Aprendizado de Máquina - MO444

Exercício 1

Aluno: Paulo Ricardo Finardi. RA: 144809

0 Preliminares

Foi utilizado `Python 3` na resolução dos exercícios. O código abaixo é o *header* de todos os itens.

```python
import numpy as np
import pandas as pd
import csv
import matplotlib.pyplot as plt

# read dada1.csv in python with Pandas
data1 = pd.read_csv('data1.csv')

# delete the last column of data1
data1.set_index('clase').to_csv('data.csv', index=None)

# saving new file in csv format
data = pd.read_csv('data.csv')

# convert the datas in numpy-array
data1 = data1.as_matrix(columns=None)
data  = data.as_matrix(columns=None)
```

1 PCA

Enunciado: Faça o *Principal Component Analysis* (PCA) dos dados sem a última coluna. Se você quiser que os dados transformados tenham 80% da variância original, quantas dimensões do

PCA você precisa manter?

Resposta: São necessárias 12 dimensões. A variância original é fornecida em cada componente do PCA. Somamos componente a componente até obter a variância de 80%. O código para realização dessa tarefa é fornecido a seguir.

```python
# function PCA with sklearn
def doPCA(x): # x is the argument for the number of components in PCA
    from sklearn.decomposition import PCA
    pca = PCA(n_components=x)
    pca.fit(data)
    return pca

# this step is obsolete, but we'll repeat the PCA after we find the requested variance.
pca = doPCA(data.shape[1])

# the elements of the _variance contains the total variance.
_variance = pca.explained_variance_ratio_

# store the requested variance
requested_variance = 0

# the number of the variance requested
lim = 0.80

# loop to find the quantity of elements
for i in range (len(_variance)):
    requested_variance += _variance[i]
    if requested_variance >= lim:
        element = i
        requested_variance -= _variance[i] # just for adjust the index
        break

print('With %d elements, we get the requested variance, that is: %.3f.' %
     (element, requested_variance))
```

```
31 # repeating PCA
32 pca = doPCA(element)
33 variance_ = pca.explained_variance_ratio_
34
35 # prints
36 print('The %d principal components:' % (element))
37 np.set_printoptions(precision=3)
38 print(variance_)
39 print('\nThe sum of the components is: %.3f.' % np.sum(variance_))
```

Na linha `38`, a função `print` fornece:

$$[0.312,\ 0.139,\ 0.076,\ 0.051,\ 0.049,\ 0.041,\ 0.032,\ 0.03,\ 0.02,\ 0.017,\ 0.015,\ 0.014],$$

e a soma dessas componentes vale 0.798 ou 79.8% (dada pela linha `39`).

2 Regressão logística

Enunciado: Considere as primeiras 200 linhas dos dados como o conjunto de treino, e as 276 ultimas como o conjunto de dados. Treine uma regressão logística no conjunto de treino dos dados originais e nos dados transformados. Qual a taxa de acerto no conjunto de teste nas 2 condições (sem e com PCA)?

`Resposta:` Nessa etapa foi utilizado a matriz de confusão. A escolha da matriz de confusão vem do fato que ela oferece uma medida efetiva do modelo de classificação ao mostrar o número de classificações corretas versus as classificações preditas para cada classe, sobre os conjuntos de testes. As métricas utilizadas foram as seguintes (os termos em inglês foram mantidos):

- `True Positive Rate` (TPR) – fornece a sensibilidade do classificador;
- `False Positive Rate` (FPR) – fornece a quantidade de alarmes falsos do classificador;
- `Accuracy` – descrição de erros sistemáticos;
- `Precision` – descrição de erros aleatórios, variabilidade estatística.
- `Recall` – razão entre $TP/(TP + FN)$;
- `F1-Score` – média harmônica entre `Precision` e `Recall`.

As Figuras (1) e (2) exibem as matrizes de confusão dos conjuntos de teste (*sem PCA*) e (*com PCA*) respectivamente.

