Aluno: Vinícius França Lima Vilaça

Regressão Logística

1. A questão 1 busca validar os dados do database utilizado. Primeiramente é necessário validar os tipos de dados, precisando que estes sejam númericos para que seja possível efetuar os cálculos corretamente. Em segundo momento se estuda a escala dos valores, onde é preciso realizar a uniformização dos dados, ou seja, colocar diferentes variáveis na mesma escala, é interessante que se faça quando estamos lidando com algoritmos baseados no Gradiente Descendente, pois estes algoritmos são sensíveis ao range dos valores dos dados e dessa forma a acurácia do algoritmo irá ser beneficiada pois o limiar de decisão da função logística ficará bem definido. Avalise também a distribuição das classes, assim pode se verificar se o dataset está desbalanceado, ou seja, se o algoritmo pode ser tendencioso ou não. Por fim, caso os dados da classicação não estejam em dados númericos é necessario que se codifique os valores para que seja possível utilizar o classificador.
2. A questão 2 realiza o **feature selection**, que significa buscas as features que possuem os valóres mais precisos e preciosos para o modelo. Dito isto, a correlação das features é estudada, a correlação pode ajudar a predizer um atributo de outro, ou identificar atributos que não interferem no modelo pois não possuem relação com nenhum outro. Correlação pode identificar um problema conhecido como **Multicollinearity** quando uma variável pode ser linearmente predita por outra com um alto grau de precisão, isso pode levar a resultados distorcidos ou enganosos e por isso e por outros problemas é um bom método de avaliação.
3. A questão 3 faz a separação dos dados de treino e teste, essa etapa é importante para evitar um modelo com underfitting ou overfitting. Foi utilizado o método *Cross Validator*, a validação cruzada é uma fusão de StratifiedKFold e ShuffleSplit, que retorna **folds** estratificados aleatórios em que os fold são feitos preservando o percentual de amostras de cada classe.
4. O modelo foi implementado ao final do notbook...
5. A acurácia foi calculada ao final do notbook...

Logistic Regression and Classification Error Metrics

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

In [48]:
```
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 = ['.']
```

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

In [50]:
```
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(data_path + ['data.csv'])
data = pd.read_csv(filepath, sep=',')
data
```

Out[50]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyB
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	
...	...	...	...	...	...	...	...	...	...	...	...	
10294	0.310155	-0.053391	-0.099109	-0.287866	-0.140589	-0.215088	-0.356083	-0.148775	-0.232057	0.185361	...	
10295	0.363385	-0.039214	-0.105915	-0.305388	0.028148	-0.196373	-0.373540	-0.030036	-0.270237	0.185361	...	
10296	0.349966	0.030077	-0.115788	-0.329638	-0.042143	-0.250181	-0.388017	-0.133257	-0.347029	0.007471	...	
10297	0.237594	0.018467	-0.096499	-0.323114	-0.229775	-0.207574	-0.392380	-0.279610	-0.289477	0.007471	...	
10298	0.153627	-0.018437	-0.137018	-0.330046	-0.195253	-0.164339	-0.430974	-0.218295	-0.229933	-0.111527	...	

10299 rows × 562 columns

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

In [52]:
```
data.dtypes.value_counts()
```

Out[52]:
```
float64    561
object       1
dtype: int64
```

In [54]:
```
data.dtypes.tail(10)
```

Out[54]:
```
fBodyBodyGyroJerkMag-skewness()         float64
fBodyBodyGyroJerkMag-kurtosis()         float64
angle(tBodyAccMean,gravity)             float64
angle(tBodyAccJerkMean),gravityMean)    float64
angle(tBodyGyroMean,gravityMean)        float64
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).

In [56]:
```
data.iloc[:, :-1].min().value_counts()
```

Out[56]:
```
-1.0    561
dtype: int64
```

In [58]:
```
data.iloc[:, :-1].max().value_counts()
```

Out[58]:
```
1.0    561
dtype: int64
```

Examine the breakdown of activities--they are relatively balanced.

In [60]:
```
data.Activity.value_counts()
```

Out[60]:
```
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.

In [62]:
```
from sklearn.preprocessing import LabelEncoder

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

Out[62]:
```
7972    2
497     5
886     1
3026    5
4532    5
Name: Activity, dtype: int64
```

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).

In [64]:
```
# Calculate the correlation values
feature_cols = data.columns[:-1]
corr_values = data[feature_cols].corr()
corr_values
```

Out[64]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y
tBodyAcc-mean()-X	1.000000	0.128037	-0.230302	0.004590	-0.016785	-0.036071	0.010303	-0.017488
tBodyAcc-mean()-Y	0.128037	1.000000	-0.029882	-0.046352	-0.046996	-0.054153	-0.045247	-0.047673
tBodyAcc-mean()-Z	-0.230302	-0.029882	1.000000	-0.024185	-0.023745	-0.015632	-0.022872	-0.022966
tBodyAcc-std()-X	0.004590	-0.046352	-0.024185	1.000000	0.922525	0.861910	0.998662	0.916087
tBodyAcc-std()-Y	-0.016785	-0.046996	-0.023745	0.922525	1.000000	0.888259	0.918561	0.997510
...	...	...	...	...	...	...	...	...
angle(tBodyGyroMean,gravityMean)	0.036047	0.013241	-0.066233	0.027464	0.001902	-0.004984	0.027729	-0.002924
angle(tBodyGyroJerkMean,gravityMean)	0.034296	0.077627	-0.030748	-0.027123	-0.015784	-0.012196	-0.027097	-0.013411
angle(X,gravityMean)	-0.041021	-0.007513	0.003215	-0.374104	-0.381391	-0.353271	-0.371168	-0.378013
angle(Y,gravityMean)	0.034053	-0.005616	-0.012986	0.449425	0.506106	0.459092	0.444926	0.507947
angle(Z,gravityMean)	0.030656	-0.016233	-0.028406	0.393063	0.425511	0.483424	0.389481	0.424479

561 rows × 561 columns

In [66]:
```
# Simplify by emptying all the data below the diagonal
tril_index = np.tril_indices_from(corr_values)
tril_index
```

Out[66]:
```
(array([  0,   1,   1, ..., 560, 560, 560]),
 array([  0,   0,   1, ..., 558, 559, 560]))
```

In [68]:
```
# Make the unused values NaNs
for coord in zip(*tril_index):
    corr_values.iloc[coord[0], coord[1]] = np.NaN
corr_values
```

Out[68]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y
tBodyAcc-mean()-X	NaN	0.128037	-0.230302	0.004590	-0.016785	-0.036071	0.010303	-0.017488
tBodyAcc-mean()-Y	NaN	NaN	-0.029882	-0.046352	-0.046996	-0.054153	-0.045247	-0.047673
tBodyAcc-mean()-Z	NaN	NaN	NaN	-0.024185	-0.023745	-0.015632	-0.022872	-0.022966
tBodyAcc-std()-X	NaN	NaN	NaN	NaN	0.922525	0.861910	0.998662	0.916087
tBodyAcc-std()-Y	NaN	NaN	NaN	NaN	NaN	0.888259	0.918561	0.997510
...	...	...	...	...	...	...	...	...
angle(tBodyGyroMean,gravityMean)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
angle(tBodyGyroJerkMean,gravityMean)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
angle(X,gravityMean)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
angle(Y,gravityMean)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
angle(Z,gravityMean)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

561 rows × 561 columns

In [70]:
```
# 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'}))
corr_values
```

Out[70]:

	feature1	feature2	correlation
0	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	0.128037
