Geoherb identification based on spectroscopic profiling data and NNRW

Test Subjects

Radix astragali (Astragalus Root; Chinese: huang-qi) is a medicinal herb usually used in traditional Chinese medicine for the treatment of conditions such as diabetes and cardiovascular diseases

Data set summary:

Y labels: ["内蒙黄芪","四川黄芪","山西黄芪","甘肃黄芪"]

```
7044.txt - Raman
  X meaning: Raman shift / wave number
  X range: 100 ~ 4278 cm-1
  Resolution: 2cm-1

7143.txt - UV
  X meaning: wave length
  X range: 200 ~ 800 nm
  Resolution: 1 nm
```

the origins of these samples were the Shanxi, Neimenggu, Sichuan, and Gansu provinces of China. The Raman spectrum of each sample was collected by a portable laser Raman spectrometer ProttezRaman-D3 (Enwave Optronics, USA). The excitation wavelength of the laser was 785 nm and the spectral measurements were conducted with an exposure time of 5 s and a laser power of 450 mW. The ultraviolet-visible absorption spectrum of each sample was recorded from a T6 New century ultraviolet-visible spectrometer (Purkinje General Instrument Company Limited, Beijing, China). Sample preparation. Radix astragali samples were added into a high speed multifunction grinder for processing for 5–10 min at 25000 rpm. 3 g of the obtained powder sample was added into 30 mL of ethanol solution, and then the mixture solution was stirred at 100°C for 60 min under reflux condition. Finally, the treated sample was cooled naturally and filtered.

Install library

```
pip install pyNNRW==0.0.3
```

In [15]:
```
from pyNNRW.pyNNRW import *
```

2D scatter plot

In [16]:
```
%run plotComponents2D.py
```

```
<Figure size 432x288 with 0 Axes>
```

Visualize feature importance

In [17]:
```
%run feature_importance.py
```

Load Data

In [4]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

FILES = {}
FILES['RAMAN'] = '7044X_RAMAN.csv'
FILES['UV'] = '7143X_UV.csv'
```

In [5]:
```
DS = {}

for key in FILES:
    DS[key] = pd.read_csv("ds2/" + FILES[key])
    print(key, DS[key].shape)
```

```
RAMAN (160, 2091)
UV (160, 602)
```

Preprocessing

Feature Scaling

Coefficients from LASSO or ElasticNet depend on the magnitude of each variable. It is therefore necessary to rescale, or standardize, the variables.
The result of centering the variables means that there is no longer an intercept.

Without feature scaling, the feature selection result can be quite different!

Feature selection via ElasticNet

1. Reduce dimension and remove unrelavent features for the classification problem
2. reduce the overfitting risk of successive classification model. Improve the generalization
3. By selecting only a few bunch of features, easier to interpret its chemcial meansings. i.e. gain a better understanding of the features and easy to notice the co-ocurrance of multiple variables (important for Raman, as one bond or molecule have muliptle peaks).

Highly recommendated for Raman, as well as other high-dimensional physio-chemical spectroscopic data, such as MALDI-TOF.

Feature Selection for Multi-Class

[Evaluating Feature Selection Methods for Multi-Label Text Classification, by Newton Spolaˆor1 and Grigorios Tsoumakas2]

"Rank features according to the average or the maximum Chi-squared score across all labels, led to most of the best classifiers while using less features."

Data fusion by concatenating selected feature vectors

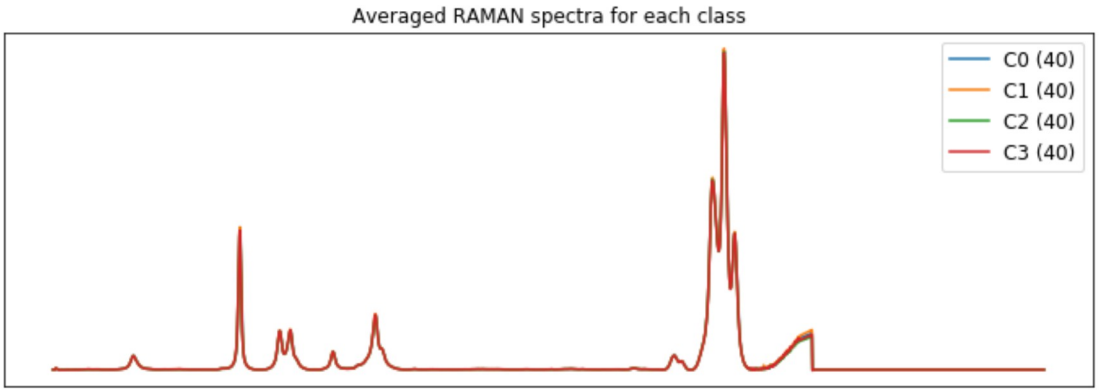
Use `hstack` to get the combined feature vector `Xcmb`

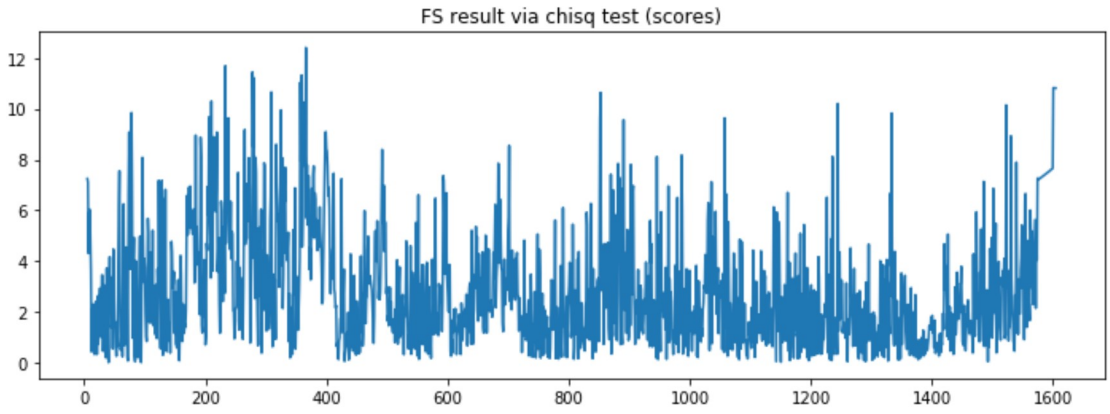
In [6]:

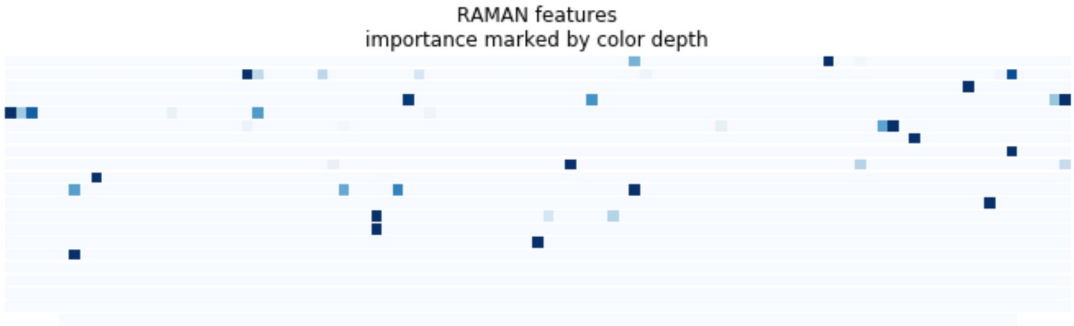
```
dict_metrics = {}
XNAMES = []
XFSNAMES = []
Xcmb = None  # concatenated raw feature vector
ycmb = None
XFS = None # concatenated selected feature vector

