Regressao_Logistica_V1

September 9, 2020

1 Logistic Regression and Classification Error Metrics

2 Gustavo Gimpel Correia Lima

3 Matrícula: 201512040488

3.1 Introduction

We will be using the Human Activity Recognition with Smartphones database, which was built from the recordings of study participants performing activities of daily living (ADL) while carrying a smartphone with an embedded inertial sensors. The objective is to classify activities into one of the six activities (walking, walking upstairs, walking downstairs, sitting, standing, and laying) performed.

For each record in the dataset it is provided:

- Triaxial acceleration from the accelerometer (total acceleration) and the estimated body acceleration.
- Triaxial Angular velocity from the gyroscope.
- A 561-feature vector with time and frequency domain variables.
- Its activity label.

More information about the features is available on the website: above or at https://www.kaggle.com/uciml/human-activity-recognition-with-smartphones

[1]:
```
from __future__ import print_function
import os
#Data Path has to be set as per the file location in your system
#data_path = ['..', 'data']
data_path = ['']
```

3.2 Question 1

Import the data and do the following:

- Examine the data types–there are many columns, so it might be wise to use value counts
- Determine if the floating point values need to be scaled
- Determine the breakdown of each activity
- Encode the activity label as an integer

4 Os dados são lidos e analisados. Como a coluna de atividades é a única não numérica, as mesmas são codificadas para números inteiros de 0 a 5 utilizando o LabelEncoder, através da função fit_transform

[2]:
```
import pandas as pd
import numpy as np
#The filepath is dependent on the data_path set in the previous cell
filepath = os.sep.join(['Human_Activity_Recognition_Using_Smartphones_Data.
 ↪csv'])
data = pd.read_csv(filepath, sep=',')
```

The data columns are all floats except for the activity label.

[3]:
```
data.dtypes.value_counts()
```

[3]:
```
float64    561
object       1
dtype: int64
```

[4]:
```
data.dtypes.tail()
```

[4]:
```
angle(tBodyGyroJerkMean,gravityMean)    float64
angle(X,gravityMean)                    float64
angle(Y,gravityMean)                    float64
angle(Z,gravityMean)                    float64
Activity                                 object
dtype: object
```

The data are all scaled from -1 (minimum) to 1.0 (maximum).

[5]:
```
data.iloc[:, :-1].min().value_counts()
```

[5]:
```
-1.0    561
dtype: int64
```

[6]:
```
data.iloc[:, :-1].max().value_counts()
```

[6]:
```
1.0    561
dtype: int64
```

Examine the breakdown of activities–they are relatively balanced.

[7]:
```
data.Activity.value_counts()
```

[7]:
```
LAYING                1944
STANDING              1906
```

```
SITTING               1777
WALKING               1722
WALKING_UPSTAIRS      1544
WALKING_DOWNSTAIRS    1406
Name: Activity, dtype: int64
```

Scikit learn classifiers won't accept a sparse matrix for the prediction column. Thus, either `LabelEncoder` needs to be used to convert the activity labels to integers, or if `DictVectorizer` is used, the resulting matrix must be converted to a non-sparse array.
Use `LabelEncoder` to fit_transform the "Activity" column, and look at 5 random values.

[8]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
data['Activity'] = le.fit_transform(data.Activity)
data['Activity'].sample(5)
```

[8]:
```
8788    0
6719    0
8092    5
304     5
7882    0
Name: Activity, dtype: int64
```

4.1 Question 2

- Calculate the correlations between the dependent variables.
- Create a histogram of the correlation values
- Identify those that are most correlated (either positively or negatively).

5 É calculada a correlação entre as características

[9]:
```
# Calculate the correlation values
feature_cols = data.columns[:-1]
corr_values = data[feature_cols].corr()

# Simplify by emptying all the data below the diagonal
tril_index = np.tril_indices_from(corr_values)

# Make the unused values NaNs
for coord in zip(*tril_index):
    corr_values.iloc[coord[0], coord[1]] = np.NaN

# Stack the data and convert to a data frame
corr_values = (corr_values.stack().to_frame().reset_index().
 ↪rename(columns={'level_0':'feature1','level_1':'feature2',0:'correlation'}))
```

```python
# Get the absolute values for sorting
corr_values['abs_correlation'] = corr_values.correlation.abs()
```

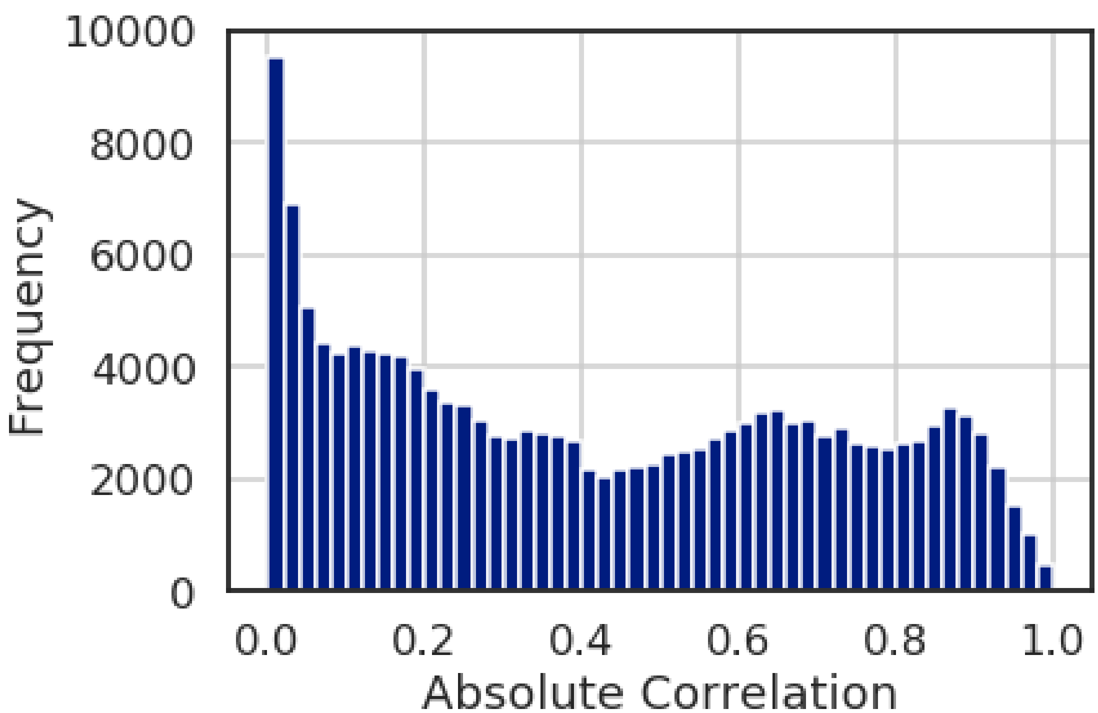
A histogram of the absolute value correlations.

[10]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[11]:
```python
sns.set_context('talk')
sns.set_style('white')
sns.set_palette('dark')

ax = corr_values.abs_correlation.hist(bins=50)

ax.set(xlabel='Absolute Correlation', ylabel='Frequency');
```



6 Verificar as features que estão mais correlacionadas ajuda a identificar a necessidade de eventuais ajustes no dataset, como lidar com características similares ou redundantes.

[12]:
```
# The most highly correlated values
corr_values.sort_values('correlation', ascending=False).
↪query('abs_correlation>0.8')
```

[12]:
```
                               feature1                    feature2  correlation  \
156894  fBodyBodyGyroJerkMag-mean()  fBodyBodyGyroJerkMag-sma()     1.000000
93902              tBodyAccMag-sma()         tGravityAccMag-sma()     1.000000
101139        tBodyAccJerkMag-mean()        tBodyAccJerkMag-sma()     1.000000
96706          tGravityAccMag-mean()         tGravityAccMag-sma()     1.000000
94257           tBodyAccMag-energy()      tGravityAccMag-energy()     1.000000
…                                  …                           …          …
22657           tGravityAcc-mean()-Y         angle(Y,gravityMean)    -0.993425
39225      tGravityAcc-arCoeff()-Z,3   tGravityAcc-arCoeff()-Z,4    -0.994267
38739      tGravityAcc-arCoeff()-Z,2   tGravityAcc-arCoeff()-Z,3    -0.994628
23176           tGravityAcc-mean()-Z         angle(Z,gravityMean)    -0.994764
38252      tGravityAcc-arCoeff()-Z,1   tGravityAcc-arCoeff()-Z,2    -0.995195

        abs_correlation
156894         1.000000
93902          1.000000
101139         1.000000
96706          1.000000
94257          1.000000
…                  …
22657          0.993425
39225          0.994267
38739          0.994628
23176          0.994764
38252          0.995195

[22815 rows x 4 columns]
```

6.1 Question 3

- Split the data into train and test data sets. This can be done using any method, but consider using Scikit-learn's `StratifiedShuffleSplit` to maintain the same ratio of predictor classes.
- Regardless of methods used to split the data, compare the ratio of classes in both the train and test splits.

7 O StratifiedShuffleSplit é utilizado como alternativa ao KFold.

7.1 1) n_splits=1: consiste da quantidade de vezes que o dataset será embaralhado e particionado, para somente depois definir os grupos de teste e trieno.

7.2 2) test_size=0.3: proporção do número de amostras que serão utilizadas no treinamento. Empiricamente, algumas proporções de amostras de treino e de teste são ideais para a obtenção de bons modelos (Ex.: 70%-30%, 80%-20%, 75%-25%, respectivamente treino-teste)

7.3 3) random_state=42: um valor para iniciar o algoritmo de pseudo-aleatoriedade.

7.4 Os conjuntos de treino e teste são construídos preservando a porcentagem de amostras de cada classe ao particionar o dataset.

[13]:
```python
from sklearn.model_selection import StratifiedShuffleSplit

# Get the split indexes
strat_shuf_split = StratifiedShuffleSplit(n_splits=1,test_size=0.3,␣
 ↪random_state=42)

train_idx, test_idx = next(strat_shuf_split.split(data[feature_cols], data.
 ↪Activity))

# Create the dataframes
X_train = data.loc[train_idx, feature_cols]
y_train = data.loc[train_idx, 'Activity']

X_test  = data.loc[test_idx, feature_cols]
y_test  = data.loc[test_idx, 'Activity']
```

[14]:
```python
y_train.value_counts(normalize=True)
```

[14]:
```
0    0.188792
2    0.185046
1    0.172562
3    0.167152
5    0.149951
4    0.136496
Name: Activity, dtype: float64
```

[15]:
```python
y_test.value_counts(normalize=True)
```

[15]:
```
0    0.188673
2    0.185113
1    0.172492
3    0.167314
```

```
5    0.149838
4    0.136570
Name: Activity, dtype: float64
```

7.5 Question 4

- Fit a logistic regression model without any regularization using all of the features. Be sure to read the documentation about fitt ing a multi-class model so you understand the coefficient output. Store the model.

8 Para conseguir realizar a regressão linear sem a 'regularization', foi necessário adicionar o parâmetro penalty='none', visto que, de acordo com a documentação, o default deste parâmetro é a norma l2.

9 O solver utilizado por padrão é o 'lbfgs'

10 O parâmetro multi_class não foi especificado, e por padrão, caso a biblioteca detecte que há mais de 2 classes nas saídas, ele utiliza multiclasses.

11 Como as classes são codificadas entre 0 e 5, o algoritmo utiliza a estratégia 'um-vs-todos': uma classe é tida como correta e as demais como incorretas.

[16]:
```python
from sklearn.linear_model import LogisticRegression

# Standard logistic regressio
lr = LogisticRegression(max_iter=1000,n_jobs=16, penalty='none').fit(X_train,␣
 ↪y_train)
```

11.1 Question 5

Calculate the following error metric:

- accuracy

12 Com o modelo já treinado, foi feita a previsão das classes utilizando as amostras de treinamento. O resultado está em Integer devido a conversão das classes em números inteiros.

[17]:
```
predict=lr.predict(X_test)
print (predict)
```

```
[3 5 3 … 1 1 5]
```

13 Utilizando as ferramentas de metricas do sklearn, foi possível avaliar a acurácia do modelo.

13.1 Com max_iter=1000 a acurárica foi de 0.9825242718446602

13.2 Com max_iter=100 a acurácia foi de 0.9766990291262136

[18]:
```
from sklearn import metrics
print (metrics.accuracy_score(y_test, predict))
```

```
0.9825242718446602
```