1	tBodyAcc-mean()-X	tBodyAcc-mean()-Z	-0.230302
2	tBodyAcc-mean()-X	tBodyAcc-std()-X	0.004590
3	tBodyAcc-mean()-X	tBodyAcc-std()-Y	-0.016785
4	tBodyAcc-mean()-X	tBodyAcc-std()-Z	-0.036071
...	...	...	...
157075	angle(tBodyGyroJerkMean,gravityMean)	angle(Y,gravityMean)	-0.004582
157076	angle(tBodyGyroJerkMean,gravityMean)	angle(Z,gravityMean)	-0.012549
157077	angle(X,gravityMean)	angle(Y,gravityMean)	-0.748249
157078	angle(X,gravityMean)	angle(Z,gravityMean)	-0.635231
157079	angle(Y,gravityMean)	angle(Z,gravityMean)	0.545614

157080 rows × 3 columns

In [71]:
```
# Get the absolute values for sorting
corr_values['abs_correlation'] = corr_values.correlation.abs()
corr_values
```

Out[71]:

	feature1	feature2	correlation	abs_correlation
0	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	0.128037	0.128037
1	tBodyAcc-mean()-X	tBodyAcc-mean()-Z	-0.230302	0.230302
2	tBodyAcc-mean()-X	tBodyAcc-std()-X	0.004590	0.004590
3	tBodyAcc-mean()-X	tBodyAcc-std()-Y	-0.016785	0.016785
4	tBodyAcc-mean()-X	tBodyAcc-std()-Z	-0.036071	0.036071
...	...	...	...	...
157075	angle(tBodyGyroJerkMean,gravityMean)	angle(Y,gravityMean)	-0.004582	0.004582
157076	angle(tBodyGyroJerkMean,gravityMean)	angle(Z,gravityMean)	-0.012549	0.012549
157077	angle(X,gravityMean)	angle(Y,gravityMean)	-0.748249	0.748249
157078	angle(X,gravityMean)	angle(Z,gravityMean)	-0.635231	0.635231
157079	angle(Y,gravityMean)	angle(Z,gravityMean)	0.545614	0.545614

157080 rows × 4 columns

A histogram of the absolute value correlations.

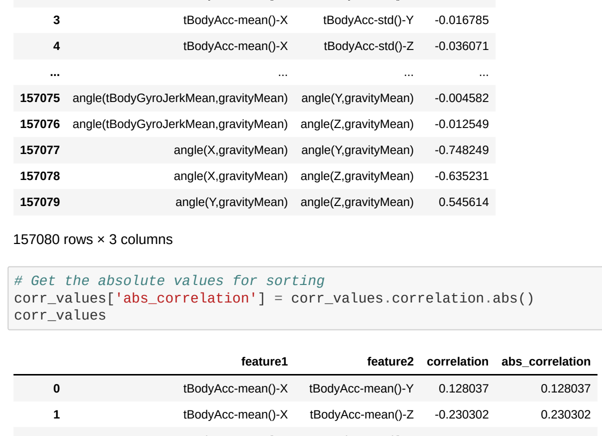
In [72]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [73]:
```
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')
```

Out[73]:
```
[Text(0.5, 0, 'Absolute Correlation'), Text(0, 0.5, 'Frequency')]
```



In [74]:
```
# The most highly correlated values
corr_values.sort_values('correlation', ascending=False).query('abs_correlation>0.8')
```

Out[74]:

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

22815 rows × 4 columns

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.

In [75]:
```
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']
```

In [76]:
```
y_train.value_counts(normalize=True)
```

Out[76]:
```
0    0.188792
2    0.185046
1    0.172562
3    0.167152
5    0.149951
4    0.136496
Name: Activity, dtype: float64
```

In [77]:
```
y_test.value_counts(normalize=True)
```

Out[77]:
```
0    0.188673
2    0.185113
1    0.172492
3    0.167314
5    0.149838
4    0.136570
Name: Activity, dtype: float64
```

Question 4

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

In [78]:
```
from sklearn.linear_model import LogisticRegression

# Standard logistic regressio
model_lr = LogisticRegression().fit(X_train, y_train)
```

In [79]:
```
predicted_activity = model_lr.predict(X_test)
```

Question 5

Calculate the following error metric:

- accuracy

In [80]:
```
accuracy = sum(predicted_activity==y_test) / len(y_test)
accuracy
```

Out[80]:
```
0.9805825242718447
```

In [ ]: