Lab1

September 28, 2016

1 Akari Ishikawa RA163282

2 Exercício 01 - MC886 A

```
In [1]: import numpy as np
        import pandas as pd
```

```
In [2]: import sklearn
        import sklearn.discriminant_analysis as lda
        import sklearn.decomposition
```

```
In [3]: data = np.genfromtxt('data.csv', delimiter=',')
        print(data.shape)
```

```
(477, 167)
```

```
In [4]: # Removemos a última coluna do csv e a primeira linha que falava das featur
        data1 = data[1:, :-1]
        print(data1.shape)
```

```
(476, 166)
```

3 1) PCA

```
In [5]: pca = sklearn.decomposition.PCA(copy=True)
```

```
In [6]: pca.fit(data1)
```

```
Out[6]: PCA(copy=True, n_components=None, whiten=False)
```

```
In [7]: np.cumsum(pca.explained_variance_ratio_)
```

```
Out[7]: array([ 0.31188125,  0.45110049,  0.52728075,  0.57871107,  0.62790775,
                0.66868927,  0.70115259,  0.73157891,  0.75149626,  0.76866767,
                0.7841581 ,  0.79847584,  0.81177395,  0.8244771 ,  0.83539864,
                0.84538246,  0.85404078,  0.86220635,  0.8697735 ,  0.87691348,
```

```
            0.88366985,  0.89003952,  0.89635954,  0.90219742,  0.90778833,
            0.91302009,  0.91796749,  0.92286063,  0.92753044,  0.93196024,
            0.93596809,  0.93996068,  0.94380535,  0.94713044,  0.95006551,
            0.952904  ,  0.95556949,  0.95810425,  0.96053757,  0.96258962,
            0.96459123,  0.96645684,  0.96824055,  0.96989498,  0.97151772,
            0.97300967,  0.97444944,  0.97580079,  0.97703677,  0.97823005,
            0.97936854,  0.98044865,  0.98148826,  0.98244777,  0.98335119,
            0.98421892,  0.98504699,  0.98582422,  0.98658195,  0.98729275,
            0.98793209,  0.98855017,  0.98909833,  0.98964073,  0.99015928,
            0.99062971,  0.99108475,  0.99150553,  0.99190155,  0.9922765 ,
            0.99262038,  0.99295524,  0.99328065,  0.99360092,  0.99391974,
            0.99421253,  0.994494  ,  0.99475744,  0.99501492,  0.99526007,
            0.99548223,  0.995696  ,  0.99590043,  0.99610138,  0.99628465,
            0.99646083,  0.99663284,  0.99680215,  0.99696691,  0.99711137,
            0.99724949,  0.99738161,  0.99750783,  0.99762882,  0.9977424 ,
            0.99785097,  0.99795661,  0.99805876,  0.99815604,  0.99825005,
            0.99834086,  0.99842216,  0.99850204,  0.99858134,  0.99865945,
            0.99873451,  0.99880654,  0.99887223,  0.99893224,  0.99899107,
            0.99904907,  0.99910407,  0.99915774,  0.99920611,  0.99925271,
            0.99929426,  0.99933471,  0.99937279,  0.99940891,  0.9994431 ,
            0.99947583,  0.99950773,  0.99953814,  0.9995667 ,  0.99959406,
            0.99962037,  0.99964579,  0.99966868,  0.99969089,  0.99971229,
            0.99973209,  0.99975141,  0.99976921,  0.99978678,  0.99980322,
            0.99981854,  0.99983254,  0.99984585,  0.99985804,  0.99987001,
            0.99988166,  0.99989134,  0.99990073,  0.99990983,  0.99991805,
            0.99992601,  0.99993322,  0.99994028,  0.99994713,  0.99995323,
            0.99995891,  0.99996385,  0.99996848,  0.99997301,  0.99997674,
            0.99998028,  0.99998358,  0.99998672,  0.99998936,  0.99999161,
            0.99999353,  0.99999532,  0.999997  ,  0.9999982 ,  0.99999927,  1.
```

3.0.1 Como eu gostaria de manter 80% da variância, o número de dimensões que devo manter deve ser igual a posição do primeiro elemento cujo valor é >0.8. No caso, 13 dimensões.

In [8]:
```
pca = sklearn.decomposition.PCA(n_components=13)
```

In [9]:
```
new_data_pca = pca.fit_transform(data1)
print(new_data_pca)
```

```
[[ -2.94087385   9.68333061  -3.62928882 ...,  -1.06968525  -1.13893206
   -0.28967336]
 [  4.7553443   -2.90471677  -4.49673432 ...,   1.6770415   -1.97068645
   -0.57238956]
 [-10.93898353  -3.20602349   2.97069575 ...,   0.0274727    0.23875956
    0.1632103 ]
 ...,
 [ -9.08467378  -3.72857516   0.22389101 ...,  -1.68233633   0.64949083
   -0.69624195]
 [  7.03199293  -3.62465261   1.3818902  ...,  -0.93995011  -0.74491111
```

```
   -0.1071369 ]
 [  7.17548827  -2.58143506   5.47042863 ...,  -0.27656227  -2.58848942
    2.23865287]]
```

3.0.2 Encontramos aqui os novos dados redimensionados pelo PCA

4 2) Logistic Regression

4.0.1 Agora aplicaremos a regressão logística nos dados com PCA

```
In [10]: lr = sklearn.linear_model.LogisticRegression()
         print(lr)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```
In [11]: #Chamamos de X1 e Y1 os dados de treino e teste, respectivamente, com PCA
         X1 =new_data_pca[0:200]
         Y1 = new_data_pca[200:len(new_data_pca)]
         print (X1.shape, Y1.shape)
```

```
(200, 13) (276, 13)
```

```
In [12]: output = (data[1:,-1:])

         treino =(output[0:200, ])
         treino = np.reshape(treino, np.size(treino))

         teste = output[200:len(output),]
         teste = np.reshape(teste, np.size(teste))
         print(treino.shape, teste.shape)
```

```
(200,) (276,)
```

```
In [13]: lr.fit(X1, treino)

Out[13]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
In [14]: pred1 = lr.predict(Y1)
```

```
In [15]: #Calculamos a taxa de acerto da LR com o PCA
         acerto1 = np.mean(pred1 == teste)
         print(acerto1)
```

```
0.797101449275
```

4.0.2 Calcularemos agora a taxa de acerto do LR sem o PCA

```
In [16]: lr2 = sklearn.linear_model.LogisticRegression()
```

```
In [17]: #Chamamos de X2 e Y2 os dados de treino e teste, respectivamente, sem PCA
         X2 = data1[0:200]
         Y2 = data1[200:len(data1)]
         print (X2.shape, Y2.shape)
```

```
(200, 166) (276, 166)
```

```
In [18]: #Usamos o conjunto de output já extraído anteriormente
         lr2.fit(X2, treino)
```

```
Out[18]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

```
In [19]: pred2 = lr2.predict(Y2)
```

```
In [20]: #Calculamos a taxa de acerto da LR sem o PCA
         acerto2 = np.mean(pred2 == teste)
         print(acerto2)
```

```
0.797101449275
```

4.0.3 Podemos ver que obtemos a mesma taxa de acerto em ambos os métodos

5 3) LDA

5.0.1 Vamos fazer o LDA com os dados do PCA:

```
In [21]: lda1 = lda.LinearDiscriminantAnalysis()
```

```
In [22]: lda1.fit(X1, treino)
```

```
Out[22]: LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                       solver='svd', store_covariance=False, tol=0.0001)
```

```
In [23]: pred3 = lda1.predict(Y1)
```

```
In [24]: #Calculamos a taxa de acerto do LDA com o PCA
         acerto3 = np.mean(pred3 == teste)
         print(acerto3)
```

```
0.786231884058
```

5.0.2 Vamos fazer o LDA sem os dados do PCA

```
In [25]: lda2 = lda.LinearDiscriminantAnalysis()
```

```
In [26]: lda2.fit(X2, treino)
```

```
Out[26]: LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                       solver='svd', store_covariance=False, tol=0.0001)
```

```
In [27]: pred4 = lda2.predict(Y2)
```

```
In [28]: #Calculamos a taxa de acerto do LDA sem o PCA
         acerto4 = np.mean(pred4 == teste)
         print(acerto4)
```

```
0.677536231884
```

6 4) Podemos concluir que o Logistic Regression não possui diferença de desempenho no uso de dados com PCA. No entanto, o LDA tem sua taxa de acerto otimizada.