Relatório do Terceiro Exercício

Délio Gomes Soares - 188947

24/10/2016

1 Atividade Proposta

Use os dados do **dataset SECOM do UCI** O arquivo **secom.data** contem os dados. O arquivo **secom_labels.data** contem (na 1ª coluna) a classe de cada dado.

Usando um 5-fold externo para calcular a acurácia, e um 3-fold interno para a escolha dos hyperparametros, determine qual algorítimo entre *kNN*, *SVM* com *kernel RBF*, *redes neurais*, *Random Forest*, e *Gradient Boosting Machine* tem a maior acurácia.

1. Preprocesse os dados do arquivo:**Substitua os dados faltantes pela media da coluna (imputação pela média).** Finalmente padronize as colunas para media 0 e desvio padrão 1.

2. Para o *kNN*, faça um *PCA* que mantem 80% da variância. Busque os valores do **k** entre os valores 1, 5, 11, 15, 21, 25.

3. 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).

4. Para a *rede neural*, teste com 10, 20, 30 e 40 **neurônios na camada escondida**.

5. Para o *RF*, teste com **mtry** ou **n_featrues** = 10, 15, 20, 25 e **ntrees** = 100, 200, 300 e 400.

6. Para o *GBM* (ou *XGB*) teste para **número de árvores** = 30, 70, e 100, com **learning rate** de 0.1 e 0.05, e **profundidade da árvore** =

5.Você pode tanto usar alguma versão do *gbm* para R ou *SKlearn*, ou usar o *XGBoost* (para ambos).

7. Você não precisam fazer os *loops* da validação cruzada explicitamente. Pode usar as fincões como *tunegrid* (do *caret*) ou *tuneParams* (do *mlr*) ou *GridSearchCV* do *SKlearn*.

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

2 Solução

2.1 Resposta

A acurácia média de cada algoritmo foi:

1. kNN: 92,98%
2. SVM: 93,36%
3. Rede Neural: 78,96%
4. Random Forest: 93,36%
5. GBM: 84,44%

2.2 Saída do código

A saída produzida pelo código em *python* com a acurácia média de cada algoritmo.

```
Acuracia media de cada um dos Classificadores
        kNN:    0.929811912242
        SVM:    0.933633568293
        NN:     0.792793904406
        RF:     0.932996625618
        GBM:    0.843163249759
```

2.3 Código fonte em *python*

```
# -*- coding: utf-8 -*-
"""
Exercicio 3
Autor: Delio Gomes Soares
RA: 188947
Data: 24/10/2016
"""
import numpy as np
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
from sklearn.decomposition import PCA
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import Imputer
from sklearn import preprocessing

#------------------------------------------------------------#
data = np.genfromtxt('secom.txt', delimiter=' ')
data_labels = np.genfromtxt('secom_labels.txt', delimiter=' ')
labels = data_labels[:,0]
#------------------------------------------------------------#
def featureImputation(X):
    imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
    X_new = imp.fit_transform(X)
    return X_new
#------------------------------------------------------------#
def featureStandardization(X):
    X_new = preprocessing.scale(X)
    return X_new
#------------------------------------------------------------#
def PCATransform(X):
    pca = PCA(n_components=0.80)
    XTransform = pca.fit_transform(X)
    return XTransform
#------------------------------------------------------------#
def knn(X, y):
    neigh = KNeighborsClassifier()
    parameters = {'n_neighbors':[1, 5, 11, 15, 21, 25]}
    best_n = 0
    clf = GridSearchCV(neigh, parameters, cv=3)
```

```
    clf.fit(X, y)
    best_n = clf.best_estimator_.n_neighbors
    return best_n
#-------------------------------------------------------------#
def svm(X, y):
    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]}
    best_c = best_gamma = 0
    clf = GridSearchCV(svc, parameters, cv=3)
    clf.fit(X, y)
    best_c = clf.best_estimator_.C
    best_gamma = clf.best_estimator_.gamma
    return best_c, best_gamma
#-------------------------------------------------------------#
def neuralNetwork(X, y):
    mlp = MLPClassifier(solver='lbfgs')
    parameters = {'hidden_layer_sizes':[10, 20, 30, 40]}
    best_neuron = 0
    clf = GridSearchCV(mlp, parameters, cv=3)
    clf.fit(X, y)
    best_neuron = clf.best_estimator_.hidden_layer_sizes
    return best_neuron
#-------------------------------------------------------------#
def randomForest(X, y):
    rf = RandomForestClassifier()
    parameters = {'max_features': [10, 15, 20, 25],
                  'n_estimators': [100, 200, 300, 400]}
    best_max = 0
    best_nEst = 0
    clf = GridSearchCV(rf, parameters, cv=3)
    clf.fit(X, y)
    best_max = clf.best_estimator_.max_features
    best_nEst = clf.best_estimator_.n_estimators
    return best_max, best_nEst
#-------------------------------------------------------------#
def gbm(X, y):
    gbmc = GradientBoostingClassifier(max_depth=5)
    parameters = {'learning_rate': [0.1, 0.05], 'n_estimators':
    [30, 70, 100]}
    best_lR = best_nEst = 0
    clf = GridSearchCV(gbmc, parameters, cv=3)
    clf.fit(X, y)
    best_lR = clf.best_estimator_.learning_rate
    best_nEst = clf.best_estimator_.n_estimators
```

```
    return best_lR, best_nEst
#-------------------------------------------------------------#
def main(X, y):
    skf5PCA = StratifiedKFold(n_splits=5)
    XPCA = PCATransform(X)

    resultKNN = 0
    for train_indexInt, test_indexInt in skf5PCA.split(XPCA, y):
        n_Neig = knn(PCATransform(XPCA[train_indexInt]), y[
    train_indexInt])
        neigh = KNeighborsClassifier(n_neighbors= n_Neig)
        neigh.fit(XPCA[train_indexInt], y[train_indexInt])
        resultKNN += neigh.score(XPCA[test_indexInt], y[
    test_indexInt])

    skf5 = StratifiedKFold(n_splits=5)
    resultSVM = resultNeural = resultRF = resultGBM = 0
    for train_indexInt, test_indexInt in skf5.split(X, y):

        C_svm, Gamma_svm = svm(X[train_indexInt], y[
    train_indexInt])
        neuron = neuralNetwork(X[train_indexInt], y[
    train_indexInt])
        maxFea, nEst = randomForest(X[train_indexInt], y[
    train_indexInt])
        lR, nEstimators = gbm(X[train_indexInt], y[
    train_indexInt])

        svc = SVC(kernel='rbf',C= C_svm, gamma= Gamma_svm)
        svc.fit(X[train_indexInt], y[train_indexInt])
        resultSVM += svc.score(X[test_indexInt], y[test_indexInt
    ])

        mlp = MLPClassifier(solver='lbfgs', hidden_layer_sizes=
    neuron)
        mlp.fit(X[train_indexInt], y[train_indexInt])
        resultNeural += mlp.score(X[test_indexInt], y[
    test_indexInt])

        rf = RandomForestClassifier(max_features= maxFea,
                                    n_estimators= nEst)
        rf.fit(X[train_indexInt], y[train_indexInt])
        resultRF += rf.score(X[test_indexInt], y[test_indexInt])

        gbmc = GradientBoostingClassifier(max_depth=5,
```

```
                                                learning_rate = lR,
125                                             n_estimators =
    nEstimators)
          gbmc.fit(X[train_indexInt], y[train_indexInt])
127       resultGBM += gbmc.score(X[test_indexInt], y[
    test_indexInt])

129     print("Acuracia media de cada um dos Classificadores")
      mediaKNN = resultKNN/5
131     print("\tkNN:\t", mediaKNN)
      mediaSVM = resultSVM/5
133     print("\tSVM:\t", mediaSVM)
      mediaNeural = resultNeural/5
135     print("\tNN:\t", mediaNeural)
      mediaRF = resultRF/5
137     print("\tRF:\t", mediaRF)
      mediaGBM = resultGBM/5
139     print("\tGBM:\t", mediaGBM)

141 dataImp = featureImputation(data)
  dataStan = featureStandardization(dataImp)
143 main(dataStan, labels)
```