Machine_learning_final_project

February 27, 2020

[1]:
```
from functools import reduce
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # standard graphics
import seaborn as sns # fancier graphics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
```

[ ]:
```
#Setting up Spark environment
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://www-us.apache.org/dist/spark/spark-2.4.5/spark-2.4.
 ↪5-bin-hadoop2.7.tgz
!tar xf spark-2.4.5-bin-hadoop2.7.tgz
!pip install -q findspark
```

[4]:
```
import os
os.environ["SPARK_HOME"] = "/content/spark-2.4.5-bin-hadoop2.7"
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
```

[ ]:
```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

[ ]:
```
from google.colab import files
uploaded = files.upload()
```

[2]:
```
df = pd.read_csv('carwood.csv')
```

[3]:
```
df.head()
```

[3]:

	f1	f2	f3	f4	f5	f6	f7	f8	f9	\
0	170.39	167.28	143.44	124.67	139.01	125.83	144.33	151.26	175.51	
1	169.75	190.96	175.53	138.27	137.47	139.23	133.23	130.25	147.73	
2	153.69	153.68	144.02	158.73	178.87	157.04	152.92	147.52	142.87	
3	131.69	151.56	151.05	134.00	151.18	175.53	171.34	159.77	151.95	
4	162.85	158.88	132.27	138.41	143.98	159.30	177.26	180.58	159.34	

	f10	...	f59	f60	f61	f62	f63	f64	f65	\

```
0  171.31  ...  169.67  157.51  161.06  133.23  124.41  138.44  142.93
1  163.93  ...  141.58  153.39  141.00  148.43  168.12  169.90  165.64
2  165.26  ...  170.51  155.37  167.11  146.89  141.01  159.43  169.68
3  146.10  ...  155.82  157.83  152.43  150.82  146.58  128.85  140.76
4  164.66  ...  130.96  135.74  167.31  188.21  179.52  146.20  153.73

      f66     f67  label
0  137.13  134.44      0
1  166.86  137.69      0
2  163.24  165.17      0
3  177.35  174.61      0
4  152.12  146.58      0

[5 rows x 68 columns]
```

[5]:
```
#Checking dimension(features, observations)
#The number of features is less than observations.
#Therefore, I consider it as a low dimensional dataset.
df.shape
```

[5]:
```
(2048, 68)
```

[6]:
```
#dataset is complete
missing_data = df.isnull().sum()
missing_data.head()
```

[6]:
```
f1    0
f2    0
f3    0
f4    0
f5    0
dtype: int64
```

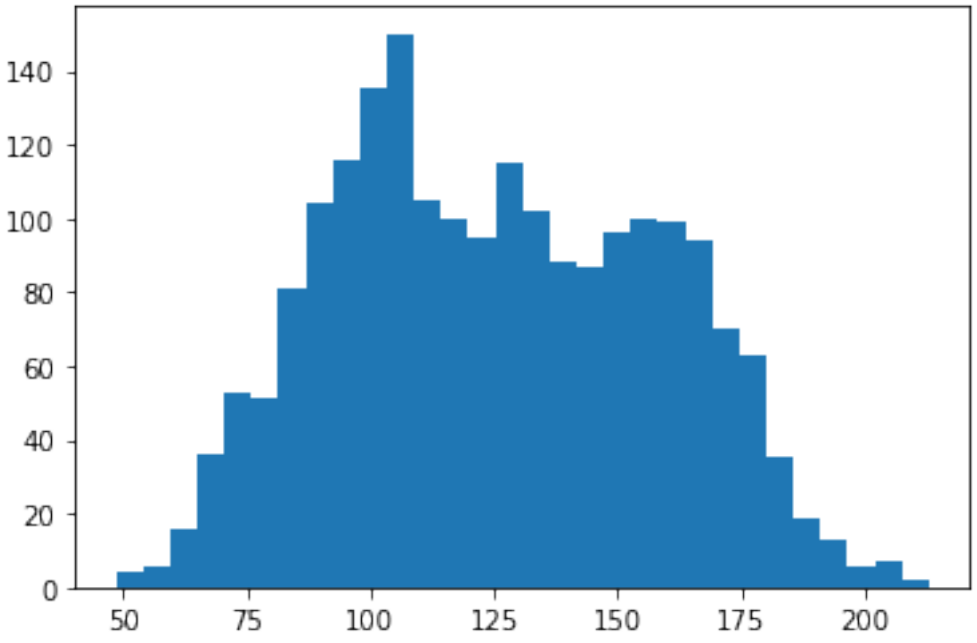
[7]:
```
#According to the properties (mean, meadian, min, max, etc.) it should be␣
 ↪claimed statistical results and range are similar
df.describe()
```

[7]:
```
                f1           f2           f3           f4           f5  \
count  2048.000000  2048.000000  2048.000000  2048.000000  2048.000000
mean    125.569231   125.356807   125.422041   125.719333   126.005055
std      33.292731    32.822212    32.643005    32.966031    33.526247
min      47.124000    47.262000    48.485000    49.323000    47.077000
25%      99.490000    99.095500   100.217500    99.784750    99.094250
50%     123.430000   124.160000   123.970000   124.460000   123.735000
75%     153.017500   151.252500   152.505000   152.347500   152.677500
max     210.650000   210.200000   212.930000   211.000000   213.100000

                f6           f7           f8           f9          f10  ...  \
count  2048.000000  2048.000000  2048.000000  2048.000000  2048.000000  ...
mean    126.084465   125.885123   125.783833   125.704766   125.858361  ...
```

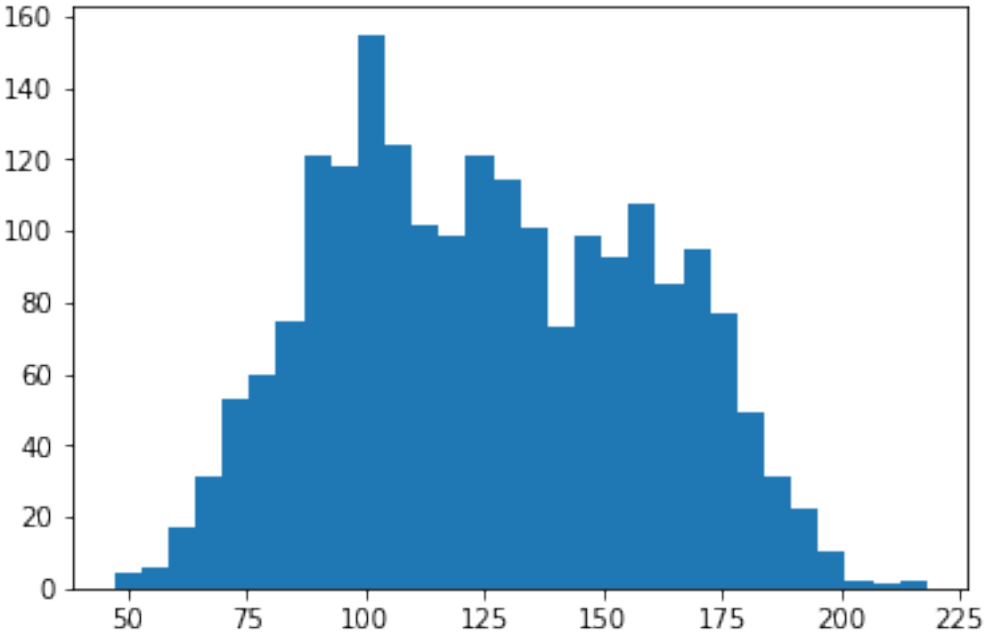
```
std      33.589233   33.220224   33.214697   33.548514   33.402901  ...
min      47.365000   47.063000   47.546000   49.302000   48.393000  ...
25%      98.990500   99.144750   99.745750   98.876750   99.449500  ...
50%     124.275000  124.465000  124.390000  123.425000  124.595000  ...
75%     153.165000  153.202500  152.085000  152.965000  152.192500  ...
max     215.900000  218.090000  215.430000  223.880000  224.050000  ...

                f59         f60         f61         f62         f63  \
count  2048.000000  2048.000000  2048.000000  2048.000000  2048.000000
mean    125.201682   124.988847   124.636116   124.841744   125.067299
std      33.795493    33.527471    33.203148    33.230530    33.728213
min      47.957000    49.456000    47.444000    45.266000    44.772000
25%     100.782500   100.001750    99.410750    99.743250    99.723250
50%     121.310000   122.830000   122.720000   121.955000   122.195000
75%     152.540000   151.660000   150.965000   152.472500   152.145000
max     219.110000   221.070000   205.700000   213.330000   216.480000

                f64         f65         f66         f67        label
count  2048.000000  2048.000000  2048.000000  2048.000000  2048.000000
mean    125.202800   125.444424   125.414388   125.564611     0.501465
std      33.685539    33.716557    33.675680    33.669027     0.500120
min      46.018000    47.871000    50.691000    53.071000     0.000000
25%      99.964250   100.127500   100.280000   100.590000     0.000000
50%     122.650000   122.845000   123.160000   123.250000     1.000000
75%     152.840000   153.142500   153.210000   152.800000     1.000000
max     217.970000   212.540000   203.640000   209.640000     1.000000

[8 rows x 68 columns]
```

Checking scalability

[8]:
```
plt.hist(df['f3'], bins=30)
plt.show()
```

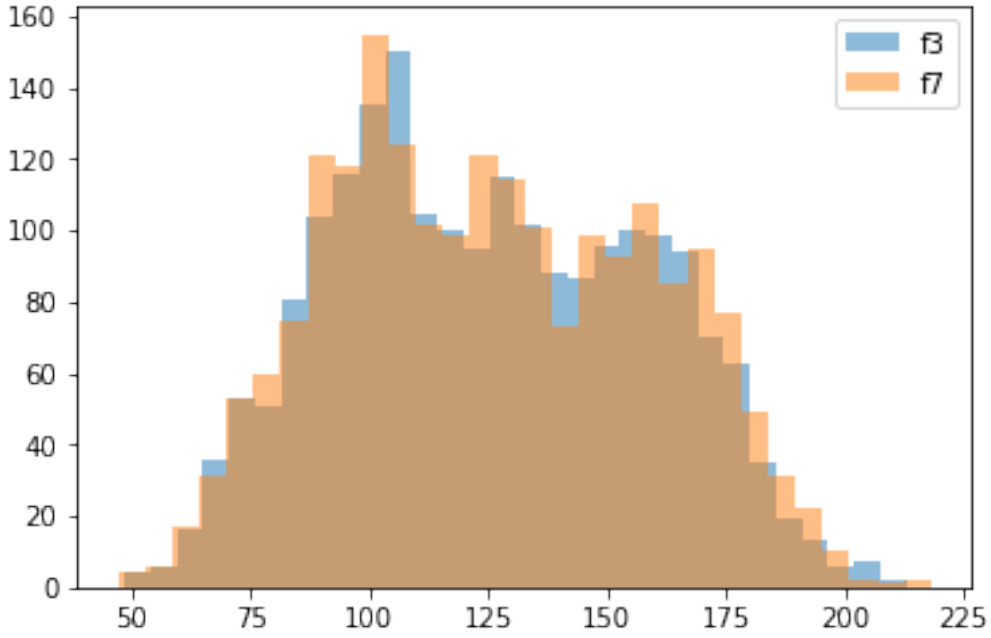


[9]:
```
plt.hist(df['f7'], bins=30)
plt.show()
```



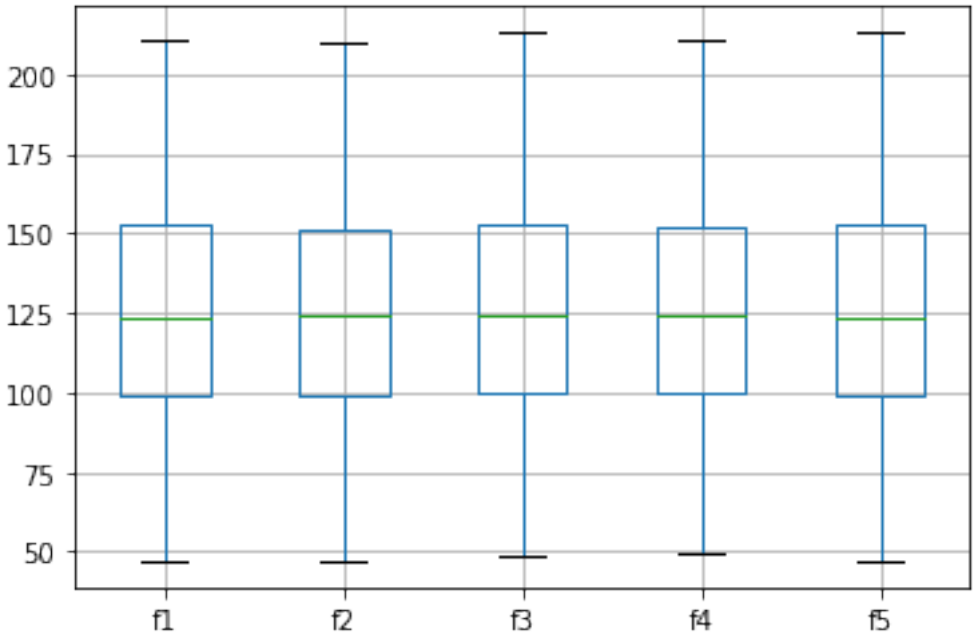
The features such as f3 and f7 have the same units and scales.

[10]:
```
f3 = [df['f3']]
f7 = [df['f7']]
plt.hist(f3, alpha=0.5, label='f3', bins=30)
plt.hist(f7, alpha=0.5, label='f7', bins=30)
plt.legend(loc='upper right')
plt.show()
```



[11]:
```
df.boxplot(column = ['f1', 'f2', 'f3', 'f4', 'f5'])
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f233d5a7b00>
```



This histogram proves the previous claim and enforces the idea that there is no need for extra action such as normalization or standardization.

[12]:
```
#Finding duplicate columns
def DuplicateColumn(df):
    DuplicateColumns= set()
    for x in range(df.shape[1]):
        col = df.iloc[:, x]
        for y in range(x + 1, df.shape[1]):
            others = df.iloc[:, y]
            if col.equals(others):
                DuplicateColumns.add(df.columns.values[y])

    return list(DuplicateColumns)
```

[13]:
```
my_duplicates = DuplicateColumn(df)
```

[14]:
```
print('Please find duplicate columns:')
for col in my_duplicates:
    print('Column name : ', col)
```

```
Please find duplicate columns:
Column name :  f61
Column name :  f57
Column name :  f60
```

[15]:
```
# Deleting duplicate columns
new_df = df.drop(columns=DuplicateColumn(df))
```

```
print("New dataframe", new_df, sep='\n')
```

```
New dataframe
           f1       f2       f3       f4       f5       f6       f7       f8  \
0     170.390  167.280  143.440  124.670  139.010  125.830  144.330  151.260
1     169.750  190.960  175.530  138.270  137.470  139.230  133.230  130.250
2     153.690  153.680  144.020  158.730  178.870  157.040  152.920  147.520
3     131.690  151.560  151.050  134.000  151.180  175.530  171.340  159.770
4     162.850  158.880  132.270  138.410  143.980  159.300  177.260  180.580
...       ...      ...      ...      ...      ...      ...      ...      ...
2043   98.263  100.060   98.223   95.452   98.313   95.614   97.842   99.964
2044   75.507   73.811   73.184   72.460   73.367   72.738   73.257   73.636
2045   97.784  101.720   99.211   99.763   89.432   86.152   88.999   90.974
2046  119.330  118.820  119.030  118.250  117.980  117.340  119.630  115.710
2047  109.510  108.060  106.520  109.350  111.750  109.470  107.080  107.300

           f9      f10  ...      f56      f58      f59      f62      f63  \
0     175.510  171.310  ...  144.650  172.960  169.670  133.230  124.410
1     147.730  163.930  ...  141.020  139.580  141.580  148.430  168.120
2     142.870  165.260  ...  155.180  155.190  170.510  146.890  141.010
3     151.950  146.100  ...  129.510  164.250  155.820  150.820  146.580
4     159.340  164.660  ...  142.470  132.800  130.960  188.210  179.520
...       ...      ...  ...      ...      ...      ...      ...      ...
2043   97.878  101.450  ...   79.926   77.238   82.547   73.833   76.082
2044   74.042   75.384  ...   56.846   55.581   56.945   67.378   65.919
2045  113.630  109.990  ...  109.260  110.030  112.550  117.050  115.920
2046  116.590  118.400  ...  110.710  110.270  109.680  110.630  113.480
2047  112.620  107.640  ...  111.720  108.240  107.170  114.650  116.890

          f64      f65      f66      f67  label
0     138.440  142.930  137.130  134.440      0
1     169.900  165.640  166.860  137.690      0
2     159.430  169.680  163.240  165.170      0
3     128.850  140.760  177.350  174.610      0
4     146.200  153.730  152.120  146.580      0
...       ...      ...      ...      ...    ...
2043   75.586   74.998   76.695   78.166      1
2044   64.443   61.225   58.836   61.005      1
2045  116.440  113.940  113.270  120.430      1
2046  110.780  109.400  108.970  109.590      1
2047  114.780  113.660  107.080  103.280      1

[2048 rows x 65 columns]
```

[16]:
```
#checking duplicates in new df
my_duplicates_new = DuplicateColumn(new_df)
```

[17]:
```
#No duplicates
print('No duplicate columns')
for col in my_duplicates_new:
    print('Column name : ', col)
```

```
No duplicate columns
```

[18]:
```
new_df['label'].value_counts()
```

[18]:
```
1    1027
0    1021
Name: label, dtype: int64
```

[ ]:
```
#Above given results show that it is imbalanced data
```

[19]:
```
# Training model on imbalanced data
# Creating X, y features
y = new_df.label
X = new_df.drop('label', axis=1)

# Train model
lgt1 = LogisticRegression(penalty='l1', solver='liblinear').fit(X, y)

# Predict on training set
prediction_y_1 = lgt1.predict(X)
```

[20]:
```
# Checking accuracy
print(accuracy_score(prediction_y_1, y))
#0.99169921875
```

```
0.9921875
```

[21]:
```
# Creating majority and minority classes
df_major = new_df[new_df.label==1]
df_minor = new_df[new_df.label==0]
```

[22]:
```
# Resampling
df_minority_resample= resample(df_minor,
                               replace=True,
                               n_samples=1027,
                               random_state=123)
```

[23]:
```
#combining majority with minority
df_resample = pd.concat([df_major, df_minority_resample])
```

[24]:
```
#After resampling it is visible that now data is balanced
df_resample.label.value_counts()
```

[24]:
```
1    1027
0    1027
Name: label, dtype: int64
```

Checking accuracy in balanced data

[25]:
```
#creating X, y features
y = df_resample.label
X = df_resample.drop('label', axis=1)
```

[26]:
```
# Logistic regression and fitting
lgt2 = LogisticRegression(penalty='l1', solver='liblinear').fit(X, y)
```

[27]:
```
# Prediction
prediction_y_2 = lgt2.predict(X)
```

[28]:
```
# It predicts both classes
print(np.unique(prediction_y_2))
```

```
[0 1]
```

[29]:
```
# Accuracy checking
print(accuracy_score(y, prediction_y_2))
# 0.997078870496592
```

```
0.997078870496592
```

[30]:
```
from sklearn.metrics import roc_auc_score
```

[31]:
```
#Checking probablity
probability_y_2 = lgt2.predict_proba(X)

#Keeping positive class
probability_y_2 = [p[1] for p in probability_y_2]
```

[32]:
```
#How model does well in terms of AUROC of the dataset
print( roc_auc_score(y, probability_y_2) )
```

```
0.9999668161205391
```

[33]:
```
#AUROC on imbalanced data
imbalance_probab = lgt1.predict_proba(X)
imbalance_probab = [p[1] for p in imbalance_probab]

print(roc_auc_score(y, imbalance_probab))
```

```
0.9997326327426286
```

The dataset is complete but imbalanced. Therefore, it must be balanced. Additionally, boxplot, histograms, the properties such as mean, meadian, min/ max values, etc. prove that no need to do normalization or standardization. Firstly, logistic regression is applied to the imbalanced data. After balancing the data the same process is executed on the balanced data. The rationale behind that is machine learning algorithms that are designed to increase overall accuracy. In this vein, the

comparison should shed light on the results from two regressions. Furthemore, there is a need to check the accuracy results and AUCROC results. From the results, it should be claimed that there is not any problem related to the scalability and the data is balanced, accuracy results are close to each other. In conclusion, above given argumentation once again proves the hypothesis that the dataset is accurate.