MO444 – Aprendizado de Máquina

Edgar Rodolfo Quispe Condori - RA 192617

April 18, 2017

Questão 1. *Pre-processamento:*

- *Leia o arquivo abalone do exercicio 2*
- *Faça o preprocessamento do atributo categorico e do atributo de saida como no exercicio 2*
- *Estandardize todos os atributos numéricos. Voce pode estardartizar todo o arquivo de uma vez. Como discutimos em aula esse não é a coisa 100*

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. *Logistic regression*

- *Faça o logistic regression com* $C = 10^{-1,0,1,2,3}$. *O loop externo deve ser um 5-fold CV estratificado. O loop interno para a escolha do hiperparametro deve ser um 3-fold estratificado.*
- *Voce tem que fazer o loop interno explicitamente, usando StratifiedKFold e não funções como GridSearchCV*
- *Qual a acurácia do LR com a melhore escolha de parametros (para cada fold)?*

Os resultados para cada um dos folds são:

Fold	Accuracy	C
1	0.895933014354	1
2	0.894736842105	1000
3	0.899401197605	100
4	0.900598802395	1000
5	0.895808383234	100

Isso da uma media de 0.897295647939 para o Logistic Regression, ele é o segundo melhor classificador. A diferencia da acurácia com os outros classificadores é muito pequena.

O código:

```
def logistic_regressor_evaluation(x, y):
  C_space = [10**i for i in range(-1, 4) ]

  #create stratified k-folds
  skFolds = model_selection.StratifiedKFold(n_splits = 5, random_state = 1)
  skFolds.get_n_splits(x, y)

  accuracy_mean = 0

  for train_index, test_index in skFolds.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]

    #create stratified k-folds for nested loop
    nested_skFolds = model_selection.StratifiedKFold(n_splits = 3, random_state = 1)
    nested_skFolds.get_n_splits(x_train, y_train)

    best_c = 0
    best_accuracy = 0

    for c in C_space:

      c_accuracy = 0.0

      for nested_train_index, nested_test_index in nested_skFolds.split(x_train, y_train):

        #set nested train and test data
        nested_x_train, nested_x_test = x_train[nested_train_index], x_train[nested_test_index]
        nested_y_train, nested_y_test = y_train[nested_train_index], y_train[nested_test_index]


        #create and train the nested logistic Regressor
        nested_clf = LogisticRegression(C=c, random_state=1)
        nested_clf.fit(nested_x_train, nested_y_train)

        #test the trained model
        nested_pred = nested_clf.predict(nested_x_test)
        nested_accuracy = metrics.accuracy_score(nested_y_test, nested_pred)

        c_accuracy += nested_accuracy

      c_accuracy /= 3.0

      print("\tnested accuracy", c_accuracy, c)

      #update best accuracy
      if best_accuracy <= c_accuracy:
        best_accuracy =  c_accuracy
        best_c = c

    #train logistic Regressor with the best C
    clf = LogisticRegression(C=best_c, random_state=1)
    clf.fit(x_train, y_train)

    #test the trained model
```

```
    pred = clf.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, pred)

    print("LR KF Accuracy", accuracy, best_c)

    #acumulate accuracy
    accuracy_mean += accuracy

  print("Logistic regressor mean accuracy", accuracy_mean/5.0)
```

Questão 3. *Linear SVM*

- *Faça o LinearSVM com $C = 10^{-1,0,1,2,3}$. O loop externo deve ser um 5-fold estratificado. O loop interno um 3-fold estratificado. Neste caso voce nao precisa fazer o 3 fold explicitamente, voce pode usar o GridSearchCV.*
- *Qual a acurácia do LinearSVM com a melhor escolha de C?*

Os resultados para cada um dos folds são:

Fold	Accuracy	C
1	0.895933014354	10
2	0.895933014354	10
3	0.901796407186	10
4	0.899401197605	10
5	0.893413173653	0.1

Isso da uma media de 0.89777383033 para o SVM, ele é o melhor classificador por uma diferencia pequena.

O código:

```
def SVM_evaluation(x, y):
  parameters = {'C' : [10**i for i in range(-1, 4) ]}

  #create stratified k-folds
  skFolds = model_selection.StratifiedKFold(n_splits = 5, random_state = 1)
  skFolds.get_n_splits(x, y)

  accuracy_mean = 0

  for train_index, test_index in skFolds.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]

    #create nested SVM
    nested_svm = svm.LinearSVC(random_state = 1)

    #create nested loop and get best parametes
    nested_clf = GridSearchCV(nested_svm, parameters, cv = 3)
    nested_clf.fit(x_train, y_train)

    print("\tNested Accuracy", nested_clf.best_score_, nested_clf.best_params_)

    #train SVM with the best parameters
    clf = svm.LinearSVC(C = nested_clf.best_params_['C'], random_state = 1)
    clf.fit(x_train, y_train)

    #test the trained model
    pred = clf.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, pred)

    print("SVM KF Accuracy", accuracy, nested_clf.best_params_['C'])
```

```
    #acumulate accuracy
    accuracy_mean += accuracy

  print("SVM mean accuracy", accuracy_mean/5.0)
```

Questão 4. *LDA*

- *Faça o LDA. Reporte a acuracia*

Os resultados para cada um dos folds são:

Fold	Accuracy
1	0.893540669856
2	0.898325358852
3	0.898203592814
4	0.897005988024
5	0.891017964072

Isso da uma media de 0.895618714724 para o LDA, ele é o terceiro melhor classificador. A diferencia da acurácia com os outros classificadores é muito pequena.

O código:

```
def LDA_evaluation(x, y):
  skFolds = model_selection.StratifiedKFold(n_splits = 5, random_state = 1)
  skFolds.get_n_splits(x, y)

  accuracy_mean = 0

  for train_index, test_index in skFolds.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]

    #create and train LDA
    clf = LinearDiscriminantAnalysis()
    clf.fit(x_train, y_train)

    #test the trained model
    pred = clf.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, pred)

    print("LDA KF Accuracy", accuracy)

    #acumulate accuracy
    accuracy_mean += accuracy

  print("LDA mean accuracy", accuracy_mean/5.0)
```

Questão 5. *Classificador final*

- *Qual o melhor classificador para esse problema?*
- *Se não o LDA, calcule o hiperparametro C a ser usado.*
- *gere o classificador final.*

As acurácias promedio obtidas foram:

Algorithm	Mean Accuracy
Log. Reg.	0.897295647939
SVM	0.89777383033
LDA	0.895618714724

Há uma pequena diferença entre as accurácias dos classificadores, o SVM é melhor por 0.0005 do que o logistic regression e por 0.002 do que o LDA.

Treinando o linear SVM o melhor hiperparametro é $C = 1$ e a melhor acurácia é 0.90016654152.

O código:

```
def train_best_classifier(x, y):
  C_space = [10**i for i in range(-1, 4) ]

  #create stratified k-folds
  skFolds = model_selection.StratifiedKFold(n_splits = 3, random_state = 1)
  skFolds.get_n_splits(x, y)

  best_c = 0
  best_accuracy = 0

  for c in C_space:
    accuracy_mean = 0

    #test each parameter C with the 3-folds
    for train_index, test_index in skFolds.split(x, y):
      x_train, x_test = x[train_index], x[test_index]
      y_train, y_test = y[train_index], y[test_index]

      svc = svm.LinearSVC(random_state = 1, C=c).fit(x_train, y_train)
      pred = svc.predict(x_test)

      accuracy = metrics.accuracy_score(y_test, pred)
      accuracy_mean += accuracy

    accuracy_mean /= 3.0

    print("\t nested", accuracy_mean, c)

    #update best accuracy and best C
    if best_accuracy <= accuracy_mean:
      best_accuracy = accuracy_mean
      best_c = c

  print("best", best_accuracy, best_c)

  #create final classifier
  return svm.LinearSVC(C=best_c, random_state = 1).fit(x, y)
```