Classification vs Regression Algorithm

February 14, 2021

[1]:
```python
#Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```python
df = pd.read_csv(r"C:\Users\admin\Desktop\Data Science\Projects\Case Study␣
 ↪-\Ionosphere.csv")
```

[3]:
```python
df.shape
```

[3]:
```
(351, 35)
```

[4]:
```python
df.head()
```

[4]:
```
   V1  V2       V3       V4       V5       V6       V7       V8       V9  \
0   1   0  0.99539 -0.05889  0.85243  0.02306  0.83398 -0.37708  1.00000
1   1   0  1.00000 -0.18829  0.93035 -0.36156 -0.10868 -0.93597  1.00000
2   1   0  1.00000 -0.03365  1.00000  0.00485  1.00000 -0.12062  0.88965
3   1   0  1.00000 -0.45161  1.00000  1.00000  0.71216 -1.00000  0.00000
4   1   0  1.00000 -0.02401  0.94140  0.06531  0.92106 -0.23255  0.77152

       V10  …      V26      V27      V28      V29      V30      V31  \
0  0.03760  … -0.51171  0.41078 -0.46168  0.21266 -0.34090  0.42267
1 -0.04549  … -0.26569 -0.20468 -0.18401 -0.19040 -0.11593 -0.16626
2  0.01198  … -0.40220  0.58984 -0.22145  0.43100 -0.17365  0.60436
3  0.00000  …  0.90695  0.51613  1.00000  1.00000 -0.20099  0.25682
4 -0.16399  … -0.65158  0.13290 -0.53206  0.02431 -0.62197 -0.05707

       V32      V33      V34  Class
0 -0.54487  0.18641 -0.45300      1
1 -0.06288 -0.13738 -0.02447      0
2 -0.24180  0.56045 -0.38238      1
3  1.00000 -0.32382  1.00000      0
4 -0.59573 -0.04608 -0.65697      1

[5 rows x 35 columns]
```

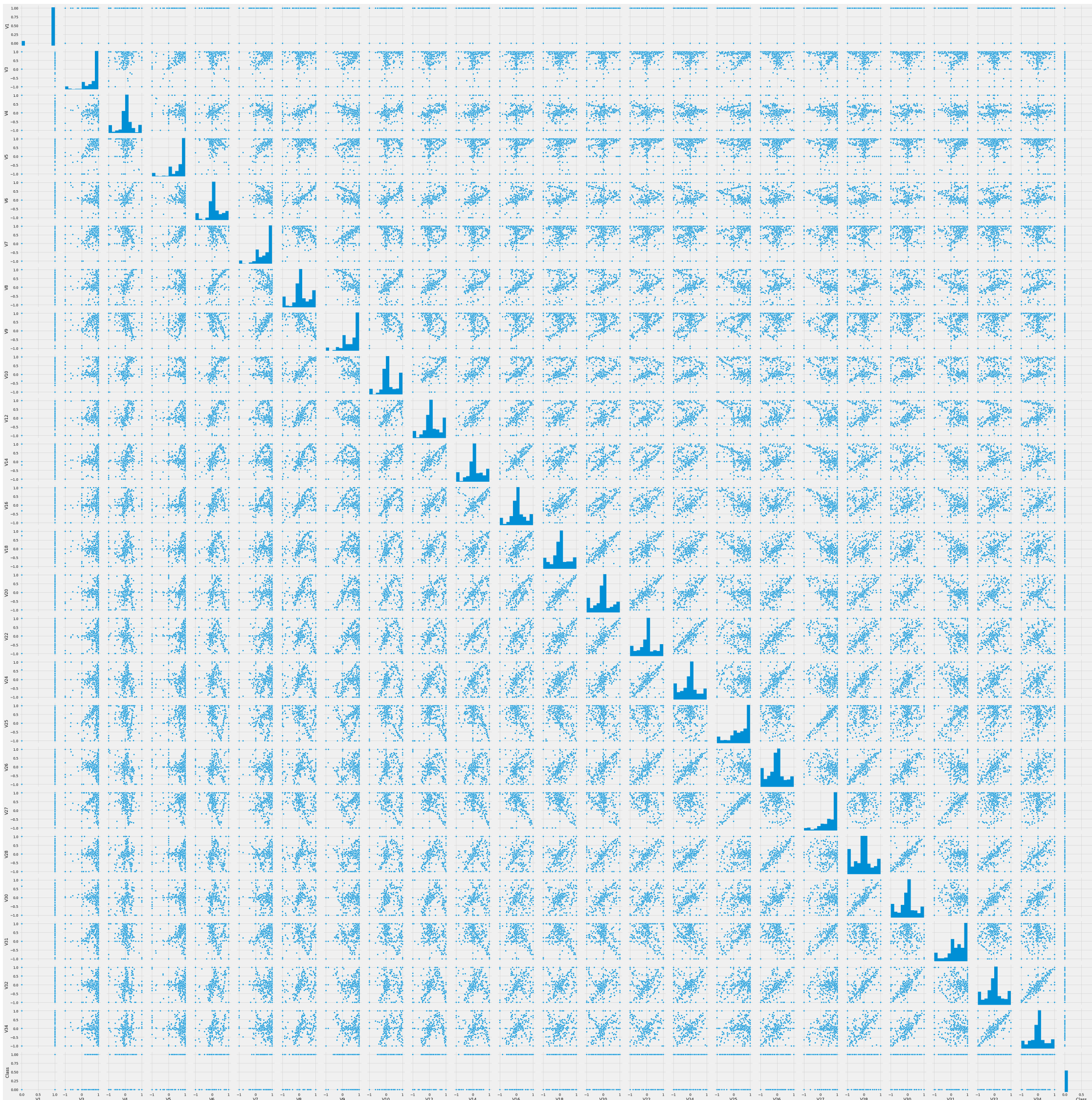
[5]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

[6]:
```
df.head()
```

[6]:
```
   V1  V2       V3       V4       V5       V6       V7       V8       V9  \
0   1   0  0.99539 -0.05889  0.85243  0.02306  0.83398 -0.37708  1.00000
1   1   0  1.00000 -0.18829  0.93035 -0.36156 -0.10868 -0.93597  1.00000
2   1   0  1.00000 -0.03365  1.00000  0.00485  1.00000 -0.12062  0.88965
3   1   0  1.00000 -0.45161  1.00000  1.00000  0.71216 -1.00000  0.00000
4   1   0  1.00000 -0.02401  0.94140  0.06531  0.92106 -0.23255  0.77152

       V10      V11      V12      V13      V14      V15      V16      V17  \
0  0.03760  0.85243 -0.17755  0.59755 -0.44945  0.60536 -0.38223  0.84356
1 -0.04549  0.50874 -0.67743  0.34432 -0.69707 -0.51685 -0.97515  0.05499
2  0.01198  0.73082  0.05346  0.85443  0.00827  0.54591  0.00299  0.83775
3  0.00000  0.00000  0.00000  0.00000  0.00000 -1.00000  0.14516  0.54094
4 -0.16399  0.52798 -0.20275  0.56409 -0.00712  0.34395 -0.27457  0.52940

       V18      V19      V20      V21      V22      V23      V24      V25  \
0 -0.38542  0.58212 -0.32192  0.56971 -0.29674  0.36946 -0.47357  0.56811
1 -0.62237  0.33109 -1.00000 -0.13151 -0.45300 -0.18056 -0.35734 -0.20332
2 -0.13644  0.75535 -0.08540  0.70887 -0.27502  0.43385 -0.12062  0.57528
3 -0.39330 -1.00000 -0.54467 -0.69975  1.00000  0.00000  0.00000  1.00000
4 -0.21780  0.45107 -0.17813  0.05982 -0.35575  0.02309 -0.52879  0.03286

       V26      V27      V28      V29      V30      V31      V32      V33  \
0 -0.51171  0.41078 -0.46168  0.21266 -0.34090  0.42267 -0.54487  0.18641
1 -0.26569 -0.20468 -0.18401 -0.19040 -0.11593 -0.16626 -0.06288 -0.13738
2 -0.40220  0.58984 -0.22145  0.43100 -0.17365  0.60436 -0.24180  0.56045
3  0.90695  0.51613  1.00000  1.00000 -0.20099  0.25682  1.00000 -0.32382
4 -0.65158  0.13290 -0.53206  0.02431 -0.62197 -0.05707 -0.59573 -0.04608

       V34  Class
0 -0.45300      1
1 -0.02447      0
2 -0.38238      1
3  1.00000      0
4 -0.65697      1
```