for i,key in enumerate(DS):

    print('##########', key, '############')

    ds = DS[key]
    # print(ds.describe())
    cols = ds.shape[1]

    # convert from pandas dataframe to numpy matrices
    X = np.matrix(ds.iloc[:,1:cols].values)
    y = np.array(ds.iloc[:,0].values.ravel(), dtype='int') # first c
ol is y label
    ys = list(map(str, y))
    X_names = list(ds.columns.values[1:])


    ################################################
    ####### Plot Averaged Waveform ######
    ################################################

    plt.figure(figsize = (12,4))

    for c in set(y):
        Xc = X[y == c]
        yc = y[y == c]

        plt.plot(X_names, np.mean(Xc,axis=0).T, label= 'C' + str(c)
+ ' (' + str(len(yc)) + ')')
        plt.legend(fontsize=12)

        cur_axes = plt.gca()
        cur_axes.axes.get_xaxis().set_visible(False) #.set_ticklabel
s([])
        cur_axes.axes.get_yaxis().set_visible(False) #.set_ticklabel
s([])

    plt.title(u'Averaged ' + key + ' spectra for each class')
    # plt.title(key, fontsize = 18)
    plt.show()


    ################################################
    ####### Preprocessing - Scaling ######
    # ! Required for feature selection (compare coefficient)
    ################################################

    from sklearn.preprocessing import StandardScaler, MinMaxScaler

    scaler = StandardScaler()
    scaler.fit(X)
    X_scaled = scaler.transform(X)
```

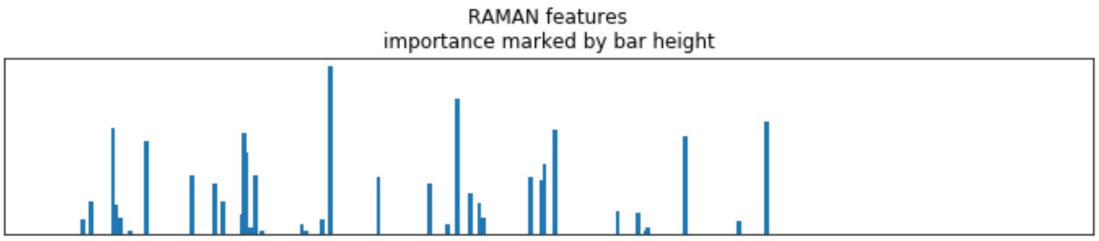
```
    ##############################################
    ####### Feature Selection. ElasticNet ######
    ##############################################
    from sklearn.linear_model import ElasticNetCV
    from sklearn.feature_selection import chi2, SelectKBest

