$

2 代码实现

2.1 Data Split

In [1]:

```
import math
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn import svm
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score

df = pd.read_csv('F:\\Data and code\\data\\FSDA\\phoneme.data',index_col = 0)
```

In [2]:

```
pd.set_option('display.max_rows',10)
```

In [3]:

```
df
```

Out[3]:

row.names	x.1	x.2	x.3	x.4	x.5	x.6	x.7	x.8	
1	9.85770	9.20711	9.81689	9.01692	9.05675	8.92518	11.28308	11.52980	1
2	13.23079	14.19189	15.34428	18.11737	19.53875	18.32726	17.34169	17.16861	1
3	10.81889	9.07615	9.77940	12.20135	12.59005	10.53364	8.54693	9.46049	1
4	10.53679	9.12147	10.84621	13.92331	13.52476	10.27831	8.97459	11.57109	1
5	12.96705	13.69454	14.91182	18.22292	18.45390	17.25760	17.79614	17.76387	1
...	...	...	...	...	...	...	...	...	
4505	11.38394	10.21040	16.32658	18.30125	16.91804	10.90029	17.10393	19.37741	1
4506	12.01774	11.86761	16.34707	18.05214	15.97599	12.86022	16.83436	18.38985	1
4507	12.30174	12.40383	9.06497	12.43750	13.48388	13.52034	12.97796	11.58782	1
4508	8.39388	9.84770	16.24534	17.35311	14.80537	12.72429	17.01145	17.54733	1
4509	8.14032	9.93753	16.30187	17.31425	14.40116	13.52353	16.85938	17.14016	1

4509 rows × 258 columns

In [4]:

```
# 按数据集中已经划分好的train和test进行划分
Xtrain = df.iloc[:3340,:-2]
Ytrain = df.iloc[:3340,-2]

Xtest = df.iloc[3340:,:-2]
Ytest = df.iloc[3340:,-2]
```

2.2 F统计量

- 通过F统计量选出m个marker，以及对应的函数段
- 这里使用论文中交叉验证后的参数，m为7，h为6
- 选出F统计量最大的m=7个值，以及对应的m=7个段，再把这7个段放入一个list——features

In [5]:

```
# 按组把训练集划分出5个对应输出['ao', 'aa', 'sh', 'iy', 'dcl']

Xtrain_ao, Ytrain_ao = Xtrain.loc[Ytrain == 'ao', :], Ytrain.loc[Ytrain == 'ao']
Xtrain_aa, Ytrain_aa = Xtrain.loc[Ytrain == 'aa', :], Ytrain.loc[Ytrain == 'aa']
Xtrain_sh, Ytrain_sh = Xtrain.loc[Ytrain == 'sh', :], Ytrain.loc[Ytrain == 'sh']
Xtrain_iy, Ytrain_iy = Xtrain.loc[Ytrain == 'iy', :], Ytrain.loc[Ytrain == 'iy']
Xtrain_dcl, Ytrain_dcl = Xtrain.loc[Ytrain == 'dcl', :], Ytrain.loc[Ytrain == 'dcl']

Xtrain_phoneme = [Xtrain_ao, Xtrain_aa, Xtrain_sh, Xtrain_iy, Xtrain_dcl]
```

In [6]:

```
# 定义F统计量、并将256个变量的F统计量存在一个数组F_all里

def F(marker, Xtrain, Xtrain_phoneme,K = 5,  n=len(Xtrain)):

    up = 0
    down = 0

    for k in range(K):

        X_marker_bar = Xtrain.mean()[marker] # X_j这里的marker从0开始
        X_marker_k_bar = Xtrain_phoneme[k].mean()[marker] # X_jk
        X_marker_in = Xtrain_phoneme[k].iloc[:, marker] # x_ij

        up += (len(Xtrain_phoneme[k]) * math.pow((X_marker_k_bar - X_marker_bar), 2) )/ (K - 1
        down += (sum((X_marker_in - X_marker_k_bar) * (X_marker_in - X_marker_k_bar)) )/ (n -

    F_marker = up / down

    return F_marker



F_all = np.zeros(256)
for i in range(256):
    F_all[i] = F(i, Xtrain, Xtrain_phoneme)
```

