Nome: Nathana Facion

Exercício 3

Aprendizado de Máquina

O relatório está dividido em partes, a função Main que começa o programa se encontra na penúltima página. O código se encontra documentado, ou seja, a descrição do que foi feito em cada parte se encontra no próprio código.

1. Bibliotecas:

```
import numpy as np
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.preprocessing import Imputer
```

2. Funções usadas:

Todos os algoritmos que deveriam ser rodados se encontram em funções: SVM, KNN, Rede Neural, Random Forest, Gradient Boosting . O k-fold que é necessário para todos os algoritmos citados acima também se encontram nessa sessão.

```
def meanFinal(acfinal, n_folds):
  return float(acfinal / n_folds)

def accuracy(matrix):
  return float(matrix[0][0] + matrix[1][1]) / float(matrix.sum())

def  SVM (parameters):
  return GridSearchCV(SVC(kernel='rbf'), parameters,cv=3,  scoring='accuracy')
```

```
def KNN (parameters):
  return GridSearchCV(KNeighborsClassifier(), parameters, cv=3, scoring='accuracy')

def RN (parameters):
  return GridSearchCV(MLPClassifier(), parameters, cv=3, scoring='accuracy')

def RF (parameters):
  return GridSearchCV(RandomForestClassifier(), parameters, cv=3, scoring='accuracy')

def GBM (parameters):
  return GridSearchCV(GradientBoostingClassifier(), parameters, cv=3, scoring='accuracy')

def kfoldExterno(parameters,X,Y,algorithm):
  n_folds = 5
  external_skf = StratifiedKFold(n_folds)
  acxFinal = 0
  for training_index, test_index in external_skf.split(X,Y):
    X_train, X_test = X[training_index], X[test_index]
    Y_train, Y_test = Y[training_index], Y[test_index]
    if algorithm =='SVM':
      model = SVM(parameters)
    elif algorithm == 'KNN':
      pca = PCA(n_components=0.8)
      pca.fit(X_train)
      X_train = pca.transform(X_train)
      X_test = pca.transform(X_test)
      model = KNN(parameters)
    elif algorithm == 'RN': # Redes Neurais
      model = RN(parameters)
    elif algorithm == 'RF':  # Random Forest
      model = RF(parameters)
    elif algorithm =='GBM': # Gradient Boosting Machine
      model = GBM(parameters)
    model.fit(X_train, Y_train)
    predicted = model.predict(X_test)

    c_matrix = confusion_matrix(Y_test, predicted)
    ac = accuracy(c_matrix)
    acxFinal = ac + acxFinal

  return meanFinal(acxFinal, n_folds)
```

3. Pré-Processamento

```
# Realiza pre processamento
# - Substitua os dados faltantes pela media da coluna (imputacao pela media)
# - Finalmente padronize as colunas para media 0 e desvio padrao 1.
def preProcessingData():
  fileData = '//home//nathana//AM//exercicio3//secom.data.txt'
  fileClass = '//home//nathana//AM//exercicio3//secom_labels.data.txt'

  # Ler dois arquivos pedido
  data = open(fileData, 'r')
  dataClass = open(fileClass, 'r')

  X = []
  Y = []
  #  Transforma vetor com tipo nan e float
  for i,row in enumerate(data):
     udateData = []
     [udateData.append(np.nan if c == 'NaN' else float(c)) for c in row.split()]
     X.append(udateData)

  # Elimina a coluna com data e guarda apenas a classe
  for i, row in enumerate(dataClass):
      Y.append(row.split()[0])

  # Substitua os dados faltantes pela media da coluna (imputacao pela media)
  imp = Imputer(missing_values=np.nan, strategy='mean', axis=0)
  X = imp.fit_transform(X)

  X = np.array(X)
  Y = np.array(Y)

  #  Finalmente padronize as colunas para media 0 e desvio padrao 1
  #  preprocessing.scale  realiza essa acao
  X = preprocessing.scale(X)

  # Close Files
  data.close()
  dataClass.close()

  return X, Y
```

4. Main

```
def main():
  X, Y = preProcessingData()

  ### KNN
  #  Para o kNN, faca um PCA que mantem 80% da variancia. Busque os valores do k entre os valores 1, 5, 11, 15, 21, 25..
  parameters = {'n_neighbors': [ 1, 5, 11, 15, 21, 25]}
  mean_knn = kfoldExterno(parameters,X,Y,'KNN')
  print('KNN:',mean_knn)

  #### SVM
  # Para o SVM RBF teste para C=2**(-5), 2**(0), 2**(5), 2**(10) e gamma= 2**(-15) 2**(-10) 2**(-5) 2**(0) 2**(5)
  parameters ={'C': [2**(-5), 2**(0), 2**(5), 2**(10)], 'gamma': [ 2**(-15), 2**(-10), 2**(-5), 2**(0), 2**(5)]}
  mean_svm = kfoldExterno(parameters, X, Y, 'SVM')
  print('SVM:',mean_svm)

  #### Rede neural
  # Para a rede neural, teste com 10, 20, 30 e 40 neuronios na camada escondida.
  parameters ={ 'hidden_layer_sizes' :[10, 20, 30 ,40 ],'max_iter' : [400]}
  mean_rn = kfoldExterno(parameters, X, Y, 'RN')
  print('RN:',mean_rn)

  #### Random Forest
  #  Para o RF, teste com mtry ou n_featrues = 10, 15, 20, 25 e ntrees = 100, 200, 300 e 400..
  parameters ={ 'max_features' : [10, 15, 20, 25], 'n_estimators': [100, 200, 300 ,400]}
  mean_rf = kfoldExterno(parameters, X, Y, 'RF')
  print('RF:',mean_rf)

  ### Gradient Boosting Machine
  # Para o GBM (ou XGB) teste para numero de arvores = 30, 70, e 100, com learning rate de 0.1 e 0.05
  parameters = {'learning_rate': [0.1,0.05], 'max_depth': [5], 'n_estimators':[30, 70,100]}
  mean_gbm = kfoldExterno(parameters, X, Y, 'GBM')
  print('GBM:', mean_gbm)

if __name__ == '__main__':
  main()
```

5. Saídas e Respostas

Reporte a acuracia de cada algoritmo calculada pelo 5-fold CV externo..

```
('KNN:', 0.9329966256178585)
('SVM:', 0.9336335682930177)
('RN:', 0.790827897640437)
('RF:', 0.9323596829426991)
('GBM:', 0.8425405648528906)
```

OBS: Os três últimos algoritmos possuem variações de resultado cada vez que são rodados.