Exercício 3 - MO444 - Aprendizado de máquina e reconhecimento de padrões

Renato Lopes Moura - 163050

Importando os módulos que serão utilizados no exercício.

```
In [1]: import numpy as np
        import pandas as pd
        from sklearn.preprocessing import Imputer
        from sklearn.preprocessing import scale
        from sklearn.decomposition import PCA
        from sklearn.model_selection import StratifiedKFold
        from sklearn.model_selection import GridSearchCV
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.svm import SVC
        from sklearn.neural_network import MLPClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.ensemble import GradientBoostingClassifier
```

Preparação dos dados para validação.

```
In [2]: #Carregando o conjunto de dados e as respectivas classes utilizando o pandas
        data = pd.read_table('secom.data', sep=' ', header=None, index_col=False)
        data_class = pd.read_table('secom_labels.data', sep=' ', header=None, index_col=Fa

        #Separando os dados em atributos e classes
        X = data.values
        Y = data_class.iloc[:,0].values

        #Instanciando o objeto Imputer para executar a imputacao de valores
        #faltantes pela media e aplicando no conjunto de dados
        imp = Imputer(missing_values='NaN', strategy='mean', axis=0)

        X = imp.fit_transform(X)

        #Aplicando padronizacao nas colunas para media 0 e desvio padrao 1
        X = scale(X)
```

In [3]:
```
#############################################################################
#Classificacao utilizando o algoritmo kNN

pca = PCA(0.8)
X_pca = pca.fit_transform(X)

knn_parameters = {'n_neighbors':[1,5,11,15,21,25]}

acc_knn = 0

external = StratifiedKFold(n_splits=5)

for train, test in external.split(X_pca,Y):
        data_train = X_pca[train]
        data_test = X_pca[test]
        classes_train = Y[train]
        classes_test = Y[test]

        grid = GridSearchCV(KNeighborsClassifier(), knn_parameters, cv=3)
        grid.fit(data_train, classes_train)

        knn = KNeighborsClassifier(n_neighbors=grid.best_params_['n_neighbors'])
        knn.fit(data_train, classes_train)

        acc_knn = acc_knn + knn.score(data_test, classes_test)

acc_knn = acc_knn / 5

print "A acuracia do KNN eh "+str(acc_knn)
```

```
A acuracia do KNN eh 0.929811912242
```

In [4]:
```
#########################################################################
#Classificacao utilizando o algoritmo SVM com kernel RBF

svm_parameters = {'C':[2**(-5), 2**(0), 2**(5), 2**(10)],
 'gamma':[2**(-15), 2**(-10), 2**(-5), 2**(0), 2**(5)]}

acc_svm = 0

external = StratifiedKFold(n_splits=5)

for train, test in external.split(X,Y):
        data_train = X[train]
        data_test = X[test]
        classes_train = Y[train]
        classes_test = Y[test]

        grid = GridSearchCV(SVC(kernel='rbf'), svm_parameters, cv=3)
        grid.fit(data_train, classes_train)

        svm = SVC(C=grid.best_params_['C'], gamma=grid.best_params_['gamma'], kern
        svm.fit(data_train, classes_train)

        acc_svm = acc_svm + svm.score(data_test, classes_test)

acc_svm = acc_svm / 5

print "A acuracia do SVM eh "+str(acc_svm)
```

```
A acuracia do SVM eh 0.933633568293
```

In [5]:
```
#########################################################################
#Classificacao utilizando uma rede neural

nn_parameters = {'hidden_layer_sizes':[10,20,30,40]}

acc_nn = 0

external = StratifiedKFold(n_splits=5)

for train, test in external.split(X,Y):
        data_train = X[train]
        data_test = X[test]
        classes_train = Y[train]
        classes_test = Y[test]

        grid = GridSearchCV(MLPClassifier(solver='lbfgs'), nn_parameters, cv=3)
        grid.fit(data_train, classes_train)

        nnet = MLPClassifier(hidden_layer_sizes=grid.best_params_['hidden_layer_si
         solver='lbfgs')
        nnet.fit(data_train, classes_train)

        acc_nn = acc_nn + nnet.score(data_test, classes_test)

acc_nn = acc_nn / 5

print "A acuracia da Rede Neural eh "+str(acc_nn)
```

```
A acuracia da Rede Neural eh 0.792154913725
```

In [6]:
```
#########################################################################
#Classificacao utilizando o algoritmo Random Forest

rf_parameters = {'max_features':[10,15,20,25],'n_estimators':[100,200,300,400]}

acc_rf = 0

external = StratifiedKFold(n_splits=5)

for train, test in external.split(X,Y):
        data_train = X[train]
        data_test = X[test]
        classes_train = Y[train]
        classes_test = Y[test]

        grid = GridSearchCV(RandomForestClassifier(), rf_parameters, cv=3)
        grid.fit(data_train, classes_train)

        rf = RandomForestClassifier(max_features=grid.best_params_['max_features']
         n_estimators=grid.best_params_['n_estimators'])
        rf.fit(data_train, classes_train)

        acc_rf = acc_rf + rf.score(data_test, classes_test)

acc_rf = acc_rf / 5

print "A acuracia da Random Forest eh "+str(acc_rf)
```

```
A acuracia da Random Forest eh 0.933633568293
```

In [7]:
```
########################################################################
#Classificacao utilizando o algoritmo Gradient Boosting Machine

gbm_parameters = {'n_estimators':[30,70,100],'learning_rate':[0.1,0.05],'max_depth

acc_gbm = 0

external = StratifiedKFold(n_splits=5)

for train, test in external.split(X,Y):
        data_train = X[train]
        data_test = X[test]
        classes_train = Y[train]
        classes_test = Y[test]

        grid = GridSearchCV(GradientBoostingClassifier(), gbm_parameters, cv=3)
        grid.fit(data_train, classes_train)

        gbm = GradientBoostingClassifier(n_estimators=grid.best_params_['n_estimat
         learning_rate=grid.best_params_['learning_rate'],
          max_depth=grid.best_params_['max_depth'])
        gbm.fit(data_train, classes_train)

        acc_gbm = acc_gbm + gbm.score(data_test, classes_test)

acc_gbm = acc_gbm / 5

print "A acuracia do GBM eh "+str(acc_gbm)
```

```
A acuracia do GBM eh 0.839345689719
```

Portanto os algoritmos que possuem a maior acurácia para este conjunto de dados são **SVM** e **Random Forest**.