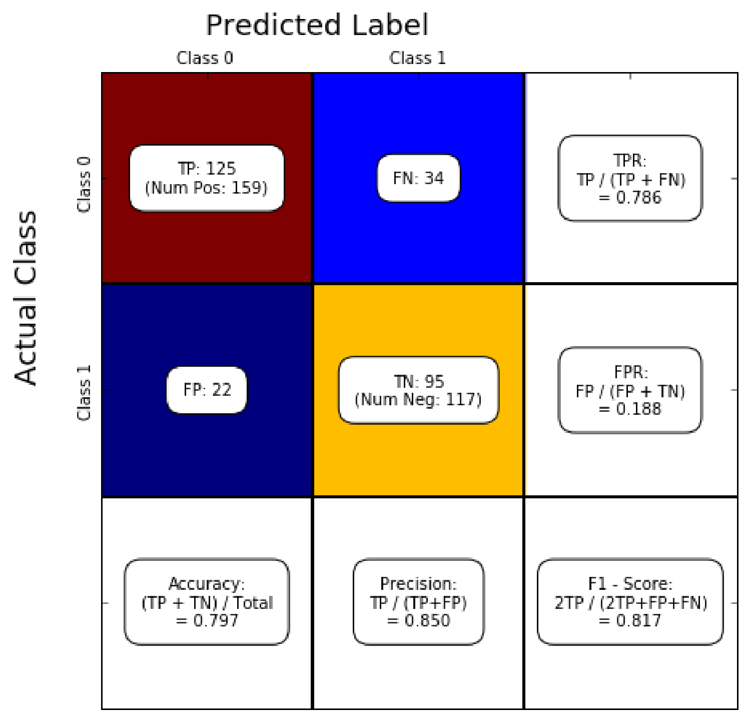

Figura 1: Matriz de confusão– dados *sem PCA*.

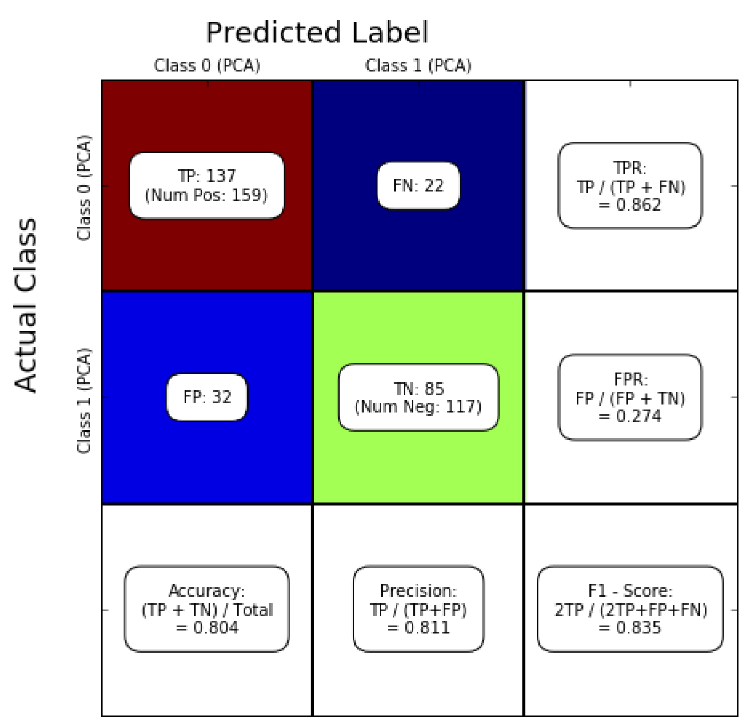

Figura 2: Matriz de confusão– dados *com PCA*.

Os dados com PCA obtiveram melhor resultado (exceto em `Precision`). A seguir o código:

```python
# data with pca
transformed_data = pca.transform(data)

# y_total is the expected
y_total = data1[: , 166]

# split data in train and test sets
X_train_PCA, X_train, y_train = transformed_data[0:200, :], data[0:200,:], y_total[0:200]
X_test_PCA,  X_test,  y_test  = transformed_data[200:,:], data[200:,:], y_total[200:]

# metrics for the confusion matrix
from sklearn import metrics

# function Linear Regression (LR) with sklearn
def doLR(X, y):
    from sklearn.linear_model \
    import LogisticRegression as LR
    lr = LR()
    lr.fit(X, y)
    return lr

# LR in PCA data
lrPCA = doLR(X_train_PCA, y_train)

# LR in data
lr = doLR(X_train, y_train)

# predicted sets
predictedPCA = lrPCA.predict(X_test_PCA)
predicted    = lr.predict(X_test)

# confusion matrix
m_confusion_PCA = metrics.confusion_matrix(y_test, predictedPCA)
m_confusion     = metrics.confusion_matrix(y_test, predicted)
```

```

# function for confusion matrix
def show_confusion_matrix(C,class_labels=['0','1']):

    # true negative, false positive, etc...
    tp = C[0,0]; fn = C[0,1]; fp = C[1,0]; tn = C[1,1];

    NP = fn+tp # num positive examples
    NN = tn+fp # num negative examples
    N  = NP+NN

    fig = plt.figure(figsize=(6.5,6.5))
    ax  = fig.add_subplot(111)
    ax.imshow(C, interpolation='nearest', cmap='jet')

    # draw the grid boxes
    ax.set_xlim(-0.5,2.5)
    ax.set_ylim(2.5,-0.5)
    ax.plot([-0.5,2.5],[0.5,0.5], '-k', lw=2)
    ax.plot([-0.5,2.5],[1.5,1.5], '-k', lw=2)
    ax.plot([0.5,0.5],[-0.5,2.5], '-k', lw=2)
    ax.plot([1.5,1.5],[-0.5,2.5], '-k', lw=2)

    # set xlabels
    ax.set_xlabel('Predicted Label', fontsize=18)
    ax.set_xticks([0,1,2])
    ax.set_xticklabels(class_labels + [''])
    ax.xaxis.set_label_position('top')
    ax.xaxis.tick_top()

    # these coordinate might require some tinkering.
    ax.xaxis.set_label_coords(0.34,1.06)

    # set ylabels
    ax.set_ylabel('Actual Class', fontsize=18, rotation=90)
    ax.set_yticklabels(class_labels + [''],rotation=90)
```

```python
    ax.set_yticks([0,1,2])
    ax.yaxis.set_label_coords(-0.09,0.65)

    # fill in initial metrics: tp, tn, etc...
    ax.text(1,1,
            'TN: %d\n(Num Neg: %d)'%(tn, NN),
            va='center',
            ha='center',
            bbox=dict(fc='w',boxstyle='round,pad=1'))

    ax.text(1,0,
            'FN: %d'%fn,
            va='center',
            ha='center',
            bbox=dict(fc='w',boxstyle='round,pad=1'))

    ax.text(0,1,
            'FP: %d'%fp,
            va='center',
            ha='center',
            bbox=dict(fc='w',boxstyle='round,pad=1'))

    ax.text(0,0,
            'TP: %d\n(Num Pos: %d)'%(tp,NP),
            va='center',
            ha='center',
            bbox=dict(fc='w',boxstyle='round,pad=1'))

    # secondary metrics: accuracy, true pos rate, etc...
    ax.text(2,1,
            'FPR:\nFP / (FP + TN)\n= %.3f'%(fp / (fp+tn)),
            va='center',
            ha='center',
            bbox=dict(fc='w',boxstyle='round,pad=1'))

    ax.text(2,0,
```

```
                'TPR:\nTP / (TP + FN)\n= %.3f'%(tp / (tp+fn)),
                va='center',
                ha='center',
                bbox=dict(fc='w',boxstyle='round,pad=1'))

    ax.text(0,2,
                'Accuracy:\n(TP + TN) / Total\n= %.3f'%((tp+tn)/N),
                va='center',
                ha='center',
                bbox=dict(fc='w',boxstyle='round,pad=1'))

    ax.text(2,2,
                'F1 - Score:\n2TP / (2TP+FP+FN)\n= %.3f'%(2*tp/(2*tp+fp+fn)),
                va='center',
                ha='center',
                bbox=dict(fc='w',boxstyle='round,pad=1'))

    ax.text(1,2,
                'Precision:\nTP / (TP+FP)\n= %.3f'%(tp/(tp+fp)),
                va='center',
                ha='center',
                bbox=dict(fc='w',boxstyle='round,pad=1'))

    plt.tight_layout()
    plt.show()

# prints
show_confusion_matrix(m_confusion, ['Class 0 ', 'Class 1'])
show_confusion_matrix(m_confusion_PCA, ['Class 0 (PCA)','Class 1 (PCA)'])
```

3 LDA

Enunciado: Treine o *Linear Discriminant Analysis* (LDA) nos conjuntos de treino com e sem PCA e teste nos respectivos conjuntos de testes. Qual a acurácia nas 2 condições?

`Resposta:` Adotamos a mesma estratégia do item anterior. Treinamos o LDA e geramos novas matrizes de confusão para o classificador LDA. Figuras (3) e (4).

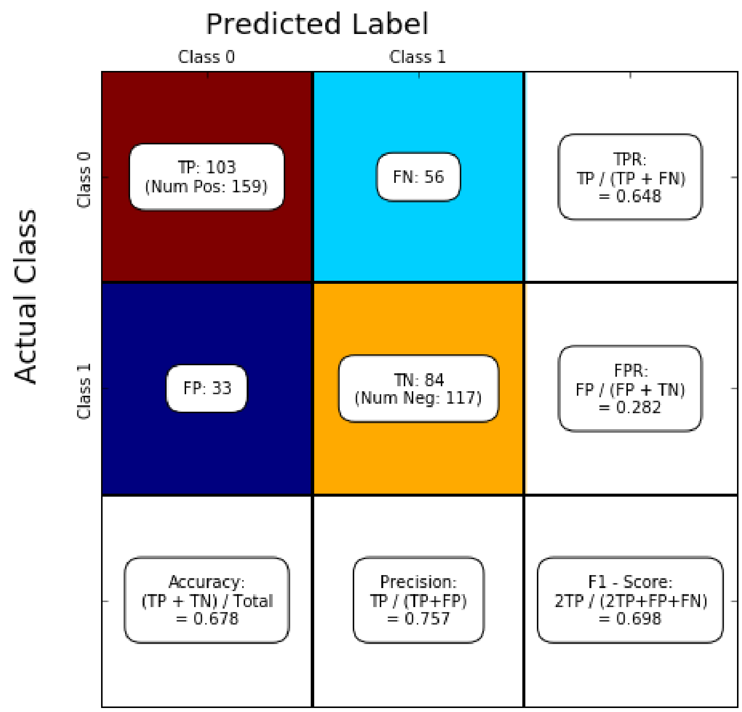

Figura 3: Matriz de confusão– LDA dados *sem PCA*.

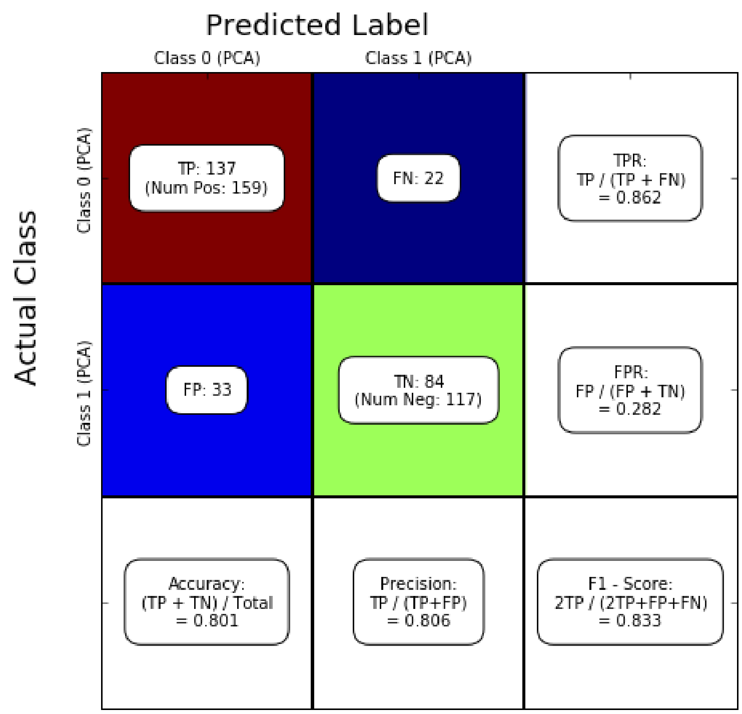

Figura 4: Matriz de confusão– LDA dados *com PCA*.

Conclusão: Diferente do item da regressão logistica (2), a variação da acurácia nos dois conjuntos foi maior. No conjunto de dados *sem PCA* a acurácia foi de 0.678 ou 67, 8%, já no conjunto de dados *com PCA* a acurácia foi de 0.801 ou 80, 1%. Na sequência o código.

```
# LDA function
def doLDA(X, y, n):
    # X is the train set, y is the expected, and n is the number of components
    from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
    lda = LDA(n_components=n)
    lda.fit(X, y)
    return lda

# lda in PCA data
ldaPCA = doLDA(X_train_PCA, y_train, element)

# lda in data
lda = doLDA(X_train, y_train, element)

```

```
# predicted sets
predicted_ldaPCA = ldaPCA.predict(X_test_PCA)
predicted_lda    = lda.predict(X_test)

# confusion matrix
lda_confusionPCA = metrics.confusion_matrix(y_test, predicted_ldaPCA)
lda_confusion    = metrics.confusion_matrix(y_test, predicted_lda)

# prints
show_confusion_matrix(lda_confusionPCA, ['Class 0 (PCA)','Class 1 (PCA)'])
show_confusion_matrix(lda_confusion, ['Class 0','Class 1'])
```

4 Melhor combinação

Enunciado: Qual a melhor combinação de classificador e PCA ou não?

`Resposta:` Considerando as matrizes de confusão das Figuras (1), (2), (3) e (4) a que apresenta os melhores resultados é a Figura (2), ou seja, **o melhor resultado** é obtido nos dados ***com PCA* e regressão logística**. Esse resultado é bem sutil; ele obteve 32 `FP` contra 33 `FP` no classificador LDA. Com essa combinação, também temos a melhor métrica `F1-Score` (dois centésimos maior do que o PCA-LDA).