    mm = MinMaxScaler()
    X_mm = mm.fit_transform(X)
    selector = SelectKBest(chi2, k='all')
    selector.fit(X_mm, ys)
    plt.figure(figsize = (12,4))
    plt.title("FS result via chisq test (scores)")
    plt.plot(selector.scores_)
    plt.show()

    elastic_net = ElasticNetCV(cv = 5, tol = 0.005
                               , alphas = [0.1])
                               # , l1_ratio = 1) # if we want more s
parse result
    elastic_net.fit(X_scaled,ys)# NOTE: pass ys not y, to guarantee
it is treated as multi-class, not regression
    N = np.count_nonzero(elastic_net.coef_)

    biggest_elastic_net_fs = (np.argsort(np.abs(elastic_net.coef
_))[-N:])[::-1] # take last N item indices and reverse (ord desc)
    xfs = X_scaled[:,biggest_elastic_net_fs[0:N]] # 前N个系数 non-zer
o
    plot_feature_importance(np.abs(elastic_net.coef_), key + " featu
res")

    XNAMES = XNAMES + ([key + '_' + s for s in X_names])
    xfsnames = np.array(X_names)[biggest_elastic_net_fs]

    if (i == 0):
        Xcmb = X
        ycmb = y
        XFS = xfs
    else:
        Xcmb = np.hstack((Xcmb, X))
        assert np.allclose(ycmb, y)
        XFS = np.hstack((XFS, xfs))

    XFSNAMES = XFSNAMES + ([key + '_' + s for s in xfsnames])

    print("Selected Features (N =", str(N), "): ", xfsnames)
```

########## RAMAN ##########



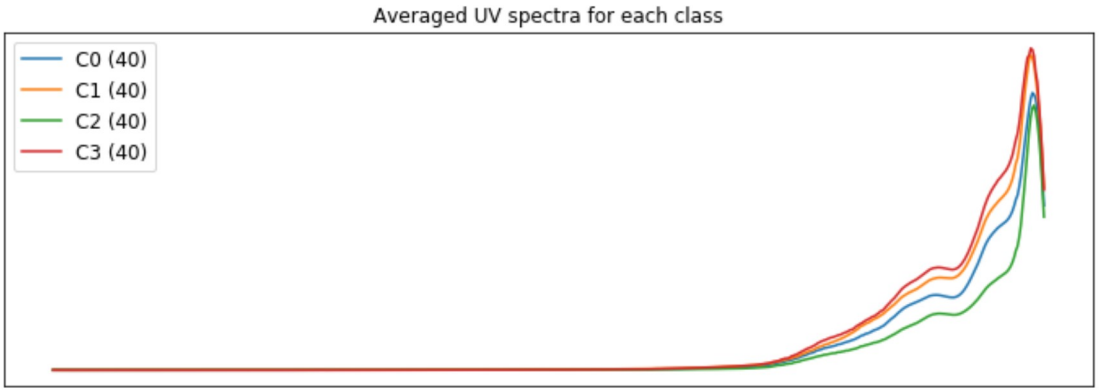




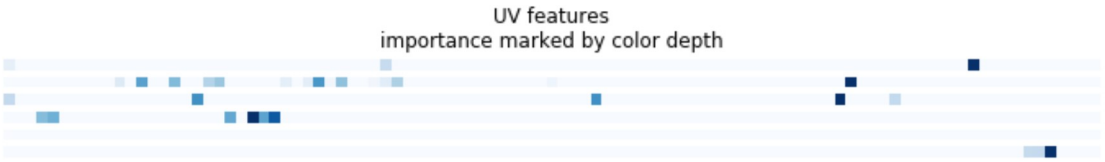


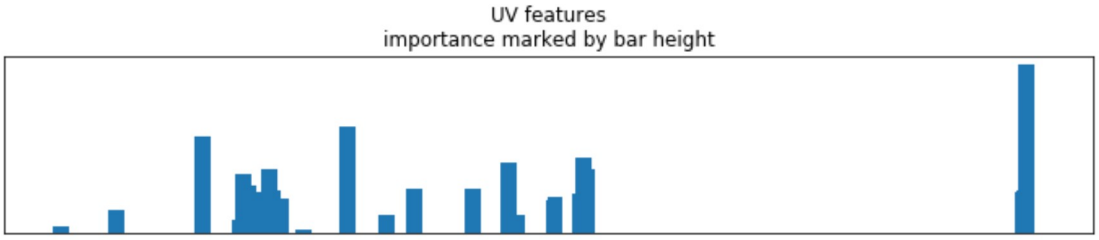
Selected Features (N = 48): ['1264' '1804' '3112' '344' '2216' '900' '2768' '486' '1262' '898' '904'
 '2172' '678' '946' '2112' '1468' '896' '2162' '774' '1686' '1858' '902'
 '808' '252' '1896' '358' '346' '2482' '2568' '894' '376' '1916' '216'
 '1232' '2998' '1144' '1760' '930' '2612' '418' '978' '2600' '484' '1162'
 '258' '906' '1762' '1970']

########## UV ##########




```
C:\Users\eleve\Documents\codex\py\machine learning\source\18. pyNN
RW\feature_importance.py:20: UserWarning: Attempting to set identi
cal left == right == -0.5 results in singular transformations; aut
omatically expanding.
  s = ax.matshow(arr.reshape(1,-1), cmap=plt.cm.Blues)
```






```
Selected Features (N = 32 ):  ['794' '376' '287' '522' '475' '328'
'524' '312' '315' '453' '417' '330'
 '793' '792' '523' '319' '520' '504' '335' '503' '318' '234' '480'
'400'
 '310' '334' '327' '325' '200' '349' '333' '329']
```

The reason we use ElasticNet:

1. elastic net uses L1 regularization, which can produce high level of sparsity. Other FS methods usually produce dense result (e.g., the chisq FS result, as shown above).
2. elastic net combines both L1 and L2 regularization. It is more versatile and flexible than the pure L1 LASSO.

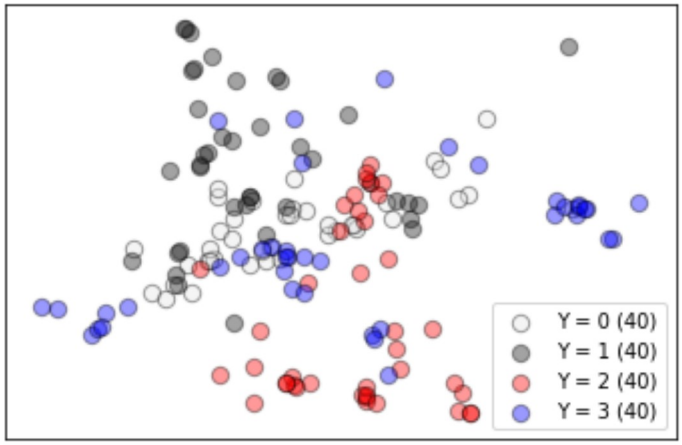
原始拼接向量 concatenated raw feature vector

In [7]:
```
Xcmb.shape, ycmb.shape, len(XNAMES)
```

Out[7]:
```
((160, 2691), (160,), 2691)
```

In [8]:
```
N = Xcmb.shape[1]
from sklearn.decomposition import PCA
Xcmb_pca = PCA(n_components = min(2, N)).fit_transform(Xcmb)
plotComponents2D(Xcmb_pca, ycmb, set(y), ax = None)
plt.legend()
```

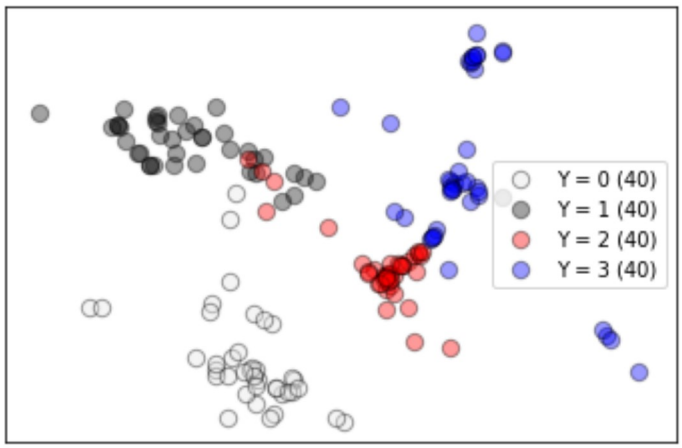
Out[8]:
```
<matplotlib.legend.Legend at 0x1c429a32888>
```



关键特征的拼接向量 concatenated selected feature vector

In [9]:
```
N = XFS.shape[1]
from sklearn.decomposition import PCA
XFS_pca = PCA(n_components = min(2, N)).fit_transform(XFS)
plotComponents2D(XFS_pca, ycmb, set(y), ax = None)
plt.legend()
```

Out[9]:
```
<matplotlib.legend.Legend at 0x1c4315dca08>
```



Persist the intermediate objects

dic = {} dic['Xcmb'] = Xcmb dic['Xcmb_names'] = XNAMES dic['Xfs'] = XFS dic['Xfs_names'] = XFSNAMES dic['y'] = ycmb import pickle with open('fusion_data.pkl', 'wb') as f: pickle.dump(dic, f)

直接加载预处理后的数据

In [10]:
```
import pickle
with open('fusion_data.pkl', 'rb') as f:
    dic = pickle.load(f)