[31]:
```
sns.pairplot(df)
```

[31]: <seaborn.axisgrid.PairGrid at 0x2977df85d08>



[8]:
```
#Statistic for each Column
df.describe()
```

[8]:

	V1	V2	V3	V4	V5	V6	\
count	351.000000	351.0	351.000000	351.000000	351.000000	351.000000	
mean	0.891738	0.0	0.641342	0.044372	0.601068	0.115889	
std	0.311155	0.0	0.497708	0.441435	0.519862	0.460810	
min	0.000000	0.0	-1.000000	-1.000000	-1.000000	-1.000000	
25%	1.000000	0.0	0.472135	-0.064735	0.412660	-0.024795	
50%	1.000000	0.0	0.871110	0.016310	0.809200	0.022800	
75%	1.000000	0.0	1.000000	0.194185	1.000000	0.334655	
max	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	

	V7	V8	V9	V10	V11	V12 \
count	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000
mean	0.550095	0.119360	0.511848	0.181345	0.476183	0.155040
std	0.492654	0.520750	0.507066	0.483851	0.563496	0.494817
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.211310	-0.054840	0.087110	-0.048075	0.021120	-0.065265
50%	0.728730	0.014710	0.684210	0.018290	0.667980	0.028250
75%	0.969240	0.445675	0.953240	0.534195	0.957895	0.482375
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

	V13	V14	V15	V16	V17	V18 \
count	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000
mean	0.400801	0.093414	0.344159	0.071132	0.381949	-0.003617
std	0.622186	0.494873	0.652828	0.458371	0.618020	0.496762
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.000000	-0.073725	0.000000	-0.081705	0.000000	-0.225690
50%	0.644070	0.030270	0.601940	0.000000	0.590910	0.000000
75%	0.955505	0.374860	0.919330	0.308975	0.935705	0.195285
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

	V19	V20	V21	V22	V23	V24 \
count	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000
mean	0.359390	-0.024025	0.336695	0.008296	0.362475	-0.057406
std	0.626267	0.519076	0.609828	0.518166	0.603767	0.527456
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.000000	-0.234670	0.000000	-0.243870	0.000000	-0.366885
50%	0.576190	0.000000	0.499090	0.000000	0.531760	0.000000
75%	0.899265	0.134370	0.894865	0.188760	0.911235	0.164630
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

	V25	V26	V27	V28	V29	V30 \
count	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000
mean	0.396135	-0.071187	0.541641	-0.069538	0.378445	-0.027907
std	0.578451	0.508495	0.516205	0.550025	0.575886	0.507974
min	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
25%	0.000000	-0.332390	0.286435	-0.443165	0.000000	-0.236885
50%	0.553890	-0.015050	0.708240	-0.017690	0.496640	0.000000
75%	0.905240	0.156765	0.999945	0.153535	0.883465	0.154075
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

	V31	V32	V33	V34	Class
count	351.000000	351.000000	351.000000	351.000000	351.000000
mean	0.352514	-0.003794	0.349364	0.014480	0.641026
std	0.571483	0.513574	0.522663	0.468337	0.480384
min	-1.000000	-1.000000	-1.000000	-1.000000	0.000000
25%	0.000000	-0.242595	0.000000	-0.165350	0.000000
50%	0.442770	0.000000	0.409560	0.000000	1.000000

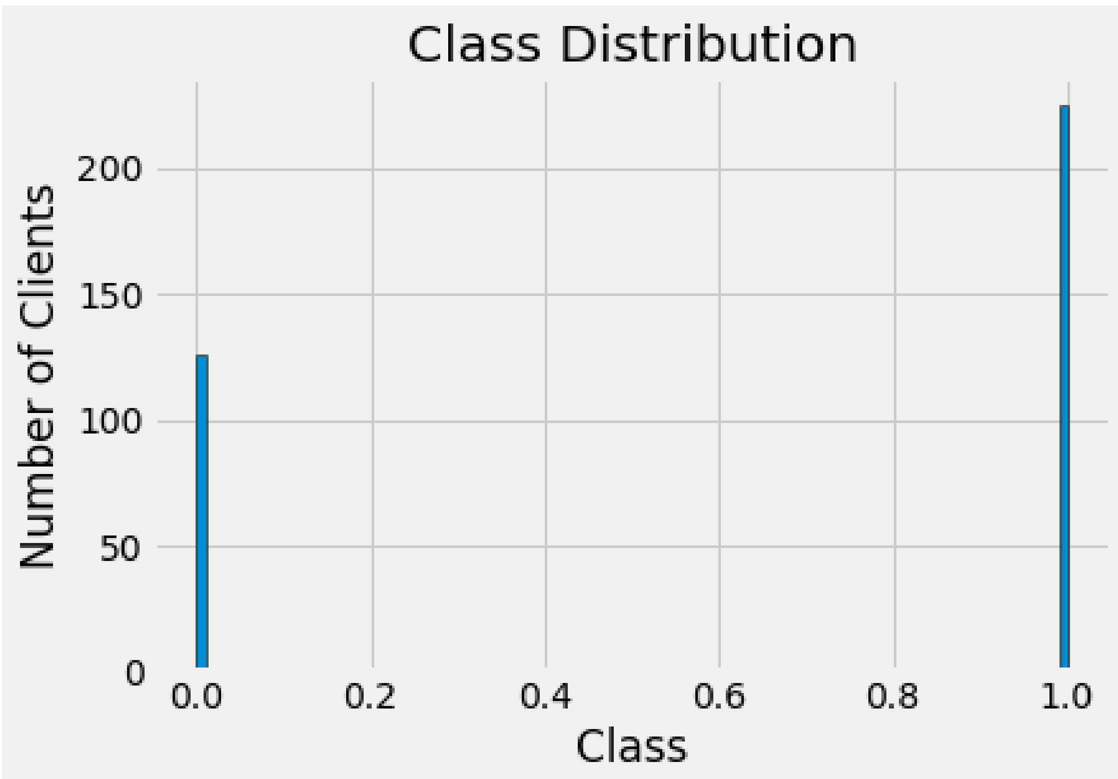
```
75%      0.857620    0.200120    0.813765    0.171660    1.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 351 entries, 0 to 350
Data columns (total 35 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      351 non-null    int64
 1   V2      351 non-null    int64
 2   V3      351 non-null    float64
 3   V4      351 non-null    float64
 4   V5      351 non-null    float64
 5   V6      351 non-null    float64
 6   V7      351 non-null    float64
 7   V8      351 non-null    float64
 8   V9      351 non-null    float64
 9   V10     351 non-null    float64
 10  V11     351 non-null    float64
 11  V12     351 non-null    float64
 12  V13     351 non-null    float64
 13  V14     351 non-null    float64
 14  V15     351 non-null    float64
 15  V16     351 non-null    float64
 16  V17     351 non-null    float64
 17  V18     351 non-null    float64
 18  V19     351 non-null    float64
 19  V20     351 non-null    float64
 20  V21     351 non-null    float64
 21  V22     351 non-null    float64
 22  V23     351 non-null    float64
 23  V24     351 non-null    float64
 24  V25     351 non-null    float64
 25  V26     351 non-null    float64
 26  V27     351 non-null    float64
 27  V28     351 non-null    float64
 28  V29     351 non-null    float64
 29  V30     351 non-null    float64
 30  V31     351 non-null    float64
 31  V32     351 non-null    float64
 32  V33     351 non-null    float64
 33  V34     351 non-null    float64
 34  Class   351 non-null    int64
dtypes: float64(32), int64(3)
memory usage: 96.1 KB
```

[9]:
```python
import matplotlib.pyplot as plt
%matplotlib inline
figsize=(8, 8)

# Histogram of the Class
plt.style.use('fivethirtyeight')
plt.hist(df['Class'], bins = 100, edgecolor = 'k')
plt.xlabel('Class')
plt.ylabel('Number of Clients');
plt.title('Class Distribution')
```

[9]: Text(0.5, 1.0, 'Class Distribution')



[10]:
```python
# # Correlations between Features and Target

# Find all correlations and sort
correlations_df = df.corr()['Class'].sort_values()

# Print the most negative correlations
print(correlations_df.head(15), '\n')

# Print the most positive correlations
```

```python
print(correlations_df.tail(15))
```

```
V22   -0.116385
V27   -0.111107
V34   -0.064168
V32   -0.036004
V30   -0.003942
V26    0.001541
V24    0.006193
V20    0.035620
V28    0.042756
V17    0.087060
V19    0.117435
V18    0.119346
V10    0.120634
V4     0.125884
V16    0.148775
Name: Class, dtype: float64

V14      0.197041
V23      0.204361
V15      0.207201
V8       0.207544
V21      0.219583
V29      0.250036
V33      0.261157
V31      0.294417
V9       0.294852
V7       0.450429
V1       0.465614
V5       0.516477
V3       0.519145
Class    1.000000
V2            NaN
Name: Class, dtype: float64
```

[11]:
```python
for i in df.columns:
    if len(set(df[i]))==1:
        df.drop(labels=[i], axis=1, inplace=True)
```

[12]:
```python
# Find all correlations and sort
correlations_df = df.corr()['Class'].sort_values()

# Print the most negative correlations
print(correlations_df.head(15), '\n')

# Print the most positive correlations
print(correlations_df.tail(15))
```

```
V22   -0.116385
V27   -0.111107
V34   -0.064168
V32   -0.036004
V30   -0.003942
V26    0.001541
V24    0.006193
V20    0.035620
V28    0.042756
V17    0.087060
V19    0.117435
V18    0.119346
V10    0.120634
V4     0.125884
V16    0.148775
Name: Class, dtype: float64

V25      0.188185
V14      0.197041
V23      0.204361
V15      0.207201
V8       0.207544
V21      0.219583
V29      0.250036
V33      0.261157
V31      0.294417
V9       0.294852
V7       0.450429
V1       0.465614
V5       0.516477
V3       0.519145
Class    1.000000
Name: Class, dtype: float64
```

[13]:
```
df.shape
```

[13]:
```
(351, 34)
```

[14]:
```
# # # Feature Engineering and Selection

def remove_collinear_features(x, threshold):
    '''
    Objective:
        Remove collinear features in a dataframe with a correlation coefficient
        greater than the threshold. Removing collinear features can help a model
        to generalize and improves the interpretability of the model.

    Inputs:
```

```
        threshold: any features with correlations greater than this value are␣
↪removed

    Output:
        dataframe that contains only the non-highly-collinear features
    '''

    # Dont want to remove correlations between Class
    y = x['Class']
    x = x.drop(columns = ['Class'])

    # Calculate the correlation matrix
    corr_matrix = x.corr()
    iters = range(len(corr_matrix.columns) - 1)
    drop_cols = []

    # Iterate through the correlation matrix and compare correlations
    for i in iters:
        for j in range(i):
            item = corr_matrix.iloc[j:(j+1), (i+1):(i+2)]
            col = item.columns
            row = item.index
            val = abs(item.values)

            # If correlation exceeds the threshold
            if val >= threshold:
                # Print the correlated features and the correlation value
                # print(col.values[0], "|", row.values[0], "|",␣
↪round(val[0][0], 2))
                drop_cols.append(col.values[0])

    # Drop one of each pair of correlated columns
    drops = set(drop_cols)
    x = x.drop(columns = drops)

    # Add the score back in to the data
    x['Class'] = y

    return x
```

[15]:
```
# Remove the collinear features above a specified correlation coefficient
df = remove_collinear_features(df, 0.6);
```

[16]:
```
df.shape
```

[16]: (351, 25)

[17]:
```python
from sklearn.model_selection import train_test_split

# # # Split Into Training and Testing Sets

# Separate out the features and targets
features = df.drop(columns='Class')
targets = pd.DataFrame(df['Class'])

# Split into 80% training and 20% testing set
X_train, X_test, y_train, y_test = train_test_split(features, targets, test_size = 0.2, random_state = 42)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(280, 24)
(71, 24)
(280, 1)
(71, 1)
```

