Exercicio 3

Diego Fernandez Merjildo

October 24, 2016

1 Descrição do exercicio

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 accuracia, e um 3-fold interno para a escolha dos hyperparametros, determine qual algoritimo entre kNN, SVM com kernel RBF, redes neurais, Random Forest, e Gradient Boosting Machine tem a maior acuracia.

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 padrao

2. Para o kNN, faça um PCA que mantem 80% da variancia. 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 neuronios 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 numero de $arvores = 30$, 70, e 100, com learning rate de 0.1 e 0.05, e profundidade da $arvore = 5$.Voce pode tanto usar alguma versao do gbm para R ou SKlearn, ou usar o XGBoost (para ambos).

7. Voce nao precisam fazer os loops da validacao cruzada explicitamente. Pode usar as funcoes como tunegrid (do caret) ou tuneParams (do mlr) ou GridSearchCV do SKlearn..

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

2 Resultados

Os resultados foram obtidos rodando o script apresentado na seção 3. No script usamos um k-fold estratificado de 5 externo (para obter a acuracia) e um outro k-fold estratificado de 3 interno (para obter os parametros).

Rodando o script em python obtemos os seguintes resultados:

```
1
2  ---- PCA for kNN - Choose components number ----
3  Variance : 0.8
4  PCA dimension: 90 with variance = 0.800134743613
5  --- kNN ---
6  Acuracia:0.93359083412
7  Valor final K (K=25)
8
9  --- SVM ---
10 Acuracia:0.929117990669
11 Valor final hiperparametros (C=0.03125, Gamma=3.0517578125e-05)
12
13 --- Neural Network ---
14 Acuracia:0.797816161135
15 Valor final parametros (Neurons=40)
16
17 --- RF ---
18 Acuracia:0.929765095103
19 Valor final parametros (Feats=15, Trees=100)
20
21 --- GBM ---
22 Acuracia:0.845037435449
23 Valor final parametros (Learn Rate=0.1, Trees=100)
```

Segundo os resultados apresentados, podemos indicar que o algoritmo kNN teve ligeramente uma melhor acuracia.

3 Codigo fonte em python

Listing 1: Codigo em Python

```
1
2  #!/usr/bin/python
3
4  import sys,os,csv
5  import pandas
6  import numpy as np
7  import math
```

```
8  from sklearn.model_selection import StratifiedKFold
9  from sklearn import svm as SVM
10 from sklearn.decomposition import PCA
11 from sklearn.neighbors import KNeighborsClassifier
12 from sklearn.ensemble import RandomForestClassifier
13 from sklearn.ensemble import GradientBoostingClassifier
14 from sklearn.neural_network import MLPClassifier
15
16 datFileName="secom.data"
17 labelsFileName="secom_labels.data"
18 dirPath=os.path.dirname(os.path.realpath(__file__))
19 classList=[]
20 data=[]
21
22 def load_data(fileName):
23     raw_data = open(fileName, 'rb')
24     rawData = pandas.read_csv(raw_data, delimiter=" ")
25     return rawData.values
26
27 def getData(rawData):
28     #print "\n---- Getting data from File ----"
29     lineNum = rawData.shape[0]
30     colNum = rawData.shape[1]
31     data = np.array(rawData[0:lineNum, 0:colNum-1])
32     for i in range(lineNum):
33         classList.append(rawData[i][colNum - 1])
34     return [data, np.array(classList) ]
35
36 def getLabels(fileName):
37     labelData = load_data(dirPath + "/" + fileName)
38     labels = labelData[:,0].clip(min=0)
39     return np.array(labels)
40
41 def svm_intern_folds(data_train, data_test, labelsTrain, labelsTest↩
   ):
42     acxmax = 0
43     c_max=0
44     gamma_max=0
45     for c in [2**(-5), 1, 2**(5), 2**(10)]:
46         for gamm in [2**(-15), 2**(-10), 2**(-5), 1, 2**5]:
47             svm = SVM.SVC(C = c, gamma = gamm)
48             svm.fit(data_train, labelsTrain)
49             accuracy = svm.score(data_test, labelsTest)
50             if accuracy > acxmax:
51                 acxmax = accuracy
52                 c_max = c
53                 gamma_max = gamm
54     return [acxmax, c_max, gamma_max]
```

```
55
56
57  def chooseComponentsNumber(matrix, percent):
58      print "\n---- PCA - Choose components number ----"
59      print "Variance :",  percent
60      mat = np.matrix(matrix) * np.matrix(matrix).transpose()
61      U,S,V = np.linalg.svd(mat)
62      #print U.shape, S.shape, V.shape
63      s_sum_all = sum(S)
64      totalComponents = matrix.shape[1]
65      num = totalComponents
66      for i in range(totalComponents):
67          if sum(S[0:i]) / s_sum_all >= percent :
68              print "PCA dimension:",i ,"with variance =", sum(S[0:i↩
                    ]) / s_sum_all
69              num = i
70              break
71      return num
72
73  def applyPCA(data, numComponents):
74      pca = PCA(n_components=numComponents)
75      pcaData = pca.fit_transform(data)
76      return pcaData
77
78  def knn_intern_folds(data_train, data_test, labels_train, ↩
        labels_test):
79      acxmax = 0
80      cores = 4
81      k_value = 0
82      for k in [1, 5, 11, 15, 21, 25]:
83          knn = KNeighborsClassifier(n_neighbors = k, n_jobs = cores)
84          knn.fit(data_train, labels_train)
85          accuracy = knn.score(data_test, labels_test)
86          if accuracy > acxmax:
87              acxmax = accuracy
88              k_value = k
89      return [acxmax, k]
90
91  def neural_intern_folds(data_train, data_test, labels_train, ↩
        labels_test):
92      # 10, 20, 30 e 40 neuronios na camada escondida.
93      acxmax = 0
94      cores = 4
95      n_value = 0
96      for n in [10, 20, 30, 40]:
97          clf = MLPClassifier(hidden_layer_sizes=(n,), solver='lbfgs'↩
                )
98          clf.fit(data_train, labels_train)
```

```
99          accuracy = clf.score(data_test, labels_test)
100         if accuracy > acxmax:
101             acxmax = accuracy
102             n_value = n
103     return [acxmax, n]
104
105 def rf_intern_folds(data_train, data_test, labels_train, ↩
        labels_test):
106     # teste com mtry ou n_featrues = 10, 15, 20, 25 e ntrees = 100,↩
            200, 300 e 400
107     acxmax = 0
108     n_feats = 0
109     n_trees = 0
110     for feat in [10, 15, 20, 25]:
111         for trees in [100, 200, 300, 400]:
112             clf = RandomForestClassifier (max_features = feat, ↩
                    n_estimators = trees)
113             clf.fit(data_train, labels_train)
114             accuracy = clf.score(data_test, labels_test)
115             #print "first acc:", accuracy
116             if accuracy > acxmax:
117                 acxmax = accuracy
118                 n_feats = feat
119                 n_trees = trees
120     return [acxmax, n_feats, n_trees]
121
122 def gbm_intern_folds(data_train, data_test, labels_train, ↩
        labels_test):
123     ##  numero de arvores = 30, 70, e 100, com learning rate de 0.1↩
            e 0.05, e profundidade da arvore=5.
124     acxmax = 0
125     n_learn_rate = 0
126     n_trees = 0
127     depth_tree = 5
128     for trees in [30, 70, 100]:
129         for learn_rate in [0.1, 0.05]:
130             clf = GradientBoostingClassifier (n_estimators = trees,↩
                        learning_rate = learn_rate, max_depth = depth_tree↩
                    )
131             clf.fit(data_train, labels_train)
132             accuracy = clf.score(data_test, labels_test)
133             #print "first acc:", accuracy
134             if accuracy > acxmax:
135                 acxmax = accuracy
136                 n_trees = trees
137                 n_learn_rate = learn_rate
138     return [acxmax, n_learn_rate, n_trees]
139
```

```
140 ## Data preprocessing
141 def data_preprocess(fileName):
142     rawdata = load_data(dirPath + "/" + fileName)
143     ## column mean
144     column_mean = np.nanmean(np.array(rawdata), axis=0)
145     ## Nan values index
146     nan_indexes = np.where(np.isnan(rawdata))
147     ## Replace Nan values
148     rawdata[nan_indexes] = np.take(column_mean, nan_indexes[1])
149     ## Standarize each column individually
150     rawdata = (rawdata - np.mean(rawdata, axis=0)) / np.std(rawdata↩
        , axis=0)
151     rawdata = np.nan_to_num(rawdata)
152     return rawdata
153
154 def run_folds( alg, data, labels):
155     print "--- %s ---" % alg
156     final_accuracy = 0
157     params_final = [0.0, 0.0]
158     skf = StratifiedKFold(n_splits=5)
159     for train_index, test_index in skf.split(data, labels):
160         new_data_train = data[train_index]
161         new_data_test = data[test_index]
162         new_labels_train = labels[train_index]
163         new_labels_test = labels[test_index]
164         acx = 0
165         skf_intern = StratifiedKFold(n_splits=3)
166         for intern_train_index, intern_test_index in skf_intern.↩
            split(new_data_train, new_labels_train):
167             intern_data_train = new_data_train[intern_train_index]
168             intern_data_test = new_data_train[intern_test_index]
169             intern_labels_train = new_labels_train[↩
                intern_train_index]
170             intern_labels_test = new_labels_train[intern_test_index↩
                ]
171             params = get_intern_folds (alg, intern_data_train, ↩
                intern_data_test, intern_labels_train, ↩
                intern_labels_test)
172             if params[0] > acx:
173                 acx = params[0]
174                 params_final[0] = params[1]
175                 if len(params) > 2:
176                     params_final[1] = params[2]
177
178         final_accuracy = final_accuracy + model_score(alg, ↩
            params_final,
179                                                       new_data_train↩
                                                          ,
```

```
                                                    new_labels_train,
                                                    new_data_test,
                                                    new_labels_test)
    final_accuracy = final_accuracy / 5
    print_results(alg, final_accuracy, params_final)

def model_score(alg, params, new_data_train, new_labels_train, new_data_test, new_labels_test):
    if 'svm' == alg:
        svm_model = SVM.SVC(C = params[0], gamma = params[1])
        svm_model.fit(new_data_train, new_labels_train)
        return svm_model.score(new_data_test, new_labels_test)
    elif 'knn' == alg:
        knn = KNeighborsClassifier(n_neighbors = params[0], n_jobs = 4)
        knn.fit(new_data_train, new_labels_train)
        return knn.score(new_data_test, new_labels_test)
    elif 'neural' == alg:
        clf = MLPClassifier(hidden_layer_sizes=(params[0],), solver='lbfgs')
        clf.fit(new_data_train, new_labels_train)
        return clf.score(new_data_test, new_labels_test)
    elif 'rf' == alg:
        clf = RandomForestClassifier (max_features = params[0], n_estimators = params[1])
        clf.fit(new_data_train, new_labels_train)
        return clf.score(new_data_test, new_labels_test)
    elif 'gbm' == alg:
        clf = GradientBoostingClassifier (learning_rate = params[0], n_estimators = params[1], max_depth = 5)
        clf.fit(new_data_train, new_labels_train)
        return clf.score(new_data_test, new_labels_test)

def get_intern_folds (alg, data_train, data_test, labels_train, labels_test):
    if 'svm' == alg:
        return svm_intern_folds(data_train, data_test, labels_train, labels_test)
    elif 'knn' == alg:
        return knn_intern_folds(data_train, data_test, labels_train, labels_test)
    elif 'neural' == alg:
        return neural_intern_folds(data_train, data_test, labels_train, labels_test)
    elif 'rf' == alg:
```

```
216         return rf_intern_folds(data_train, data_test, labels_train,↩
                labels_test)
217     elif 'gbm' == alg:
218         return gbm_intern_folds(data_train, data_test, labels_train↩
               , labels_test)
219
220 def print_results(alg, final_accuracy, params):
221     if 'svm' == alg:
222         print("Acuracia:%s" % final_accuracy)
223         print("Valor final hiperparametros (C=%s, Gamma=%s)" % (↩
               params[0], params[1]) )
224     elif 'knn' == alg:
225         print("Acuracia:%s" % final_accuracy)
226         print("Valor final K (K=%s)" % (params[0]))
227     elif 'neural' == alg:
228         print("Acuracia:%s" % final_accuracy)
229         print("Valor final parametros (Neurons=%s)" % (params[0]) )
230     elif 'rf' == alg:
231         print("Acuracia:%s" % final_accuracy)
232         print("Valor final parametros (Feats=%s, Trees=%s)" % (↩
               params[0], params[1]) )
233     elif 'gbm' == alg:
234         print("Acuracia:%s" % final_accuracy)
235         print("Valor final parametros (Learn Rate=%s, Trees=%s)" % ↩
               (params[0], params[1]))
236
237
238 def PCA_for_knn(data):
239     variance = 80
240     numComponents = chooseComponentsNumber(data, float(variance) / ↩
           100)
241     if numComponents == -1 : print "Invalid components number. Exit↩
           "; return
242     return applyPCA(data, numComponents)
243
244 def main(argv=None):
245     if argv is None:
246         arv = sys.argv
247
248     ## Data pre-processing
249     data = data_preprocess(datFileName)
250     labels = getLabels(labelsFileName)
251     labels = np.array(list(labels[:data.shape[0]]))
252
253     ## kNN , PCA com 80% da variancia
254     pcaData = PCA_for_knn(data)
255     run_folds('knn', pcaData, labels)
256
```

```
    ## SVM RBF
    run_folds('svm', data, labels)

    ## Neural network
    run_folds('neural', data, labels)

    ## RF
    run_folds('rf', data, labels)

    ## GBM
    run_folds('gbm', data, labels)

if __name__ == "__main__":
    sys.exit(main())
```