dic.keys()
```

Out[10]:
```
dict_keys(['Xcmb', 'Xcmb_names', 'Xfs', 'Xfs_names', 'y'])
```

In [11]:
```
Xcmb = dic['Xcmb']
XNAMES = dic['Xcmb_names']
XFS = dic['Xfs']
XFSNAMES = dic['Xfs_names']
ycmb = dic['y']
```

Performance comparison. Define a wrapper (use callback function parameter), which calls each specific classifier

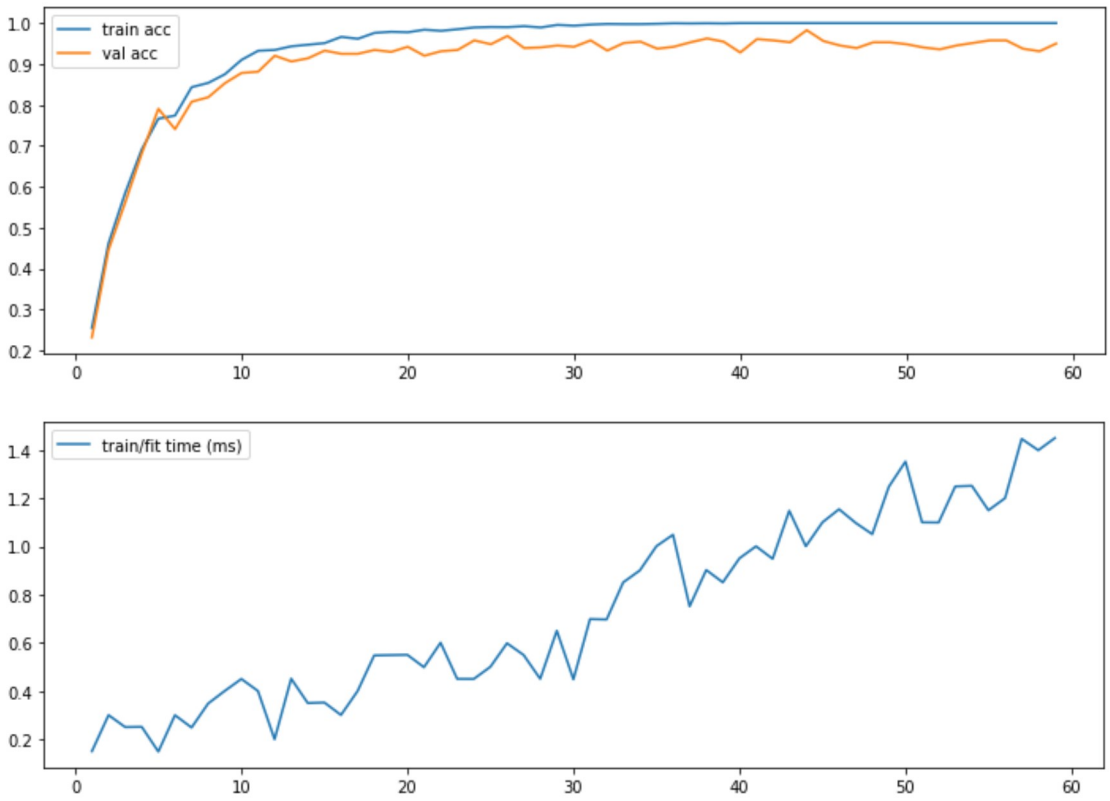
Due to inherent randomness (e.g., random initialization), some algorithms (e.g., NNRW, MLP) have slightly different results for each run.

A. NNRW on the combined feature

Refer to: `py/keras/1. ANN/ELM.ipynb`

In [15]:
```
MTacc, MVacc, MT = PerformenceTests(ELMClf, XFS, ycmb, Ls = list(range(1, 60)))
```

```
C:\Users\eleve\Documents\codex\py\machine learning\source\18. NNRW\ELM.py:96: RuntimeWarning: invalid value encountered in long_scalars
  precision.append(tp / tp_fp)
```



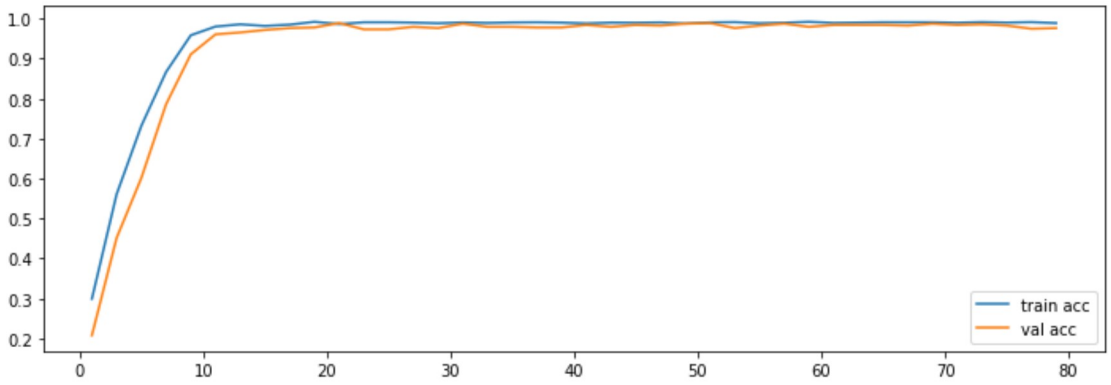
In [16]:
```
IDX = 30
print('Mean Train Acc = ', MTacc[IDX], ' Mean Test Acc = ', MVacc[IDX], ' Mean Consumed Time = ', MT[IDX])
```

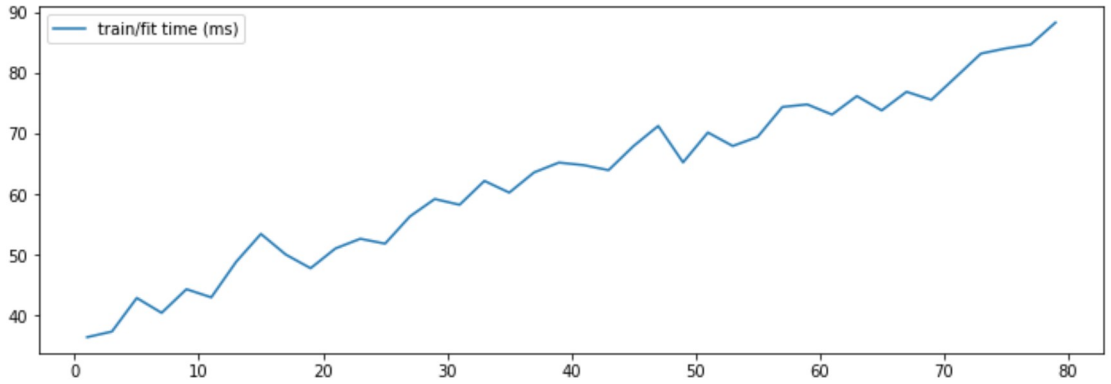
```
Mean Train Acc =  0.996484375  Mean Test Acc =  0.9578125  Mean Consumed Time =  0.69869
```

NNRW flavor 2: RVFL MTacc, MVacc, MT = PerformenceTests(RVFLClf, XFS, ycmb, Ls = list(range(1,10)))
use more time but has slightly higher acc than ELM

B. MLP

In [21]:
```
MTacc, MVacc, MT = PerformenceTests(MLPClf, XFS, ycmb, Ls = list(ran
ge(1, 80, 2)))
```



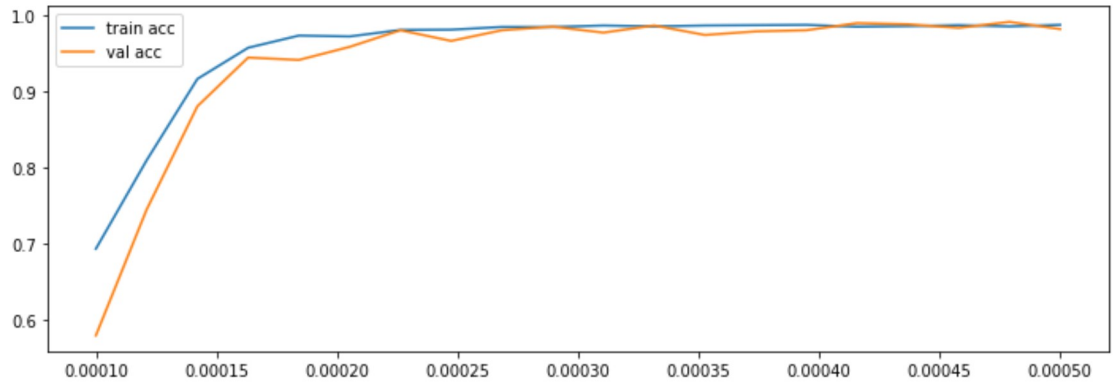


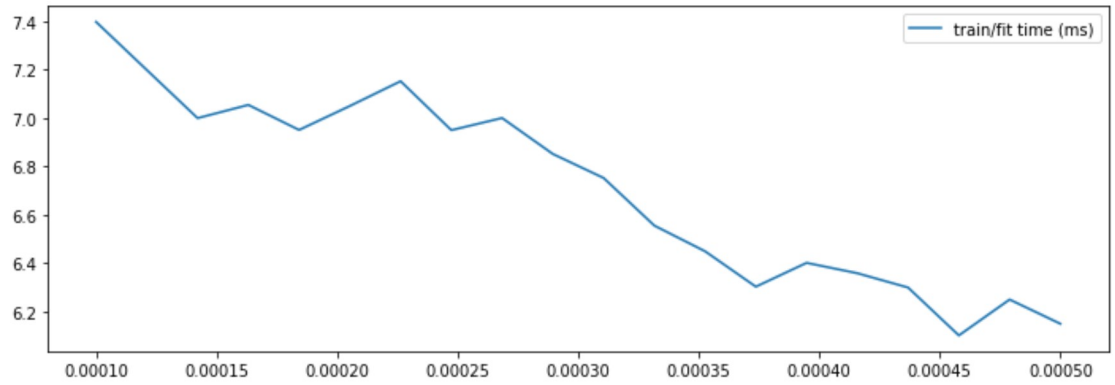
In [22]:
```
IDX = 20
print('Mean Train Acc = ', MTacc[IDX], ' Mean Test Acc = ', MVacc[ID
X], ' Mean Consumed Time = ', MT[IDX])
```

```
Mean Train Acc =  0.98828125  Mean Test Acc =  0.984375  Mean Cons
umed Time =  64.77451500000001
```

C. rbf-SVM

In [23]:
```
MTacc, MVacc, MT = PerformenceTests(SVMClf, XFS, ycmb, Ls = np.linspace(0.0001,0.0005,20).tolist())
print("gamma = 1/(2σ^2)")
```





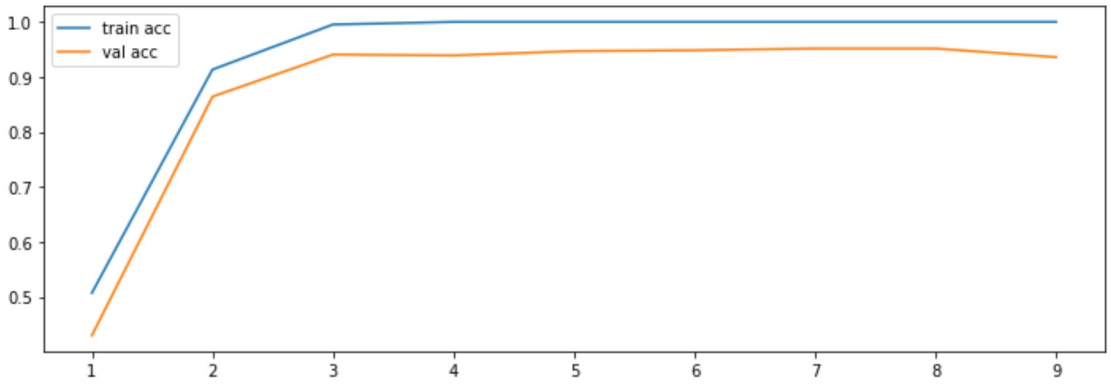

```
gamma = 1/(2σ^2)
```

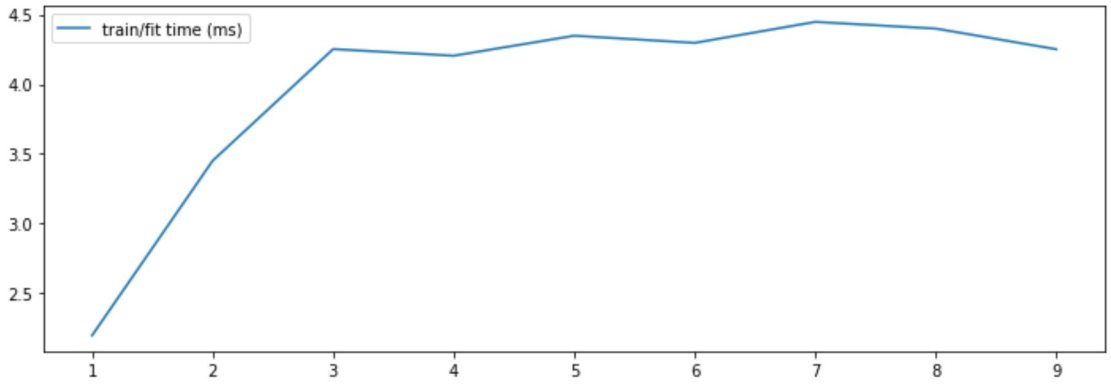
In [24]:
```
IDX = 10
print('Mean Train Acc = ', MTacc[IDX], ' Mean Test Acc = ', MVacc[IDX], ' Mean Consumed Time = ', MT[IDX])
```

```
Mean Train Acc =  0.9875  Mean Test Acc =  0.978125  Mean Consumed Time =  6.752005000000001
```

D. Decision Tree

In [25]:
```
MTacc, MVacc, MT = PerformenceTests(TreeClf, XFS, ys, Ls = list(rang
e(1,10))) # here we use ys to gurantee it is multi-class, not regres
sion
```





In [26]:
```
IDX = 2  # best depth 3
print('Mean Train Acc = ', MTacc[IDX], ' Mean Test Acc = ', MVacc[ID
X], ' Mean Consumed Time = ', MT[IDX])
```

```
Mean Train Acc =  0.9953125  Mean Test Acc =  0.940625  Mean Consu
med Time =  4.25175
```

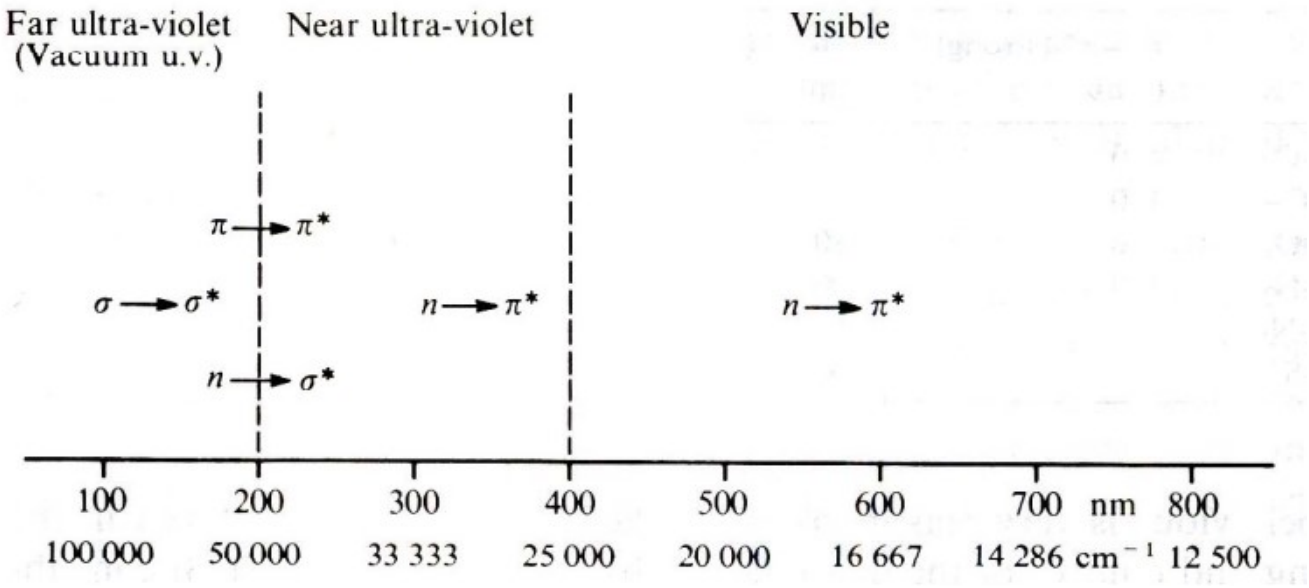
Appendix

Raman Interpretation

Wavenumber (cm-1)	Raman Shift Interpretation
1748	C=O伸缩振动，主要可能源自于脂肪有关的酯基
1663	C=O伸缩振动和C=C伸缩振动，其中C=O伸缩振动可能主要源自于蛋白质的酰胺I键，C=C伸缩振动主要源自于不饱和脂肪酸
1465	CH2变形振动，可能主要源自于糖类和脂肪分子
1337	糖类的C-H变形振动、C-O伸缩振动或/和游离胆固醇的C-C伸缩振动
1304/1260	糖类以及饱和脂肪酸的CH2扭曲振动
1130	饱和脂肪酸的C-C伸缩振动或/和糖类的C-C伸缩振动、C-O伸缩振动以及C-O-H变形振动
1080	游离胆固醇的C-C伸缩振动或/和糖类的C-C伸缩振动、C-O伸缩振动以及C-O-H变形振动
930	胆固醇或/和糖类的C-O-C变形振动、C-O-H变形振动和C-O伸缩振动
862	糖类的C-C-H变形振动和C-O-C变形振动
777	C-C-O变形振动
719	C-S伸缩振动
652	C-C-O变形振动
591/573	C-C-C变形振动、C-O扭曲振动
518	葡萄糖
484/427/363	C-C-C变形振动、C-O扭曲振动

Reference (Raman Bands) (~/Assets/Raman bands.pdf)

UV Interpretation



Substance	Absorption Peak	Molar Absorption Coefficient
Ethylene(CH2=CH2)	180nm	10000
1.3-butadiene	217nm	21000
Vitamin A	328nm	51000
β-carotene	450nm	140000
Benzene	255nm	180
Naphthalene	286nm	360
Anthracene	375nm	7100
Naphthacene	477nm	110000

* chromophore: the part of a molecule responsible for its colour.. The colour is caused when a molecule absorbs certain wavelengths of visible light.It transmits or reflects only other wavelengths, which causes the colour we see

Reference (Visible and Ultraviolet Spectroscopy) (https://www2.chemistry.msu.edu/faculty/reusch/VirtTxtJml/Spectrpy/UV-Vis/spectrum.htm)

Data Science Package Version Info

In [27]:
```
import sklearn
import numpy
import pandas
import matplotlib
import time

print("numpy " + numpy.__version__)
print("pandas " + pandas.__version__)
print("matplotlib " + matplotlib.__version__)

print("NNRW: self-implementation")
print("MLP: self-implementation")
print("SVM: sklearn " + sklearn.__version__)
print("DTC: sklearn " + sklearn.__version__)
```

```
numpy 1.18.1
pandas 1.0.1
matplotlib 3.1.3
NNRW: self-implementation
MLP: self-implementation
SVM: sklearn 0.22.1
DTC: sklearn 0.22.1
```