Marcos Roberto e Souza - 188973

Passo 0: Preparação dos dados

Antes de começarmos com o código de fato, devemos importar algumas bibliotecas do python que serão necessárias para a execução dos passos seguintes.

```
import csv
import numpy as np
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.lda import LDA
```

Com a biblioteca csv, o arquivo data1.csv pode ser facilmente lido e disposto em uma lista (rows). A primeira linha (cabeçalho) foi descartada por meio do comando "f.next()".

```
filename = '../data1.csv'
f = csv.reader(open(filename), delimiter=',')
f.next();
rows = []
for row in f:
    rows.append(row)
```

A partir da lista rows, um array da biblioteca numpy será criado. Este passo permitirá uma melhor manipulação dos dados.

```
data = np.array(rows,dtype=float)
```

Agora, os dados podem ser facilmente separados de seus rótulos (classe). Em primeiro lugar, salvamos todos valores da última coluna (rótulos) como inteiros. Em seguida, os obtemos do array de dados. E por fim, separamos os 200 primeiros dados para o conjunto de treinamento, e o restante para o conjunto de teste.

```
labels = [int(i) for i in data[:,-1]]
data = data[:,:-1]
train = data[:200]
train_labels = labels[:200]
test = data[200:]
test_labels = labels[200:]
```

Passo 1: PCA

Para aplicarmos o PCA em de forma que os dados tenham 80% da variância original, basta criar uma instância do PCA com 0.8 como parâmetro. No código a seguir, além da criação da instância do PCA, os dados são transformados. É importante notar que o PCA deve ser aplicado somente ao conjunto de treinamento. Ou seja, as instâncias de teste, que são teoricamente conhecimento futuro, não devem ser utilizadas para se estimar o PCA.

```
pca = PCA(0.8)
train_pca = pca.fit_transform(train)
test_pca = pca.transform(test)
print "Variância de 80%: "\
+str(train_pca.shape[1])+" dimensões."
```

```
Variância de 80%: 12 dimensões.
```

Passo 2: Regressão Logística (LR)

Para treinarmos um modelo devemos tipicamente possuir:

- Conjunto de Treinamento (train)
- Rótulos para o conjunto de Treinamento (labels_train)

Enquanto que para determinarmos a taxa de acerto do modelo:

- Conjunto de Teste (test)
- Rótulos para o conjunto de teste (labels_test)

A seguir, treinaremos e determinaremos a taxa de acerto utilizando a Regressão Logística para a classificação. Consideraremos tanto os dados originais, quanto os dados transformados pelo PCA.

```
#Dados Originais
lr = LogisticRegression()
model_lr = lr.fit(train,train_labels)
accuracy_lr = model_lr.score(test,test_labels)

#Dados Transformados
lr_pca = LogisticRegression()
model_lr_pca = lr_pca.fit(train_pca,train_labels)
accuracy_pca = model_lr_pca.score(test_pca,test_labels)
```

```
print "LR: "+str(accuracy_lr)
print "LR + PCA: "+str(accuracy_pca)
```

```
LR: 0.797101449275
LR + PCA: 0.800724637681
```

Podemos notar um leve ganho na taxa de acerto ao se utilizar o PCA para o classificador de Regressão Logística.

Passo 3: Análise de Discriminantes Lineares (LDA)

Este passo segue o mesmo padrão do passo 2. Os mesmos dados de entrada são necessários, e as funções devem ser executadas da mesma forma. O código a seguir treina e determina a taxa de acerto para o classificador LDA.

```
#Dados Originais
lda = LDA()
model_lda = lda.fit(train,train_labels)
accuracy_lda = model_lda.score(test,test_labels)

#Dados Transformados
lda = LDA()
model_lda_pca = lda.fit(train_pca,train_labels)
accuracy_lda_pca = model_lda_pca.score(test_pca,test_labels)

print "LDA: "+str(accuracy_lda)
print "LDA + PCA: "+str(accuracy_lda_pca)
```

```
LDA: 0.677536231884
LDA + PCA: 0.778985507246
```

Passo 4: Comentários Finais

Para o LDA, houve um ganho mais expressivo na taxa de acerto ao transformar os dados com o PCA. De qualquer forma, o melhor resultado, para esse conjunto de treino/teste, ainda é obtido por meio do uso do classificador de Regressão Logística com PCA (80%), contra 77,8% do LDA com PCA.