MO444 – Aprendizado de Máquina

Edgar Rodolfo Quispe Condori - RA 192617

May 12, 2017

Questão 1. *Pre-processamento: faça os pre-processamentos do exercicio 3.*

O código:

```
def preprocess_data():
  #read cvs file
  file_obj = open("abalone.csv", "rt")
  reader = csv.reader(file_obj)
  data = []
  for row in reader:
    data.append(row)

  data = np.array(data)

  #extract first column and convert data to floats
  first_column = [row[0] for row in data]
  data = np.array([row[1::] for row in   data]).astype(np.float)

  #convert first column using one-hot-encoding and restore it to data
  first_column = np.array(pd.get_dummies(first_column))
  data = np.hstack((first_column, data))

  #separe and transform last column
  labels = data[:, -1]
  labels[labels <= 13] = 0
  labels[labels > 13] = 1
  data = data[:, 0:-1]

  #standardize data
  scaler = preprocessing.StandardScaler().fit(data)
  data = scaler.transform(data)

  return data, labels
```

Questão 2. *Para o kNN, faça um PCA que mantem 90% da variancia. Busque os valores do k entre os valores 1, 5, 11, 15, 21, 25.*

Os resultados para cada um dos folds são:

Fold	n_neighbors	Accuracy
1	21	0.88038277512
2	25	0.882775119617
3	21	0.882634730539
4	21	0.881437125749
5	21	0.882634730539

Isso da uma media de 0.882 para o k-Neastest Neighbors, ele é o quinto melhor classificador.
O código:

```
def kNN_evaluation(x, y):
    #PCA
    x = PCA(n_components = 0.9).fit_transform(x)

    print "K Neastest Neighbors"
    params = {'n_neighbors' : [1, 5, 11, 15, 21, 25]}
    #create stratified k-folds
    kf = StratifiedKFold(n_splits = 5, random_state = 1)
    kf.get_n_splits(x, y)

    accuracy_mean = 0
    for train_index, test_index in kf.split(x, y):
        #set train and test data
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        #run gridSearch
        gridSearch = GridSearchCV(KNeighborsClassifier(), params, cv = 3)
        gridSearch.fit(x_train, y_train)

        #create model with best params
        clf = KNeighborsClassifier(**gridSearch.best_params_).fit(x_train, y_train)

        #test created model
        nested_accuracy = clf.score(x_test, y_test)
        accuracy_mean += nested_accuracy
        print ("\tn_neighbors: ",gridSearch.best_params_['n_neighbors'],
        " nested acc:", nested_accuracy)

    print ("Mean acc:", round(accuracy_mean/5.0, 3))
```

Questão 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).*

Os resultados para cada um dos folds são:

Fold	C	gamma	Accuracy
1	1024	0.0009765625	0.898325358852
2	32	0.03125	0.885167464115
3	1024	0.0009765625	0.898203592814
4	1024	0.0009765625	0.893413173653
5	1024	0.0009765625	0.88502994012

Isso da uma media de 0.892 para o SVM, ele é o segundo melhor classificador.
O código:

```
def svm_evaluation(x, y):
    print "Support Vector Machine"
    params = {'C' : [2**(-5), 2**(0), 2**(5), 2**(10)], 'gamma': [2**(-15),
    2**(-10), 2**(-5), 2**(0), 2**(5)]}
    #create stratified k-folds
    kf = StratifiedKFold(n_splits = 5, random_state = 1)
    kf.get_n_splits(x, y)

    accuracy_mean = 0
    for train_index, test_index in kf.split(x, y):
        #set train and test data
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        #run gridSearch
        gridSearch = GridSearchCV(SVC(random_state = 1), params, cv = 3)
        gridSearch.fit(x_train, y_train)

        #create model with best params
        clf = SVC(**gridSearch.best_params_).fit(x_train, y_train)
```

```
        #test created model
        nested_accuracy = clf.score(x_test, y_test)
        accuracy_mean += nested_accuracy
        print ("\tC: ",gridSearch.best_params_['C'], "gamma: ",
        gridSearch.best_params_['gamma']," nested acc:", nested_accuracy)

    print ("Mean acc:", round(accuracy_mean/5.0, 3))
```

Questão 4. *Para a rede neural, teste com 3, 7, 10, e 20 neuronios na camada escondida.*

Os resultados para cada um dos folds são:

Fold	hidden_layer_sizes	Accuracy
1	(7,)	0.891148325359
2	(3,)	0.893540669856
3	(3,)	0.902994011976
4	(10,)	0.893413173653
5	(20,)	0.895808383234

Isso da uma media de 0.895 para a rede neural, ele é o melhor classificador.
O código:

```
def mlp_evaluation(x, y):
    print "Multi Layer Perceptron"
    params = {'hidden_layer_sizes' : [(3,), (7, ), (10, ), (20, )]}
    #create stratified k-folds
    kf = StratifiedKFold(n_splits = 5, random_state = 1)
    kf.get_n_splits(x, y)

    accuracy_mean = 0
    for train_index, test_index in kf.split(x, y):
        #set train and test data
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        #run gridSearch
        gridSearch = GridSearchCV(MLPClassifier(random_state  = 1), params, cv = 3)
        gridSearch.fit(x_train, y_train)

        #create model with best params
        clf = MLPClassifier(**gridSearch.best_params_).fit(x_train, y_train)

        #test created model
        nested_accuracy = clf.score(x_test, y_test)
        accuracy_mean += nested_accuracy
        print ("\thidden_layer_sizes: ",gridSearch.best_params_['hidden_layer_sizes'],
        "nested acc:", nested_accuracy)

    print ("Mean acc:", round(accuracy_mean/5.0, 3))
```

Questão 5. *Para o RF, teste max_features = 2, 3, 5, 7 e n_estimators = 100, 200, 400 e 800.*

Os resultados para cada um dos folds são:

Fold	n_estimators	max_features	Accuracy
1	100	2	0.877990430622
2	800	5	0.894736842105
3	200	2	0.901796407186
4	800	5	0.88502994012
5	100	3	0.891017964072

Isso da uma media de 0.890 para o Random Forest, ele é o terceiro melhor classificador.
O código:

```
def rf_evaluation(x, y):
    print "Random Forest"
    params = {'n_estimators' : [100, 200, 400, 800], 'max_features' : [2, 3, 5, 7]}
    #create stratified k-folds
    kf = StratifiedKFold(n_splits = 5, random_state = 1)
    kf.get_n_splits(x, y)

    accuracy_mean = 0
    for train_index, test_index in kf.split(x, y):
        #set train and test data
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        #run gridSearch
        gridSearch = GridSearchCV(RandomForestClassifier(random_state  = 1), params, cv = 3)
        gridSearch.fit(x_train, y_train)

        #create model with best params
        clf = RandomForestClassifier(**gridSearch.best_params_).fit(x_train, y_train)

        #test created model
        nested_accuracy = clf.score(x_test, y_test)
        accuracy_mean += nested_accuracy
        print ("\tn_estimators: ",gridSearch.best_params_['n_estimators'], "max_features: ",gridSearch.best_pa

    print ("Mean acc:", round(accuracy_mean/5.0, 3))
```

Questão 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 a versao do SKlearn ou o XGBoost.*

Os resultados para cada um dos folds são:

Fold	n_estimators	learning_rate	max_depth	Accuracy
1	100	0.1	5	0.872009569378
2	70	0.1	5	0.894736842105
3	70	0.05	5	0.906586826347
4	100	0.1	5	0.881437125749
5	70	0.05	5	0.891017964072

Isso da uma media de 0.893 para o GBM, ele é o quarto melhor classificador.
O código:

```
def gb_evaluation(x, y):
    print "Gradient Boosting Machine"
    params = {'n_estimators' : [30, 70, 100], 'learning_rate' : [0.1, 0.05], 'max_depth': [5]}
    #create stratified k-folds
    kf = StratifiedKFold(n_splits = 5, random_state = 1)
    kf.get_n_splits(x, y)

    accuracy_mean = 0
    for train_index, test_index in kf.split(x, y):
        #set train and test data
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        #run gridSearch
        gridSearch = GridSearchCV(GradientBoostingClassifier(random_state  = 1), params, cv = 3)
        gridSearch.fit(x_train, y_train)

        #create model with best params
        clf = GradientBoostingClassifier(**gridSearch.best_params_).fit(x_train, y_train)

        #test created model
        nested_accuracy = clf.score(x_test, y_test)
        accuracy_mean += nested_accuracy
        print ("\tn_estimators: ",gridSearch.best_params_['n_estimators'], "learning_rate: ",
        gridSearch.best_params_['learning_rate'], "max_depth: ",
```

```
        gridSearch.best_params_['max_depth'], " nested acc:", nested_accuracy)

    print ("Mean acc:", round(accuracy_mean/5.0, 3))
```

Questão 7. *Reporte a acuracia (com 3 digitos) de cada algoritmo calculada pelo 5-fold CV externo e para o algoritmo com maior accuracia, reporte o valor dos hiperparamtertos obtidos para gerar o classificador final.*

As acurácias promedio obtidas foram:

Algorithm	**Mean Accuracy**
kNN	0.882
SVM	0.892
MLP	0.895
RF	0.890
GBM	0.889

O melhor classificador é o Multilayer Perceptron, a diferença com o segundo melhor classificador é so 0.003 e com o pior classificador é 0.013.

Treinando o MLP para gerar o classificador final tem-se o hiperparámetro 'hidden_layer_sizes': 20

O código:

```
def train_best_classifier(x, y):
    #params for gridSearch
    params = {'hidden_layer_sizes' : [(3,), (7, ), (10, ), (20, )]}

    gridSearch = GridSearchCV(MLPClassifier(random_state = 1), params, cv = 3)
    gridSearch.fit(x, y)

    print gridSearch.best_params_
    print gridSearch.cv_results_['mean_test_score']
    #train final classifier
    return MLPClassifier(**gridSearch.best_params_).fit(x, y)
```