Logistic-Regression-prediction-by-retzam-ai

May 2, 2024

[5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
```

[6]:
```
df = pd.read_csv('machine_failure_cleaned.csv', header=0)
df.head()
```

[6]:

	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	TWF	HDF	PWF	OSF	Machine failure
0	1551	42.8	0	0	0	0	0	0
1	1408	46.3	3	0	0	0	0	0
2	1498	49.4	5	0	0	0	0	0
3	1433	39.5	7	0	0	0	0	0
4	1408	40.0	9	0	0	0	0	0

[7]:
```
header = df.columns
header
```

[7]:
```
Index(['Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]', 'TWF',
       'HDF', 'PWF', 'OSF', 'Machine failure'],
      dtype='object')
```

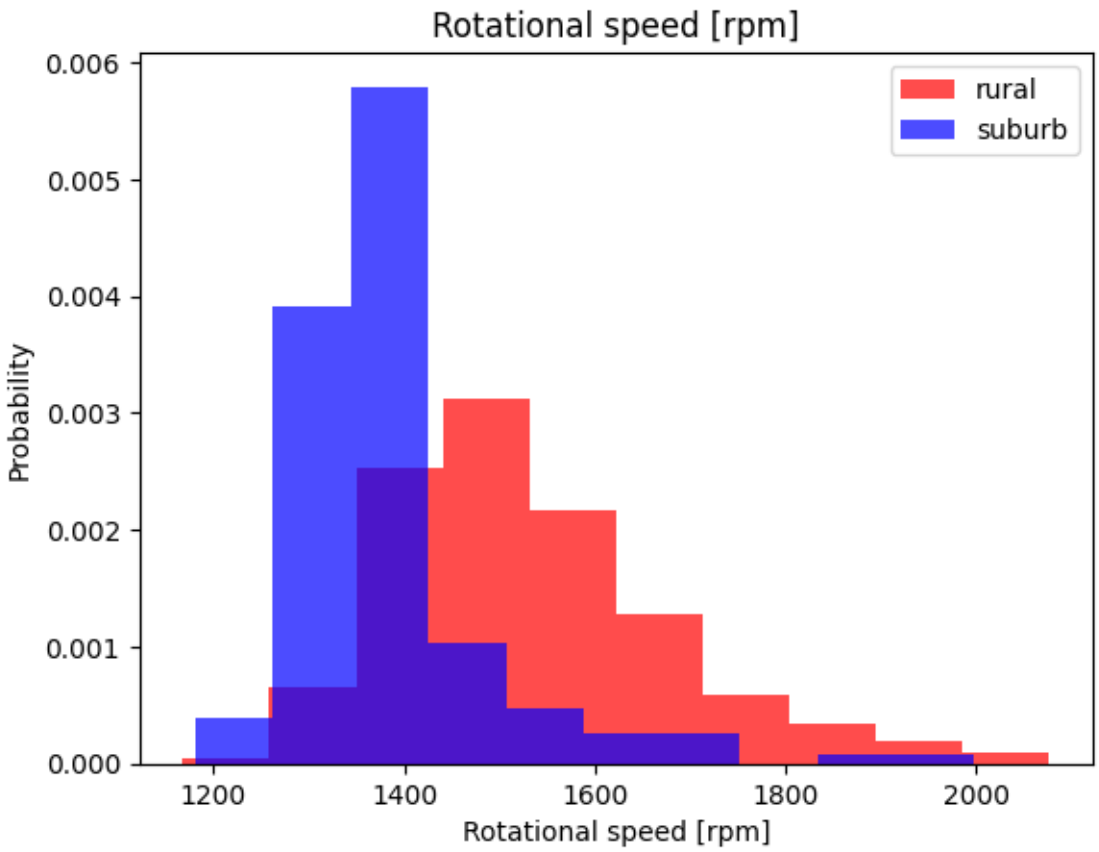
[8]:
```
df.head()
```

