Exercício 3 - MC886

Pedro De Nigris Vasconcellos

`RA: 147623`

19 de outubro de 2016

1 Código

1.1 Imports

```
import csv
import numpy as np
from numpy import genfromtxt

from sklearn.cross_validation import StratifiedKFold
from sklearn.grid_search import GridSearchCV
from sklearn.decomposition import PCA

from sklearn import preprocessing
import scipy.stats as stats
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
```

1.2 Lendo o arquivo

Utilizei o SCIPY para realizar o cálculo da média das colunas assim como a imputação dos valores NAN. O método *nanmean* cálcula a média das colunas desconsiderando os dados NAN. *where* é um método que retorna uma tupla de listas, no qual a primeira contém os índices das linhas onde os dados são NAN e a segunda os índices das colunas. Tomamos as coordenads i e j, indicadas por cada lista, e substituímos pela média referente àquela coluna (motivo de utilizarmos INDS[1]). Por fim, utilizamos o método *scale* do sklearn no qual ele normaliza a matriz.

```python
# Contém todos os dados do arquivo
X = genfromtxt('secom.data.csv', delimiter = ' ')

col_mean = stats.nanmean(X);
inds = np.where(np.isnan(X))
X[inds] = np.take(col_mean, inds[1])

X_scaled = preprocessing.scale(X)

# Contém todas as classes dos dados da tabela X
Y = genfromtxt('secom_labels.data.csv', delimiter = ' ', usecols = 0)
```

1.3 Classificador: KNN

```python
def KNN (ext_fold, X, Y):
    # Fazemos o PCA no conjunto de dados
    pca = PCA(n_components = 0.80)
    X_pca = pca.fit_transform(X)
    accuracy = 0
    # Folds externos
    for ext_train_index, ext_test_index in ext_fold:
        X_5fold_train = X_pca[ext_train_index]
        Y_5fold_train = Y[ext_train_index]
        X_5fold_test = X_pca[ext_test_index]
        Y_5fold_test = Y[ext_test_index]

        neigh = KNeighborsClassifier()
        parameters = {"n_neighbors":[1, 5, 11, 15, 21, 25]}

        clf = GridSearchCV(neigh, parameters, cv = 3)
        clf.fit(X_5fold_train, Y_5fold_train)

        new_neigh = KNeighborsClassifier(clf.best_params_["n_neighbors"])
        new_neigh.fit(X_5fold_train, Y_5fold_train)
        accuracy += new_neigh.score(X_5fold_test, Y_5fold_test)

    accuracy /= 5
    return accuracy
```

1.4 Classificador: SVM

```
def SVM_RBF (ext_fold, X, Y):
    accuracy = 0
    # Folds externos
    for ext_train_index, ext_test_index in ext_fold:
        X_5fold_train = X[ext_train_index]
        Y_5fold_train = Y[ext_train_index]
        X_5fold_test = X[ext_test_index]
        Y_5fold_test = Y[ext_test_index]

        svc = SVC(kernel = "rbf")
        parameters = {"C": [2**(-5), 2**(0), 2**(5), 2**(10)], "gamma": [2**(-15), 2**(-10), 2**(-5), 2**(0),
     2**(5)]}

        clf = GridSearchCV(svc, parameters, cv = 3)
        clf.fit(X_5fold_train, Y_5fold_train)

        new_svc = SVC(C = clf.best_params_["C"], kernel = "rbf", gamma = clf.best_params_["gamma"])
        new_svc.fit(X_5fold_train, Y_5fold_train)
        accuracy += new_svc.score(X_5fold_test, Y_5fold_test)

    accuracy /= 5
    return accuracy
```

1.5 Classificador: Redes Neurais

```
def RN (ext_fold, X, Y):
    accuracy = 0
    # Folds externos
    for ext_train_index, ext_test_index in ext_fold:
        X_5fold_train = X[ext_train_index]
        Y_5fold_train = Y[ext_train_index]
        X_5fold_test = X[ext_test_index]
        Y_5fold_test = Y[ext_test_index]

        rn = MLPClassifier(solver = "lbfgs")
        parameters = {"hidden_layer_sizes": [10, 20, 30 , 40]}

        clf = GridSearchCV(rn, parameters, cv = 3)
```

```
        clf.fit(X_5fold_train, Y_5fold_train)

        new_rn = MLPClassifier(solver = "lbfgs", hidden_layer_sizes = clf.best_params_["hidden_layer_sizes"])
        new_rn.fit(X_5fold_train, Y_5fold_train)
        accuracy += new_rn.score(X_5fold_test, Y_5fold_test)

    accuracy /= 5
    return accuracy
```

1.6 Classificador: Random Forest

```
def RF (ext_fold, X, Y):
    accuracy = 0
    # Folds externos
    for ext_train_index, ext_test_index in ext_fold:
        X_5fold_train = X[ext_train_index]
        Y_5fold_train = Y[ext_train_index]
        X_5fold_test = X[ext_test_index]
        Y_5fold_test = Y[ext_test_index]

        rfc = RandomForestClassifier()
        parameters = {"n_estimators": [100, 200, 300 , 400], "max_features": [10, 15, 20, 25]}

        clf = GridSearchCV(rfc, parameters, cv = 3)
        clf.fit(X_5fold_train, Y_5fold_train)

        new_rfc = RandomForestClassifier(n_estimators = clf.best_params_["n_estimators"], max_features = clf.
     best_params_["max_features"])
        new_rfc.fit(X_5fold_train, Y_5fold_train)
        accuracy += new_rfc.score(X_5fold_test, Y_5fold_test)

    accuracy /= 5
    return accuracy
```

1.7 Classificador: GBM

```
def GBM (ext_fold, X, Y):
    accuracy = 0
    # Folds externos
```

```
    for ext_train_index, ext_test_index in ext_fold:
        X_5fold_train = X[ext_train_index]
        Y_5fold_train = Y[ext_train_index]
        X_5fold_test = X[ext_test_index]
        Y_5fold_test = Y[ext_test_index]

        gbm = GradientBoostingClassifier(max_depth = 5)
        parameters = {"learning_rate": [0.1, 0.05], "n_estimators": [30, 70, 100]}

        clf = GridSearchCV(gbm, parameters, cv = 3)
        clf.fit(X_5fold_train, Y_5fold_train)

        new_gbm = GradientBoostingClassifier(n_estimators = clf.best_params_["n_estimators"], max_depth = 5, learning_rate = clf.best_params_["learning_rate"])
        new_gbm.fit(X_5fold_train, Y_5fold_train)
        accuracy += new_gbm.score(X_5fold_test, Y_5fold_test)

    accuracy /= 5
    return accuracy
```

1.8 Main

Criei um dicionário no qual a chave é o nome do algoritmo de classificação e o valor é a sua acurácia.

```
accuracies = dict = {"KNN": -1, "SVM": -1, "RedesNeurais": -1, "RandomForest": -1, "GBM": -1}
external_5_fold = StratifiedKFold(Y, n_folds = 5)

accuracies["KNN"] = KNN(external_5_fold, X, Y)
accuracies["SVM"] = SVM_RBF(external_5_fold, X, Y)
accuracies["RedesNeurais"] = RN(external_5_fold, X, Y)
accuracies["RandomForest"] = RF(external_5_fold, X, Y)
accuracies["GBM"] = GBM(external_5_fold, X, Y)
print (accuracies)
```

2 Acurácias dos algoritmos

Por fim, imprime-se este dicionário, obtendo a acurácia externa de cada método.

Algoritmo	Acurácia (%)
K-Nearest Neighbours	92,981
SVM com kernel RBF	93,363
Redes Neurais	80,744
Random Forest	93,109
Gradient Boosting Machine	84,445

Tabela 1: Tabela com os resultados de acurácia da validação externa para cada algoritmo

Percebe-se que o algoritmo SVM com kernel RBF apresenta a maior acurácia, indo de encontro com a afirmação do professor no qual este algoritmo tem tido uma taxa de predição maior do que os outros algoritmos. Outro fato constatado é que o algoritmo RANDOM FOREST está entre os melhores (no caso, em segundo). Em seguida, segue o K-NEAREST NEIGHBOURS com acurácia ainda na casa de 90%. Tanto REDES NEURAIS quanto GBM ficaram com uma taxa de predição na casa de 80%, sendo que GBM obteve uma acurácia maior.