[18]:
```python
# # Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[19]:
```python
# Convert y to one-dimensional array (vector)
y_train = np.array(y_train).reshape((-1, ))
y_test = np.array(y_test).reshape((-1, ))
```

[20]:
```python
X_train
```

[20]:
```
array([[ 0.35284273,  0.70670705,  0.53265652, …, -0.88625238,
         1.92502166,  2.18430028],
       [ 0.35284273,  0.47583927, -0.5080449 , …,  0.13719315,
         1.35747018,  1.17715782],
       [ 0.35284273,  0.6996991 ,  0.13363531, …,  0.7149698 ,
         0.2840665 ,  0.25014271],
       …,
       [ 0.35284273,  0.70670705,  0.16846513, …,  0.98020209,
        -0.09448333, -0.09472362],
       [ 0.35284273,  0.60210095, -0.01294299, …,  0.97486423,
        -0.01824245, -0.02666483],
       [-2.83412386, -1.26736552, -0.01217651, …, -0.59190758,
        -0.02686981, -0.01398075]])
```

[21]:
```
X_test
```

[21]:
```
array([[ 0.35284273, -0.33777475,  0.50071228, …, -0.17050478,
        -0.50194067,  0.07907248],
       [ 0.35284273,  0.35638814, -0.18287643, …,  0.59526619,
        -0.10923963,  0.01270638],
       [ 0.35284273,  0.50118636,  0.87347023, …,  0.81698222,
        -0.16406826,  0.59991121],
       …,
       [ 0.35284273,  0.30887221,  0.85045325, …,  1.10265089,
        -0.39284946, -0.14016208],
       [ 0.35284273,  0.70670705,  2.24218061, …,  0.06337818,
        -0.02206815,  1.74002768],
       [-2.83412386,  0.70670705, -2.26653363, …,  1.10265089,
         1.92502166,  2.18430028]])
```

[22]:
```
# Function to calculate mean absolute error
def cross_val(X_train, y_train, model):
    # Applying k-Fold Cross Validation
    from sklearn.model_selection import cross_val_score
    accuracies = cross_val_score(estimator = model, X = X_train, y = y_train,␣
 ↪cv = 5)
    return accuracies.mean()

# Takes in a model, trains the model, and evaluates the model on the test set
def fit_and_evaluate(model):

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions and evalute
    model_pred = model.predict(X_test)
    model_cross = cross_val(X_train, y_train, model)

    # Return the performance metric
    return model_cross
```

[23]:
```
# # Naive Bayes
from sklearn.naive_bayes import GaussianNB
naive = GaussianNB()
naive_cross = fit_and_evaluate(naive)

print('Naive Bayes Performance on the test set: Cross Validation Score = %0.4f'␣
 ↪% naive_cross)
```

```
Naive Bayes Performance on the test set: Cross Validation Score = 0.8893
```

[24]:
```
# # Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
random = RandomForestClassifier(n_estimators = 10, criterion = 'entropy')
random_cross = fit_and_evaluate(random)

print('Random Forest Performance on the test set: Cross Validation Score = %0.
 ↪4f' % random_cross)
```

```
Random Forest Performance on the test set: Cross Validation Score = 0.9357
```

[25]:
```
# # Gradiente Boosting Classification
from xgboost import XGBClassifier
gb = XGBClassifier()
gb_cross = fit_and_evaluate(gb)

print('Gradiente Boosting Classification Performance on the test set: Cross␣
 ↪Validation Score = %0.4f' % gb_cross)
```

```
C:\Users\admin\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in a
future release. To remove this warning, do the following: 1) Pass option
use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, …, [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\admin\anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning:
The use of label encoder in XGBClassifier is deprecated and will be removed in a
future release. To remove this warning, do the following: 1) Pass option
use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, …, [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[12:18:44] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:18:44] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:18:44] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:18:44] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from
```

```
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:18:44] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
[12:18:44] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from
'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the
old behavior.
Gradiente Boosting Classification Performance on the test set: Cross Validation
Score = 0.9250
```

[26]:
```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)

classifier.score(X_test, y_test)
```

[26]: 0.8450704225352113

[27]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
lr = LinearRegression()
lr_lasso = Lasso()
lr_ridge = Ridge()

def rmse(y_test, y_pred):
    return np.sqrt(mean_squared_error(y_test, y_pred))
```

[28]:
```
#Ridge
lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test)
lr_rmse = rmse(y_test, lr.predict(X_test))
lr_score, lr_rmse
```

[28]: (0.42866763283334197, 0.36940205657606945)

[29]:
```
# Lasso
lr_lasso.fit(X_train, y_train)
lr_lasso_score=lr_lasso.score(X_test, y_test)
lr_lasso_rmse = rmse(y_test, lr_lasso.predict(X_test))
lr_lasso_score, lr_lasso_rmse
```

[29]: (-0.00824127906976746, 0.4907238114814987)

1 Conclusion

1.1 Classification Algorithm is better than Regression Technique

[ ]: