LDAimplementation

2019 年 3 月 4 日

1 LDA 実装

```
In [19]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         from sklearn.linear_model import LogisticRegression
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
         from layers.decisionregionplotfunction import plot_decision_regions as pdr
```

```
In [20]: # data loading
         df_wine=pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.dat
         x,y=df_wine.iloc[:,1:].values,df_wine.iloc[:,0].values
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,stratify=y,random_state=0)
```

```
In [21]: # standard scaling
         sc=StandardScaler()
         x_train_std=sc.fit_transform(x_train)
         x_test_std=sc.transform(x_test)
```

```
In [22]: #mean vecs
         np.set_printoptions(precision=4)
         mean_vecs=[]
         for label in range(1,4):
             mean_vecs.append(np.mean(x_train_std[y_train==label],axis=0))
             print('MV %s: %s\n' % (label,mean_vecs[label-1]))
```

```
MV 1: [ 0.9066 -0.3497  0.3201 -0.7189  0.5056  0.8807  0.9589 -0.5516  0.5416
  0.2338  0.5897  0.6563  1.2075]


MV 2: [-0.8749 -0.2848 -0.3735  0.3157 -0.3848 -0.0433  0.0635 -0.0946  0.0703
 -0.8286  0.3144  0.3608 -0.7253]


MV 3: [ 0.1992  0.866   0.1682  0.4148 -0.0451 -1.0286 -1.2876  0.8287 -0.7795
```

```
  0.9649 -1.209  -1.3622 -0.4013]
```

```
In [23]: # within-class scatter matrix
         d=13
         S_W=np.zeros((d,d))
         for label, mv in zip(range(1,4), mean_vecs):
             class_scatter=np.zeros((d,d))
             for row in x_train_std[y_train==label]:
                 row,mv=row.reshape(d,1),mv.reshape(d,1)
                 class_scatter+=(row-mv).dot((row-mv).T)
             S_W+=class_scatter

         print('Within-class scatter matrix: %sX%s' % (S_W.shape[0],S_W.shape[1]))
```

```
Within-class scatter matrix: 13X13
```

```
In [24]: # check class label distribution
         print('Class label distribution: %s' % np.bincount(y_train)[1:])
```

```
Class label distribution: [41 50 33]
```

```
In [25]: # scaled Sw(scaling===>>>covariance)
         d=13
         S_W=np.zeros((d,d))
         for label, mv in zip(range(1,4), mean_vecs):
             class_scatter=np.cov(x_train_std[y_train==label].T)
             S_W+=class_scatter

         print('Scaled-Within-class scatter matrix: %sX%s' % (S_W.shape[0],S_W.shape[1]))
```

```
Scaled-Within-class scatter matrix: 13X13
```

```
In [26]: # Between-Class scatter matrix
         mean_overall=np.mean(x_train_std,axis=0)
         d=13
         S_B=np.zeros((d,d))
         for i,mean_vec in enumerate(mean_vecs):
             n=x_train[y_train==i+1,:].shape[0]  #class=i+1 のサンプル数
             mean_vec=mean_vec.reshape(d,1)
             mean_overall=mean_overall.reshape(d,1)
             S_B+=n*(mean_vec-mean_overall).dot((mean_vec-mean_overall).T)
```

```
        print('Between-class scatter matrix: %sX%s' % (S_B.shape[0],S_B.shape[1]))
```

```
Between-class scatter matrix: 13X13
```

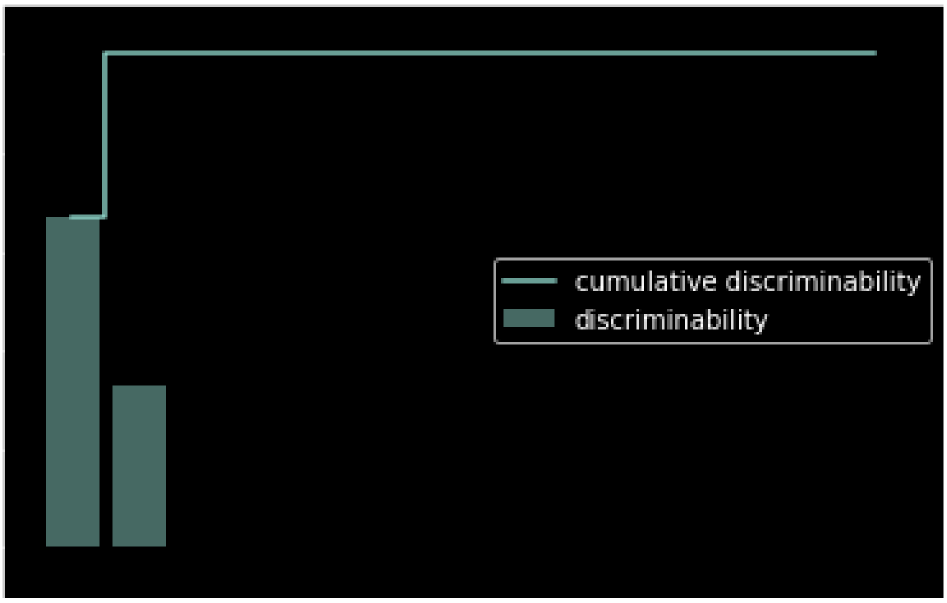
```
In [27]: # calculate eigen factors and sort eigen values
         eigen_vals,eigen_vecs=np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
         eigen_pairs=[(np.abs(eigen_vals[i]),eigen_vecs[:,i]) for i in range(len(eigen_vals))]
         eigen_pairs=sorted(eigen_pairs,key=lambda k:k[0],reverse=True)
         print('Eigenvalues in descending order:\n')
         for eigen_val in eigen_pairs:
             print(eigen_val[0])
         # Sb はランクが 1 以下の行列をクラス数 c だけ足したものなので、最大 c-1 しか固有値は存在しない
```

```
Eigenvalues in descending order:

349.617808905994
172.76152218979385
3.192705060156757e-14
2.842170943040401e-14
2.1616177227423074e-14
1.4466723349503452e-14
1.387463933334994e-14
1.1996509047065428e-14
1.1996509047065428e-14
4.191784243012234e-15
4.191784243012234e-15
3.975578490862407e-15
1.2123024580195727e-15
```

```
In [28]: # dsicriminability
         tot=sum(eigen_vals.real)
         discr=[(i/tot) for i in sorted(eigen_vals.real,reverse=True)]
         cum_discr=np.cumsum(discr)
```

```
In [29]: # show figure(discriminability & cumlative dicriminability)
         plt.bar(range(1,14),discr,alpha=0.5,align='center',label='discriminability')
         plt.step(range(1,14),cum_discr,where='mid',label='cumulative discriminability')
         plt.ylabel('discriminability ratio')
         plt.xlabel('Linear discriminants')
         plt.ylim([-0.1,1.1])
         plt.legend(loc='best')
         plt.tight_layout()
         plt.show()
```



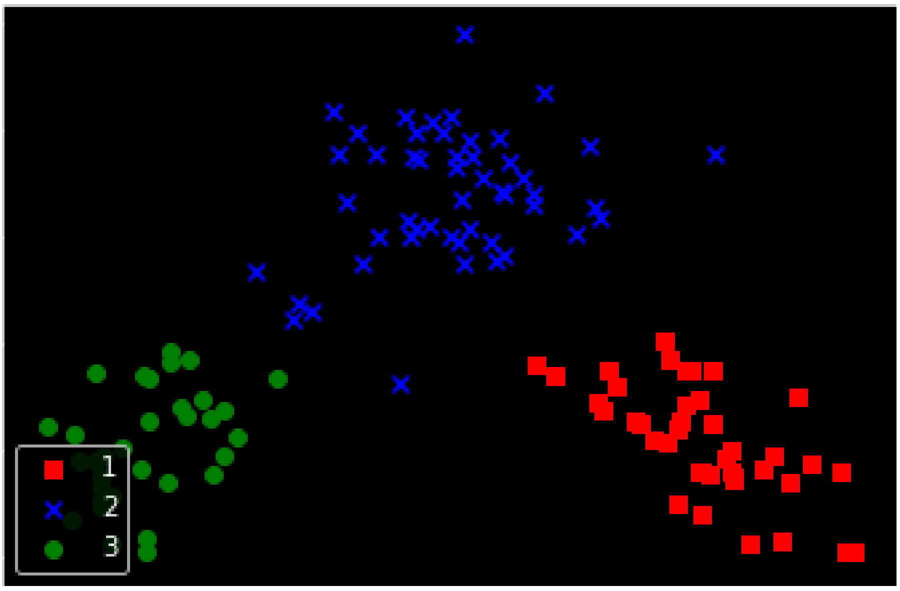
```
In [30]: # select most important eigen factors
         w=np.hstack((eigen_pairs[0][1][:,np.newaxis].real,eigen_pairs[1][1][:,np.newaxis].real))
         print('Matrix W:\n',w)
```

```
Matrix W:
 [[ 0.1481 -0.4092]
 [-0.0908 -0.1577]
 [ 0.0168 -0.3537]
 [-0.1484  0.3223]
 [ 0.0163 -0.0817]
 [-0.1913  0.0842]
 [ 0.7338  0.2823]
 [ 0.075  -0.0102]
 [-0.0018  0.0907]
 [-0.294  -0.2152]
 [ 0.0328  0.2747]
 [ 0.3547 -0.0124]
 [ 0.3915 -0.5958]]
```

```
In [31]: # X'=XW
         x_train_lda=x_train_std.dot(w)
         colors=['r','b','g']
         markers=['s','x','o']
```

```
        for l,c,m in zip(np.unique(y_train),colors,markers):
            plt.scatter(x_train_lda[y_train==l,0],x_train_lda[y_train==l,1],c=c,label=l,marker=m)

        plt.xlabel('LD 1')
        plt.ylabel('LD 2')
        plt.legend(loc='lower left')
        plt.tight_layout
        plt.show()
```



2 Sklearn での LogisticRegression を用いた実装

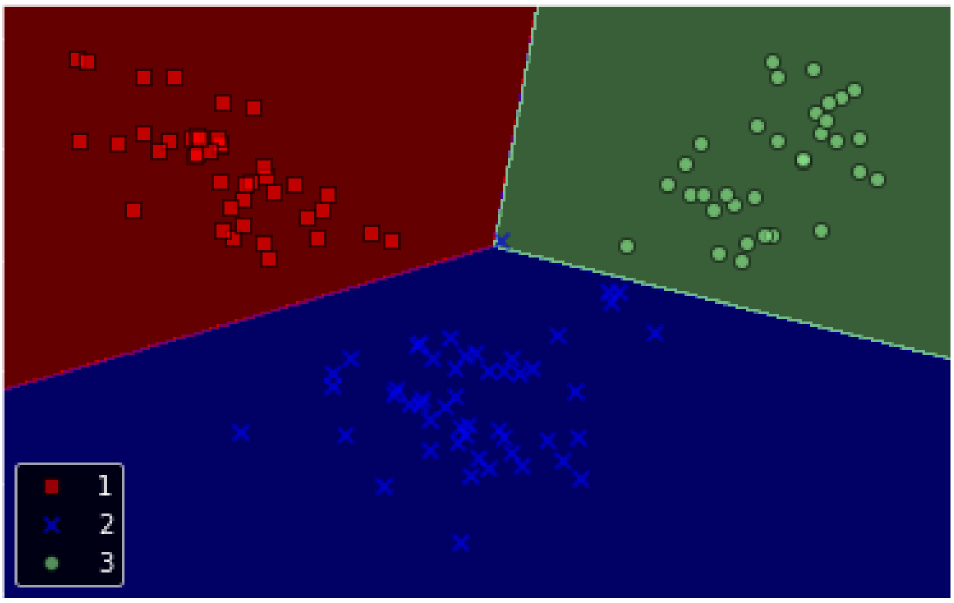
```
In [32]: # Sklearn implementation
         lda=LDA(n_components=2)
         x_train_lda_sk=lda.fit_transform(x_train_std,y_train)
         lr=LogisticRegression()
         lr=lr.fit(x_train_lda_sk,y_train)
```

```
C:\Users\taiki\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\logisti
  FutureWarning)
C:\Users\taiki\AppData\Local\Programs\Python\Python37\lib\site-packages\sklearn\linear_model\logisti
  "this warning.", FutureWarning)
```

```
In [33]: # show figure
         pdr(x_train_lda_sk,y_train,classifier=lr)
```

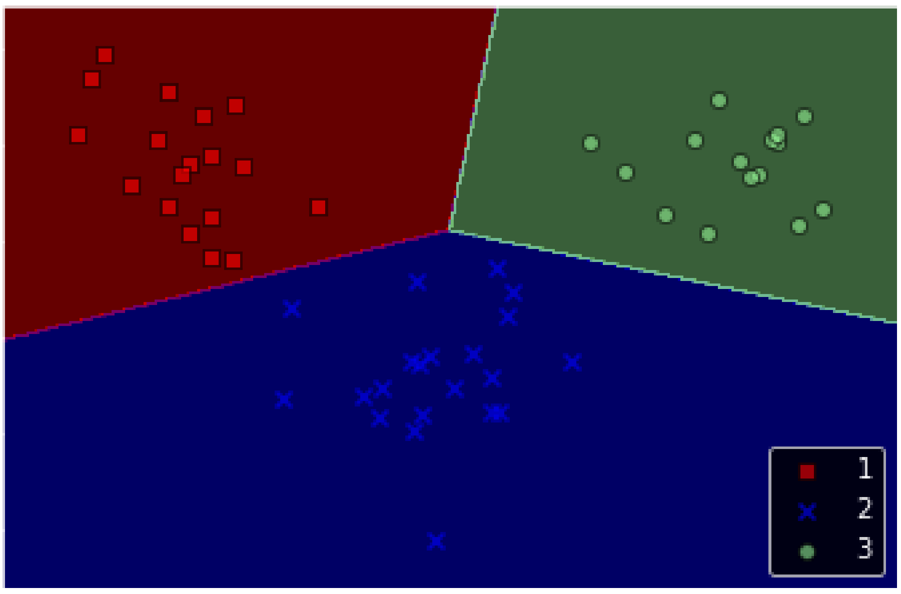
```
        plt.xlabel('LD1')
        plt.ylabel('LD2')
        plt.legend(loc='lower left')
        plt.tight_layout()
        plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapp
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapp
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapp
```



```
In [34]: # test check
         x_test_lda_sk=lda.transform(x_test_std)
         pdr(x_test_lda_sk,y_test,classifier=lr)
         plt.xlabel('LD1')
         plt.ylabel('LD2')
         plt.legend(loc='lower right')
         plt.tight_layout
         plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapp
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapp
'c' argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapp
```



2 次元の特徴部分空間でもテストで 100 %線形分類できていることがわかる

In [35]: