Exercicio 3

Diego Fernandez Merjildo

October 23, 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  --- kNN--
3  Acuracia:0.93359083412
4  Valor final K (K=25)
5
6  --- SVM ---
7  Acuracia:0.929117990669
8  Valor final hiperparametros (C=0.03125, Gamma=3.0517578125e-05)
9
10 --- Neural Network ---
11 Acuracia:0.797816161135
12 Valor final parametros (Neurons=40)
13
14 --- RF ---
15 Acuracia:0.929765095103
16 Valor final parametros (Feats=15, Trees=100)
17
18 --- GBM ---
19 Acuracia:0.845037435449
20 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.neighbors import KNeighborsClassifier
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier

datFileName="secom.data"
labelsFileName="secom_labels.data"
dirPath=os.path.dirname(os.path.realpath(__file__))
classList=[]
data=[]

def load_data(fileName):
    raw_data = open(fileName, 'rb')
    rawData = pandas.read_csv(raw_data, delimiter=" ")
    return rawData.values

def getData(rawData):
    #print "\n---- Getting data from File ----"
    lineNum = rawData.shape[0]
    colNum = rawData.shape[1]
    data = np.array(rawData[0:lineNum, 0:colNum-1])
    for i in range(lineNum):
        classList.append(rawData[i][colNum - 1])
    return [data, np.array(classList) ]

def getLabels(fileName):
    labelData = load_data(dirPath + "/" + fileName)
    labels = labelData[:,0].clip(min=0)
    return np.array(labels)

def svm_intern_folds(data_train, data_test, labelsTrain, labelsTest↩
    ):
    acxmax = 0
    c_max=0
    gamma_max=0
    for c in [2**(-5), 1, 2**(5), 2**(10)]:
        for gamm in [2**(-15), 2**(-10), 2**(-5), 1, 2**5]:
            svm = SVM.SVC(C = c, gamma = gamm)
            svm.fit(data_train, labelsTrain)
            accuracy = svm.score(data_test, labelsTest)
            if accuracy > acxmax:
                acxmax = accuracy
                c_max = c
                gamma_max = gamm
    return [acxmax, c_max, gamma_max]

def knn_intern_folds(data_train, data_test, labels_train, ↩
    labels_test):
    acxmax = 0
```

```
57      cores = 4
58      k_value = 0
59      for k in [1, 5, 11, 15, 21, 25]:
60          knn = KNeighborsClassifier(n_neighbors = k, n_jobs = cores)
61          knn.fit(data_train, labels_train)
62          accuracy = knn.score(data_test, labels_test)
63          if accuracy > acxmax:
64              acxmax = accuracy
65              k_value = k
66      return [acxmax, k]
67
68  def neural_intern_folds(data_train, data_test, labels_train, ↩
        labels_test):
69      # 10, 20, 30 e 40 neuronios na camada escondida.
70      acxmax = 0
71      cores = 4
72      n_value = 0
73      for n in [10, 20, 30, 40]:
74          clf = MLPClassifier(hidden_layer_sizes=(n,), solver='lbfgs'↩
                )
75          clf.fit(data_train, labels_train)
76          accuracy = clf.score(data_test, labels_test)
77          if accuracy > acxmax:
78              acxmax = accuracy
79              n_value = n
80      return [acxmax, n]
81
82  def rf_intern_folds(data_train, data_test, labels_train, ↩
        labels_test):
83      # teste com mtry ou n_featrues = 10, 15, 20, 25 e ntrees = 100,↩
            200, 300 e 400
84      acxmax = 0
85      n_feats = 0
86      n_trees = 0
87      for feat in [10, 15, 20, 25]:
88          for trees in [100, 200, 300, 400]:
89              clf = RandomForestClassifier (max_features = feat, ↩
                    n_estimators = trees)
90              clf.fit(data_train, labels_train)
91              accuracy = clf.score(data_test, labels_test)
92              #print "first acc:", accuracy
93              if accuracy > acxmax:
94                  acxmax = accuracy
95                  n_feats = feat
96                  n_trees = trees
97      return [acxmax, n_feats, n_trees]
98
```

```
99  def gbm_intern_folds(data_train, data_test, labels_train, labels_test):
100     ##  numero de arvores = 30, 70, e 100, com learning rate de 0.1 e 0.05, e profundidade da arvore=5.
101     acxmax = 0
102     n_learn_rate = 0
103     n_trees = 0
104     depth_tree = 5
105     for trees in [30, 70, 100]:
106         for learn_rate in [0.1, 0.05]:
107             clf = GradientBoostingClassifier (n_estimators = trees, learning_rate = learn_rate, max_depth = depth_tree)
108             clf.fit(data_train, labels_train)
109             accuracy = clf.score(data_test, labels_test)
110             #print "first acc:", accuracy
111             if accuracy > acxmax:
112                 acxmax = accuracy
113                 n_trees = trees
114                 n_learn_rate = learn_rate
115     return [acxmax, n_learn_rate, n_trees]
116
117 ## Data preprocessing
118 def data_preprocess(fileName):
119     rawdata = load_data(dirPath + "/" + fileName)
120     ## column mean
121     column_mean = np.nanmean(np.array(rawdata), axis=0)
122     ## Nan values index
123     nan_indexes = np.where(np.isnan(rawdata))
124     ## Replace Nan values
125     rawdata[nan_indexes] = np.take(column_mean, nan_indexes[1])
126     ## Standarize each column individually
127     rawdata = (rawdata - np.mean(rawdata, axis=0)) / np.std(rawdata, axis=0)
128     rawdata = np.nan_to_num(rawdata)
129     return rawdata
130
131 def run_folds( alg, data, labels):
132     print "--- %s ---" % alg
133     final_accuracy = 0
134     params_final = [0.0, 0.0]
135     skf = StratifiedKFold(n_splits=5)
136     for train_index, test_index in skf.split(data, labels):
137         new_data_train = data[train_index]
138         new_data_test = data[test_index]
139         new_labels_train = labels[train_index]
140         new_labels_test = labels[test_index]
141         acx = 0
```