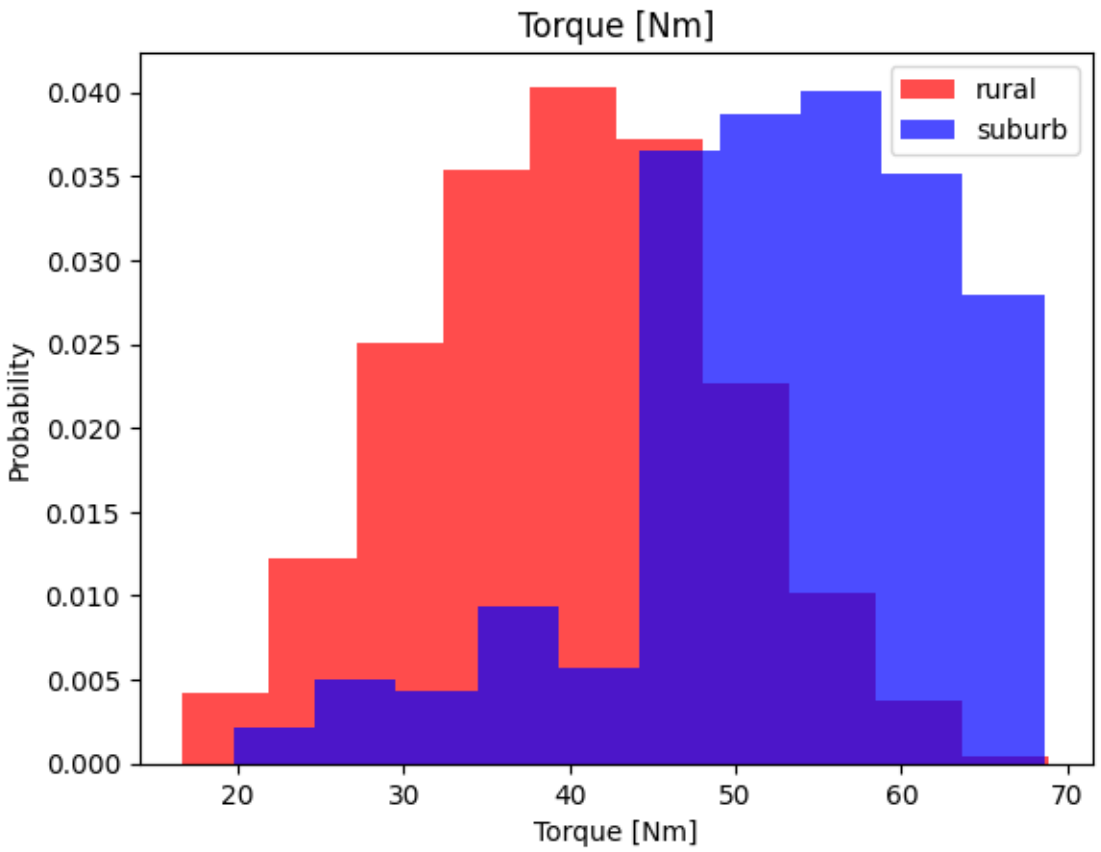
[8]:

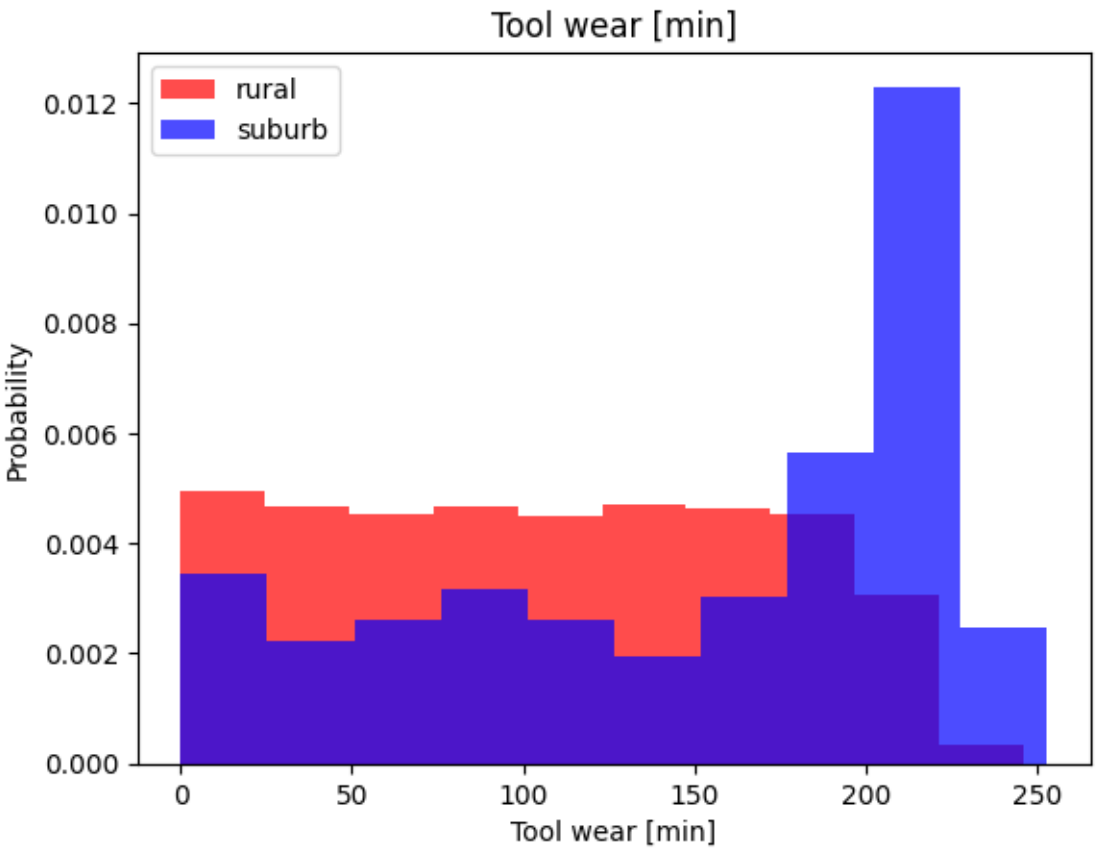
	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	TWF	HDF	PWF	OSF	Machine failure
0	1551	42.8	0	0	0	0	0	0
1	1408	46.3	3	0	0	0	0	0
2	1498	49.4	5	0	0	0	0	0
3	1433	39.5	7	0	0	0	0	0
4	1408	40.0	9	0	0	0	0	0

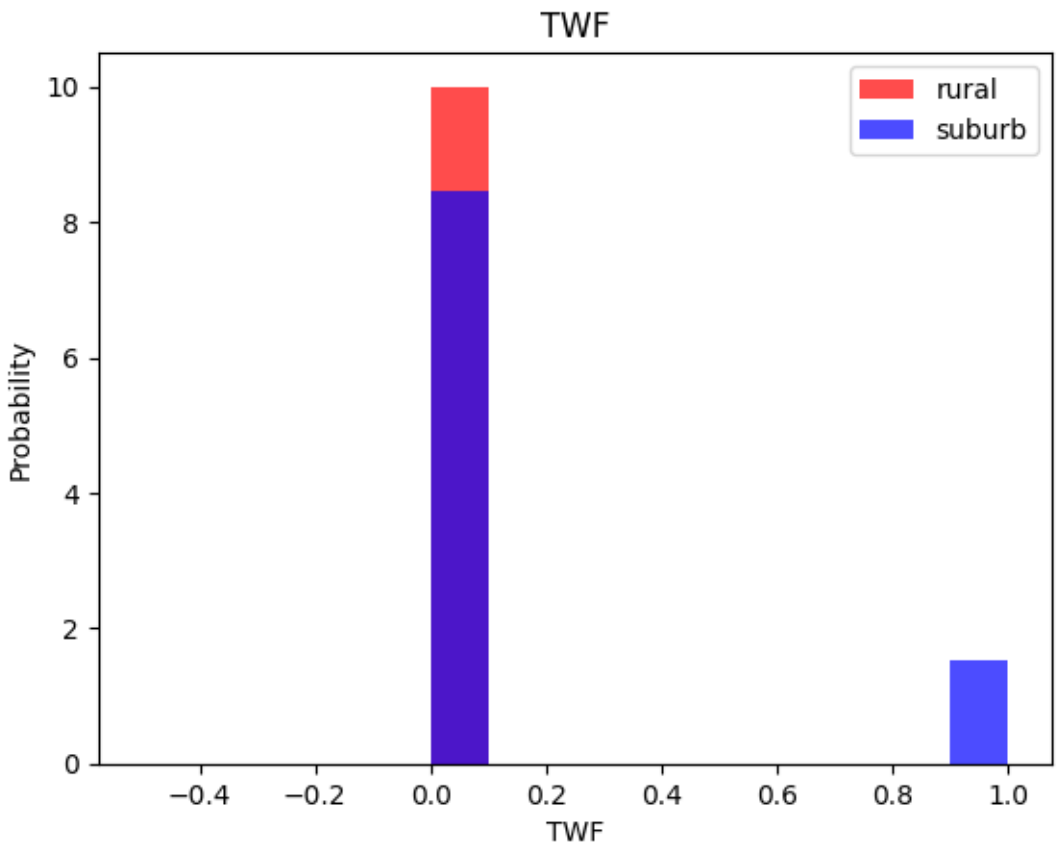
```
   Machine failure
0                0
1                0
2                0
3                0
4                0
```

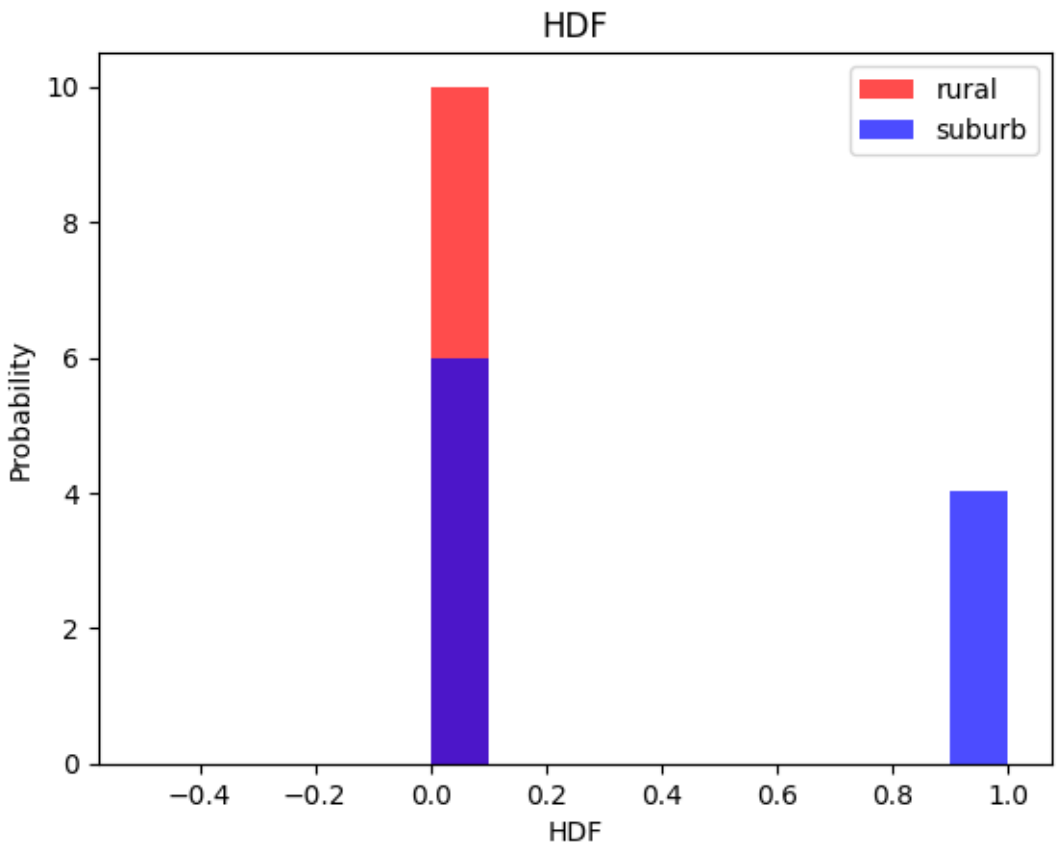
[9]:
```
# We plot a histogram to check which features affect the outcome the most or␣
 ↪the least
# This helps us determine, which features to use in training our model and the␣
 ↪ones to discard

for label in header[:-1]:
  plt.hist(df[df['Machine failure'] == 0][label], color = 'red', label='rural',␣
 ↪alpha=0.7, density=True)
  plt.hist(df[df['Machine failure'] == 1][label], color = 'blue',␣
 ↪label='suburb', alpha=0.7, density=True)
  plt.title(label)
  plt.ylabel('Probability')
  plt.xlabel(label)
  plt.legend()
  plt.show()
```

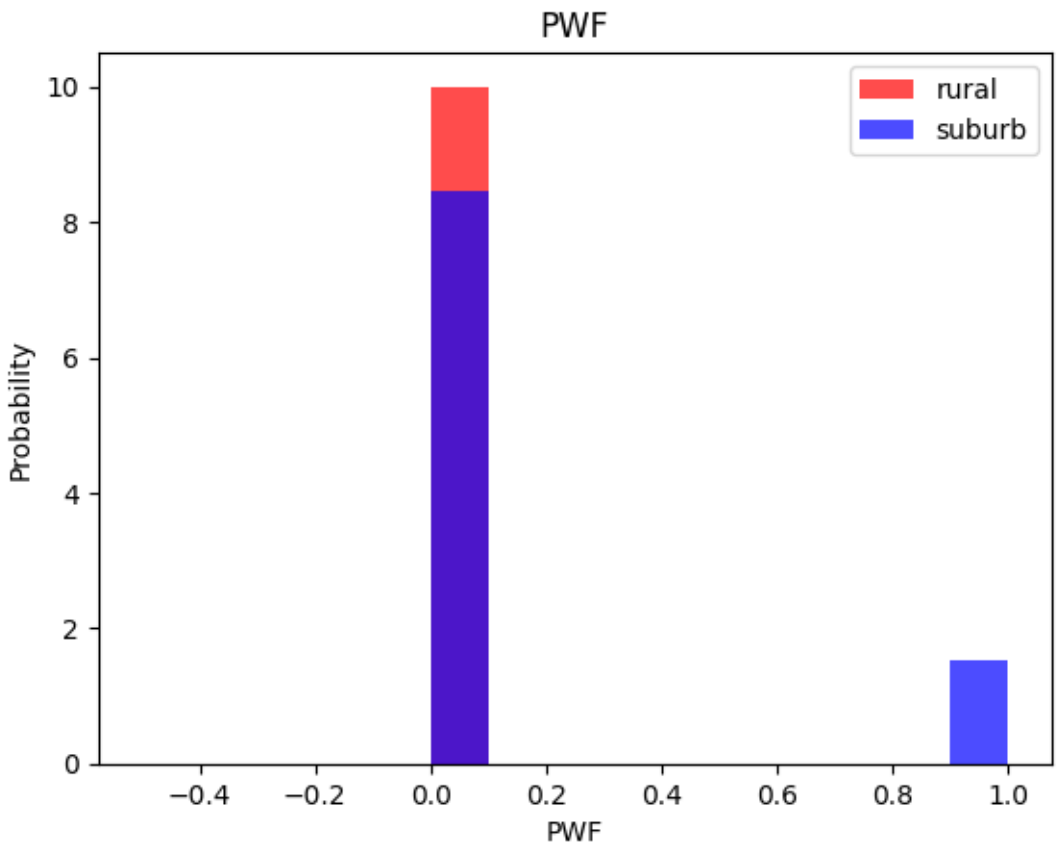

Rotational speed [rpm]
rural
suburb
Probability
0.006
0.005
0.004
0.003
0.002
0.001
0.000
1200
1400
1600
1800
2000
Rotational speed [rpm]

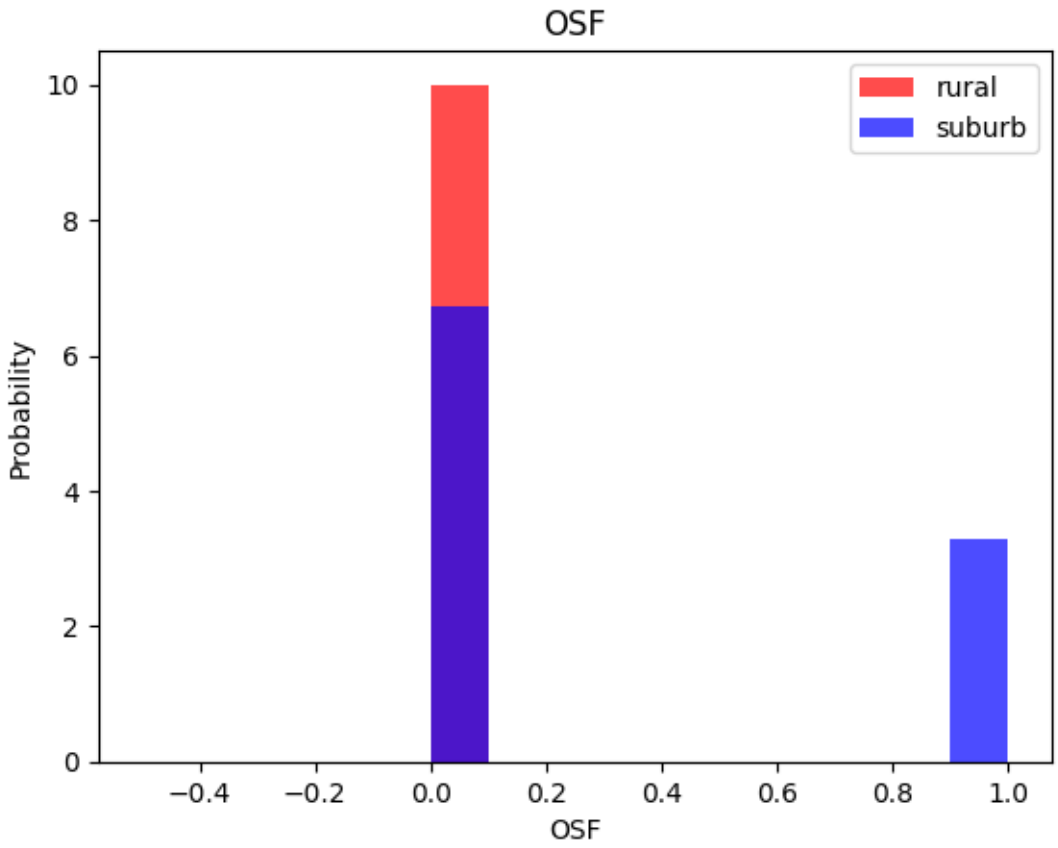

Torque [Nm]
rural
suburb
Probability
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
20
30
40
50
60
70
Torque [Nm]


Tool wear [min]
rural
suburb
0.012
0.010
0.008
0.006
0.004
0.002
0.000
Probability
0
50
100
150
200
250
Tool wear [min]


TWF
rural
suburb
Probability
10
8
6
4
2
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
TWF


HDF
rural
suburb
Probability
10
8
6
4
2
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
HDF


PWF
rural
suburb
Probability
10
8
6
4
2
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
PWF



[10]:
```
# Split dataset to train and tests sets, 80%-20%
train, test = np.split(df.sample(frac=1), [int(0.8 *len(df))])
```

[11]:
```
# Scale dataset so better prediction can be made.
def scale_dataset(dataframe, oversample=False):
  # This selects all columns in the DataFrame except the last one as the␣
 ↪features.
  X = dataframe[dataframe.columns[:-1]].values

  # This selects the last column in the DataFrame as the target.
  y = dataframe[dataframe.columns[-1]].values

  # This removes the mean and scaling to unit variance
  # Known as standardization. Basically removes outliers.
  scaler = StandardScaler()
  X = scaler.fit_transform(X)

  """
    Make both x and y sets equal sets as appropriate.
```

```
    RandomOverSampler is important in cases where there is alot more features␣
↪vector of a
    specific output.

    Example if you have a dataset with 100 rows with output as "Yes" and 20
    rows with "No".
    You can see that our datasets would be biased towards the output with "Yes".
    To solve this, RandomOverSampler strategically duplicates rows with "No" so␣
↪the dataset ends up
    having 100 rows with "Yes" and 100 with "No" outputs.

    This is called over-sampling.
  """
  if oversample:
    ros = RandomOverSampler()
    X, y = ros.fit_resample(X, y)

  # Stack horizontally
  # Reshape y and concatenate it with X
  # This simply means attaching each feature vector with the appropriate output.
  data = np.hstack((X, np.reshape(y, (-1, 1))))

  return data, X, y
```

[12]:
```
train, X_train, y_train = scale_dataset(train, oversample=True)

# test sets are not oversampled because they
# are used to test new data
test, X_test, y_test = scale_dataset(test, oversample=False)
```

[13]:
```
# We'll use LogistionRegression from sklearn linear model library
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

[14]:
```
log_reg_model = LogisticRegression()
log_reg_model.fit(X_train, y_train)
```

[14]:
```
LogisticRegression()
```

[15]:
```
y_pred = log_reg_model.predict(X_test)
y_pred
```

[15]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[16]:
```
# Check model performance with classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1919
           1       1.00      1.00      1.00        44

    accuracy                           1.00      1963
   macro avg       1.00      1.00      1.00      1963
weighted avg       1.00      1.00      1.00      1963
```

[ ]: