MO444 – Aprendizado de Máquina

Edgar Rodolfo Quispe Condori - RA 192617

April 3, 2017

Questão 1. *Pre-processamento:*

- *Converta a primeira coluna de dados categóricos para numéricos usando o one-hot encoding. A função get_dumies parece ser mais conveniente que o one-hot-encoded do scikit learn.*
- *A última coluna será transformada em um dado categórico. A classe de saída será 1 se a ultima coluna for maior que 13 e 0 se menor ou igual a 13.*
- *Uma vez que vc criou o atributo de saída remova a ultima coluna.*
- *Separe os primeiros 3133 dados como treino e os restantes como teste.*

O código:

```
import numpy as np
import csv
import pandas as pd

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)

  #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]

  #separe data for training and testing
  train_data = data[0:3133 , :]
  train_labels = labels[0:3133]

  test_data = data[3133::, :]
```

```
    test_labels = labels[3133::]

    return train_data, train_labels, test_data, test_labels

if __name__ == "__main__":

    train_data, train_labels, test_data, test_labels = preprocess_data()

    print(train_data)
    print(train_labels)
    print(test_data)
    print(test_labels)
```

Questão 2. *Regressão logística com C=1000000. Faca a regressão logística dos dados transformados, com um C alto (sem regularização). Imprima a acurácia do classificador nos dados de teste com 3 dígitos significativos. Rode o LogisticrRgression com random_state=1 para garantir que de o mesmo resultado toda vez que vc rodar (isso seta o valor da semente do gerador aleatório e portanto usará sempre o mesmo ponto inicial na otimização da regressão logística).*

O resultado obtido é:
Acurácia = 0.897
O código:

```
import numpy as np
from t02_1 import preprocess_data #import code from problem 1
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

if __name__ == "__main__":
    train_data, train_labels, test_data, test_labels = preprocess_data()

    #create and train the logistic Regressor
    clf = LogisticRegression(C=1000000, random_state=1)
    clf.fit(train_data, train_labels)

    #test the trained model
    pred = clf.predict(test_data)

    #show prediction statistics
    print (metrics.classification_report(test_labels, pred))
    print ("Accuracy: {0:.3f}".format(metrics.accuracy_score(test_labels, pred)))
```

Questão 3. *Regressão logística com regularização (C=1). Imprima com 3 dígitos a acurácia, e use random_state=1.*

O resultado obtido é:
Acurácia = 0.892
Comparando esse resultado com o anterior, a regularização piora os resultados em 0.005
Só é necessario trocar a seguinte linha no código da pergunta 2:

```
clf = LogisticRegression(C=1, random_state=1)
```

Questão 4. *Regressão logística sem regularização e com estandardização dos dados. Use C=1000000 mas transforme os dados antes de aplicar a regressão logistica.*

O resultado obtido é:
Acurácia = 0.897

No caso da estandardização, não é marcante na acurácia. Ele tem o mesmo resultado que a regressão logística sem regularação.

O código:

```
import numpy as np
from t02_1 import preprocess_data #import code from problem 1
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn import preprocessing

if __name__ == "__main__":
  #preprocess data
  train_data, train_labels, test_data, test_labels = preprocess_data()

  #fit standardizer with train data
  scaler = preprocessing.StandardScaler().fit(train_data)

  #standardize train/test data
  train_data = scaler.transform(train_data)
  test_data = scaler.transform(test_data)

  #create and train the logistic Regressor
  clf = LogisticRegression(C=1000000, random_state=1)
  clf.fit(train_data, train_labels)

  #test the trained model
  pred = clf.predict(test_data)

  #show prediction statistics
  print (metrics.classification_report(test_labels, pred))
  print ("Accuracy: {0:.3f}".format(metrics.accuracy_score(test_labels, pred)))
```

Questão 5. *Aplique um PCA nos dados, de forma que pelo menos 90% da variância dos dados seja preservada.*

O código:

```
import numpy as np
from t02_1 import preprocess_data #import code from problem 1
from sklearn.decomposition import PCA

def apply_pca():
  #preprocess data
  train_data, train_labels, test_data, test_labels = preprocess_data()

  #fit PCA with train data
  model = PCA(n_components = 0.9).fit(train_data)

  #apply PCA to train/test data
  train_data = model.transform(train_data)
  test_data = model.transform(test_data)

  return train_data, train_labels, test_data, test_labels

if __name__ == "__main__":
  train_data, train_labels, test_data, test_labels = apply_pca()

  #show data
  print(train_data)
  print(train_labels)
```

```
print(test_data)
print(test_labels)
```

Questão 6. *Rode a regressão logística sem regularização nos dados do PCA*

O resultado obtido é:
Acurácia = 0.884
O uso do PCA piora os resultados em 0.013 quando é comparada com a regressão logística sem regularização e sem PCA. Este resultado é interessante porque com só 3 dimenções (pelo menos 90% da variância) a acurácia não diminuiu muito.
O código:

```
import numpy as np
from t02_5 import apply_pca #import code from problem 5
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

if __name__ == "__main__":
  #recover, preprocess and apply pca
  train_data, train_labels, test_data, test_labels = apply_pca()

  #create and train the logistic Regressor
  clf = LogisticRegression(C=1000000, random_state=1)
  clf.fit(train_data, train_labels)

  #test the trained model
  pred = clf.predict(test_data)

  #show prediction statistics
  print (metrics.classification_report(pred, test_labels))
  print ("Accuracy: {0:.3f}".format(metrics.accuracy_score(pred, test_labels)))
```

Questão 7. *Rode a regressão logística com regularização (C=1) nos dados do PCA*

O resultado obtido é:
Acurácia = 0.884
O uso do PCA piora os resultados em 0.008 quando é comparada com a regressão logística com regularização e sem PCA. Além disso, o uso do PCA remove os efeitos da regularização.
Só é necessario trocar a seguinte linha no código da pergunta 6:

```
clf = LogisticRegression(C=1, random_state=1)
```

Questão 8. *Leia o arquivo abalone-missing.csv com dados faltantes na 2 a penúltima coluna. Faça o preprocessamento descrito em 1. e impute pela média os valores faltantes. Rode a regressão sem regularização, sem PCA e sem estandardização*

O resultado obtido é:
Acurácia = 0.887
O uso da imputação piora os resultados em só 0.01 quando é comparada com o melhor resultado obtido (regressão logística sem regularização). Este resultado é importante porque a falta de alguns dados não tem efeitos muito importantes na acurácia.
O código:

```
import numpy as np
import csv
import pandas as pd
```

```
from sklearn.preprocessing import Imputer
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

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

  #convert data to numpy array
  data = np.array(data)

  #replace missing data with np.nan
  data[data == 'NA'] = np.nan

  #get first column and update data columns
  first_column = data[:, 0]
  data = data[:, 1::]

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

  #separe and transform last column
  labels = data[:, -1].astype(np.float)
  labels[labels <= 13] = 0
  labels[labels > 13] = 1
  data = data[:, 0:-1]

  #separe data for training and testing
  train_data = data[0:3133 , :]
  train_labels = labels[0:3133]

  test_data = data[3133::, :]
  test_labels = labels[3133::]

  #fit imputer with train data
  imp = Imputer(missing_values = "NaN", strategy = "mean").fit(train_data)

  #impute missing train/test data
  train_data = imp.transform(train_data)
  test_data = imp.transform(test_data)

  return train_data, train_labels, test_data, test_labels

if __name__ == "__main__":

  train_data, train_labels, test_data, test_labels = preprocess_data_imputation()

  #create and train the logistic Regressor
  clf = LogisticRegression(C=1000000, random_state=1)
  clf.fit(train_data, train_labels)

  #test the trainet model
  pred = clf.predict(test_data)
```

```
#show prediction statistics
print (metrics.classification_report(test_labels, pred))
print ("Accuracy: {0:.3f}".format(metrics.accuracy_score(test_labels, pred)))
```