```
142         skf_intern = StratifiedKFold(n_splits=3)
143         for intern_train_index, intern_test_index in skf_intern.↩
                split(new_data_train, new_labels_train):
144             intern_data_train = new_data_train[intern_train_index]
145             intern_data_test = new_data_train[intern_test_index]
146             intern_labels_train = new_labels_train[↩
                    intern_train_index]
147             intern_labels_test = new_labels_train[intern_test_index↩
                    ]
148             params = get_intern_folds (alg, intern_data_train, ↩
                    intern_data_test, intern_labels_train, ↩
                    intern_labels_test)
149             if params[0] > acx:
150                 acx = params[0]
151                 params_final[0] = params[1]
152                 if len(params) > 2:
153                     params_final[1] = params[2]
154
155         final_accuracy = final_accuracy + model_score(alg, ↩
                params_final,
156                                                       new_data_train↩
                                                          ,
157                                                       new_labels_train↩
                                                          ,
158                                                       new_data_test↩
                                                          ,
159                                                       new_labels_test↩
                                                          )
160     final_accuracy = final_accuracy / 5
161     print_results(alg, final_accuracy, params_final)
162
163 def model_score(alg, params, new_data_train, new_labels_train, ↩
        new_data_test, new_labels_test):
164     if 'svm' == alg:
165         svm_model = SVM.SVC(C = params[0], gamma = params[1])
166         svm_model.fit(new_data_train, new_labels_train)
167         return svm_model.score(new_data_test, new_labels_test)
168     elif 'knn' == alg:
169         knn = KNeighborsClassifier(n_neighbors = params[0], n_jobs ↩
                = 4)
170         knn.fit(new_data_train, new_labels_train)
171         return knn.score(new_data_test, new_labels_test)
172     elif 'neural' == alg:
173         clf = MLPClassifier(hidden_layer_sizes=(params[0],), solver↩
                ='lbfgs')
174         clf.fit(new_data_train, new_labels_train)
175         return clf.score(new_data_test, new_labels_test)
176     elif 'rf' == alg:
```

```
177         clf = RandomForestClassifier (max_features = params[0], ↩
                n_estimators = params[1])
178         clf.fit(new_data_train, new_labels_train)
179         return clf.score(new_data_test, new_labels_test)
180     elif 'gbm' == alg:
181         clf = GradientBoostingClassifier (learning_rate = params↩
                [0], n_estimators = params[1], max_depth = 5)
182         clf.fit(new_data_train, new_labels_train)
183         return clf.score(new_data_test, new_labels_test)
184
185 def get_intern_folds (alg, data_train, data_test, labels_train, ↩
        labels_test):
186     if 'svm' == alg:
187         return svm_intern_folds(data_train, data_test, labels_train↩
                , labels_test)
188     elif 'knn' == alg:
189         return knn_intern_folds(data_train, data_test, labels_train↩
                , labels_test)
190     elif 'neural' == alg:
191         return neural_intern_folds(data_train, data_test, ↩
                labels_train, labels_test)
192     elif 'rf' == alg:
193         return rf_intern_folds(data_train, data_test, labels_train,↩
                 labels_test)
194     elif 'gbm' == alg:
195         return gbm_intern_folds(data_train, data_test, labels_train↩
                , labels_test)
196
197 def print_results(alg, final_accuracy, params):
198     if 'svm' == alg:
199         print("Acuracia:%s" % final_accuracy)
200         print("Valor final hiperparametros (C=%s, Gamma=%s)" % (↩
                params[0], params[1]) )
201     elif 'knn' == alg:
202         print("Acuracia:%s" % final_accuracy)
203         print("Valor final K (K=%s)" % (params[0]))
204     elif 'neural' == alg:
205         print("Acuracia:%s" % final_accuracy)
206         print("Valor final parametros (Neurons=%s)" % (params[0]) )
207     elif 'rf' == alg:
208         print("Acuracia:%s" % final_accuracy)
209         print("Valor final parametros (Feats=%s, Trees=%s)" % (↩
                params[0], params[1]) )
210     elif 'gbm' == alg:
211         print("Acuracia:%s" % final_accuracy)
212         print("Valor final parametros (Learn Rate=%s, Trees=%s)" % ↩
                (params[0], params[1]))
213
```

```
214
215 def main(argv=None):
216     if argv is None:
217         arv = sys.argv
218
219     ## Data pre-processing
220     data = data_preprocess(datFileName)
221     labels = getLabels(labelsFileName)
222     labels = np.array(list(labels[:data.shape[0]]))
223
224     ## kNN , PCA com 80% da variancia
225     run_folds('knn', data, labels)
226
227     ## SVM RBF
228     run_folds('svm', data, labels)
229
230     ## Neural network
231     run_folds('neural', data, labels)
232
233     ## RF
234     run_folds('rf', data, labels)
235
236     ## GBM
237     run_folds('gbm', data, labels)
238
239 if __name__ == "__main__":
240     sys.exit(main())
```