In [7]:

```
'''
    输出最大的m=7个值得索引,segment长度， h=6

'''
'''
    def findmarkers(m,h,F):
        markers=np.zeros(m)
        for i in range(m):
            markers[i]=np.argmax(F)
            F[max(np.argmax(F)-h,0):min(np.argmax(F)+h,len(F)-1)]=0
        return markers
    m=7
    h=6
    markers=findmarkers(m,h,F)
    print(markers)

'''
m, h = 7, 6
S = np.zeros(m)
F_temp = F_all.copy()

for i in range(m):
    S[i] = F_temp.argmax().copy()

    for j in range(h):

        if S[i] < h:
            F_temp[int(S[i]+j)] = 0
            F_temp[:int(S[i])] = 0
        elif S[i] > (len(F_temp) - h):
            F_temp[int(S[i]):] = 0
            F_temp[int(S[i]-j)] = 0
        else:
            F_temp[int(S[i]+j)] = 0
            F_temp[int(S[i]-j)] = 0

S
```

Out[7]:

```
array([ 29.,  22.,  35.,  16.,   3.,  41., 107.])
```

可以看到marker值多在低频率区域

included in the curve segments. Shown in the right panel of Fig. 2, we see these regions do have a good separation of two classes. In the phoneme example, we see that the channels selected most are in the low frequency region. This agrees with the fact that the signals tend to stabilize in the high frequency region.

In [8]:

```
# 这里将所有marker对应的函数段都找到，输入到一个列表features中

'''
    #LDA降维，得到3m个线性判别变量
    lda = LinearDiscriminantAnalysis(n_components=3)
    lda.fit(X_train[:,int(max(markers[0]-h,0)):int(min(markers[0]+h,X_train.shape[1]-1))],Y_tra
    X_train_ldv = lda.transform(X_train[:,int(max(markers[0]-h,0)):int(min(markers[0]+h,X_train
    X_test_ldv = lda.transform(X_test[:,int(max(markers[0]-h,0)):int(min(markers[0]+h,X_test.sh
    for i in range(1,m):
        lda.fit(X_train[:,int(max(markers[i]-h,0)):int(min(markers[i]+h,X_train.shape[1]-1))],Y
        X_train_new = lda.transform(X_train[:,int(max(markers[i]-h,0)):int(min(markers[i]+h,X_t
        X_test_new = lda.transform(X_test[:,int(max(markers[i]-h,0)):int(min(markers[i]+h,X_tes
        X_train_ldv=np.hstack((X_train_ldv,X_train_new))
        X_test_ldv=np.hstack((X_test_ldv,X_test_new))
    #print(X_ldv)
    #print(np.shape(X_ldv))

'''
features = []
for i in range(m):

    if S[i] < h:
        temp = np.arange(int(S[i]+h+1)) # 索引左开右闭，加一
    elif S[i] > (len(F_temp) - h):
        temp = np.arange(int(S[i]-h),len(F_all)+1)
    else:
        temp = np.arange(int(S[i]-h),int(S[i]+h+1))

    features.append(temp)

features
```

Out[8]:

```
[array([23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]),
 array([16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]),
 array([29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41]),
 array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]),
 array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 array([35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]),
 array([101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113])]
```

2.3 放入LDA

- 采用前三个线性判别变量——n_components=3

In [9]:

```
'''
    输入:
        训练集/测试集
        一个features列表【m=7,h=6】
        m和n

    输出:
        经过LDA压缩处理过后的3*m * n 的自变量(train or test)X_all_LDA, 和对应的标签Y
'''

lda = LinearDiscriminantAnalysis(n_components=3)
Xtrain_lda = np.zeros((len(Xtrain),m*3))
Xtest_lda = np.zeros((len(Xtest),m*3))

for i in range(m):
    X_lda1 = lda.fit(Xtrain.iloc[:,features[i]], Ytrain).transform(Xtrain.iloc[:,features[i]])
    Xtrain_lda[:,i*3:(i+1)*3] = X_lda1
    X_lda2 = lda.fit(Xtrain.iloc[:,features[i]], Ytrain).transform(Xtest.iloc[:,features[i]])
    Xtest_lda[:,i*3:(i+1)*3] = X_lda2
```

2.4 放入SVM,得出准确率

- 使用高斯核函数——kernel='rbf'

In [10]:

```
clf = svm.SVC(decision_function_shape='ovo',kernel = 'rbf')
clf.fit(Xtrain_lda, Ytrain)

# acc_train = accuracy_score(Ytrain,clf.predict(Xtrain_lda))
# acc_test = accuracy_score(Ytest, clf.predict(Xtest_lda))
acc_train = clf.score(Xtrain_lda,Ytrain)
acc_test = clf.score(Xtest_lda,Ytest)

print(
    '训练集准确率为:%.10f'%(acc_train)
)
print(
    '测试集准确率为:%.10f'%(acc_test)
)
```

训练集准确率为:0.9365269461
测试集准确率为:0.9144568007

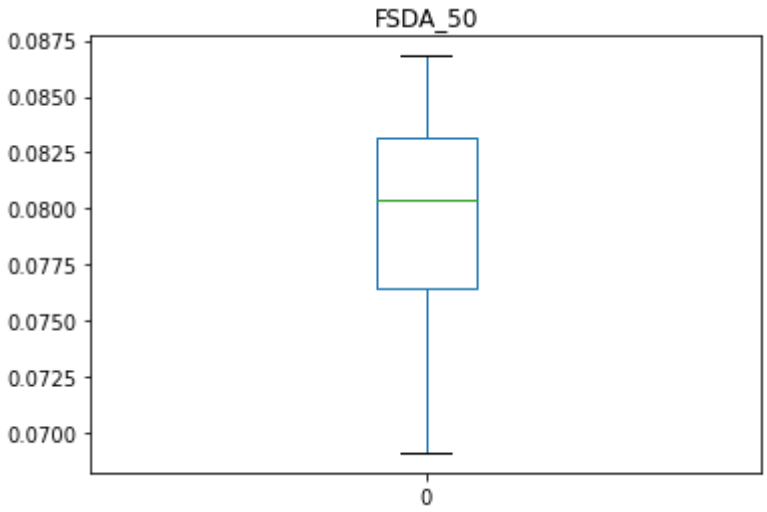
2.5 按论文中随机划分50次训练集，每次1000个训练集，剩下的测试集

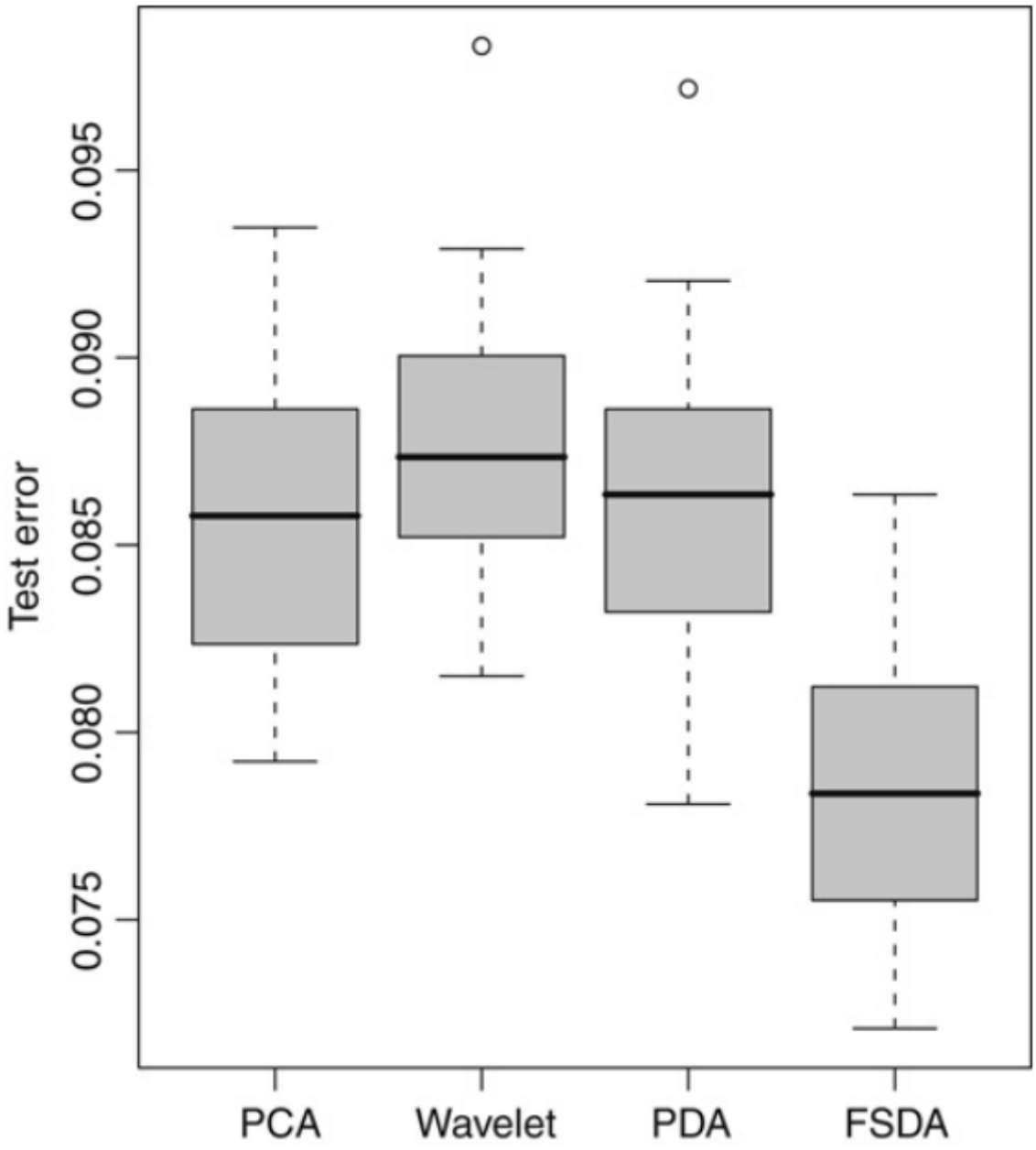
In [11]:

```
FSDA = [0.91671278, 0.91796009, 0.92874896, 0.92770083, 0.91650485, 0.91775493
, 0.92386489, 0.91952434, 0.91466446, 0.9222253, 0.91888151, 0.92118363
, 0.91369295, 0.91793313, 0.91606847, 0.93011647, 0.91781959, 0.92329072
, 0.93087174, 0.92496552, 0.91924779, 0.91682732, 0.91401714, 0.92578233
, 0.91541875, 0.92013219, 0.92744392, 0.91932354, 0.91558262, 0.92141671
, 0.92367257, 0.9199446, 0.92038567, 0.92248276, 0.92595674, 0.91943522
, 0.92277778, 0.9225255, 0.9249241, 0.92305563, 0.91765681, 0.91565265
, 0.91685144, 0.91318772, 0.92643805, 0.91829975, 0.92388597, 0.91678225
, 0.9164823, 0.9197668]
a = np.ones(50)
df = pd.DataFrame(a - FSDA)
df.plot.box(title='FSDA_50')
```

Out[11]:

<AxesSubplot:title={'center':'FSDA_50'}>





In [12]:

```
1 df.describe()
```

Out[12]:

	0
count	50.000000
mean	0.079483
std	0.004451
min	0.069128
25%	0.076423
50%	0.080354
75%	0.083167
max	0.086812