SVM-prediction-by-retzam-ai

May 9, 2024

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
```

[4]:
```
df = pd.read_csv('campaign_responses.csv', header=0)
df.head()
```

[4]:

	customer_id	age	gender	annual_income	credit_score	employed	marital_status	no_of_children	responded
0	1	35	Male	65000	720	Yes	Married	2	Yes
1	2	28	Female	45000	680	No	Single	0	No
2	3	42	Male	85000	750	Yes	Married	3	Yes
3	4	31	Female	55000	710	Yes	Single	1	No
4	5	47	Male	95000	790	Yes	Married	2	Yes

[5]:
```
# Convert each column with nominal data to numbers from 0, 1, 2...
df["gender"], _ = pd.factorize(df["gender"])
df["marital_status"], _ = pd.factorize(df["marital_status"])
df["employed"], _ = pd.factorize(df["employed"])

# Remove not needed columns
df = df.drop('customer_id', axis=1)

df.head()
```

[5]:

	age	gender	annual_income	credit_score	employed	marital_status	no_of_children	responded
0	35	0	65000	720	0	0	2	Yes
1	28	1	45000	680	1	1	0	No
2	42	0	85000	750	0	0	3	Yes
3	31	1	55000	710	0	1	1	No

```
4   47       0          95000           790         0                0

   no_of_children responded
0               2       Yes
1               0        No
2               3       Yes
3               1        No
4               2       Yes
```

[6]:
```
# Create a map using the unique values array above.
mapping = {
   'No': 0,
   'Yes': 1,
}

# Replace the values
df['responded'] = df['responded'].replace(mapping)
df.head()
```

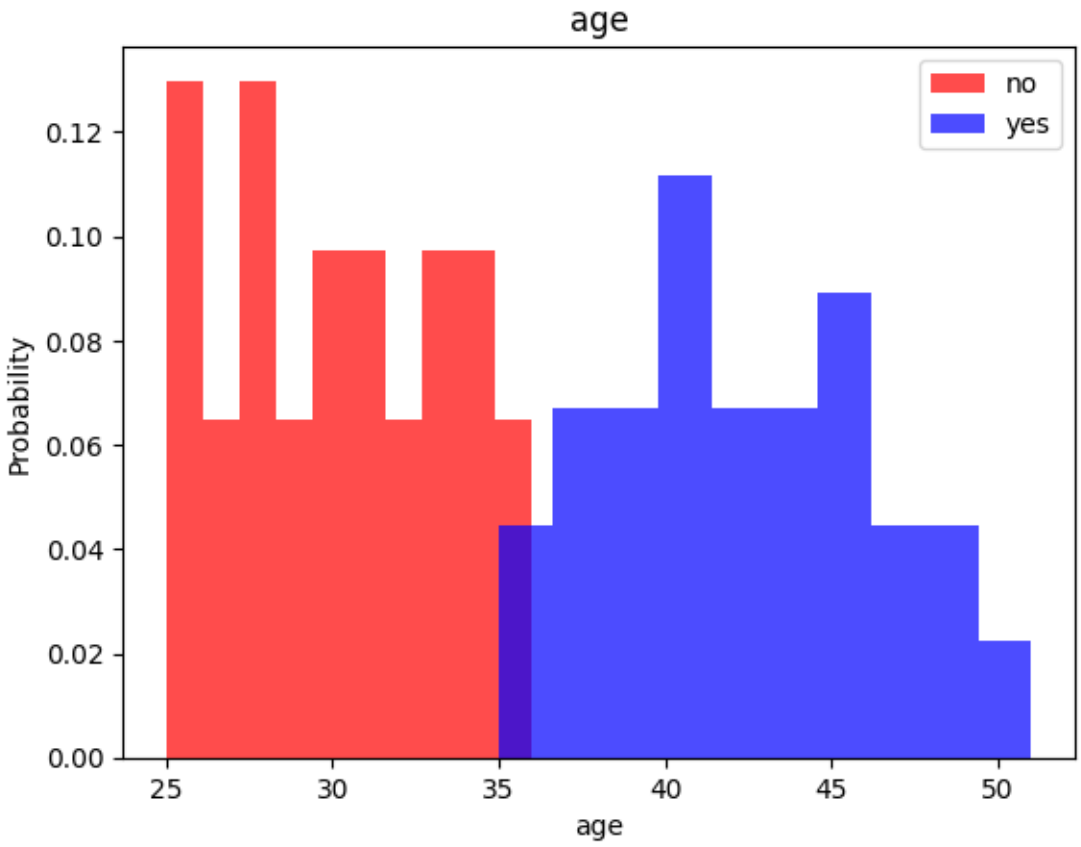
[6]:
```
   age  gender  annual_income  credit_score  employed  marital_status  \
0   35       0          65000           720         0               0
1   28       1          45000           680         1               1
2   42       0          85000           750         0               0
3   31       1          55000           710         0               1
4   47       0          95000           790         0               0

   no_of_children  responded
0               2          1
1               0          0
2               3          1
3               1          0
4               2          1
```

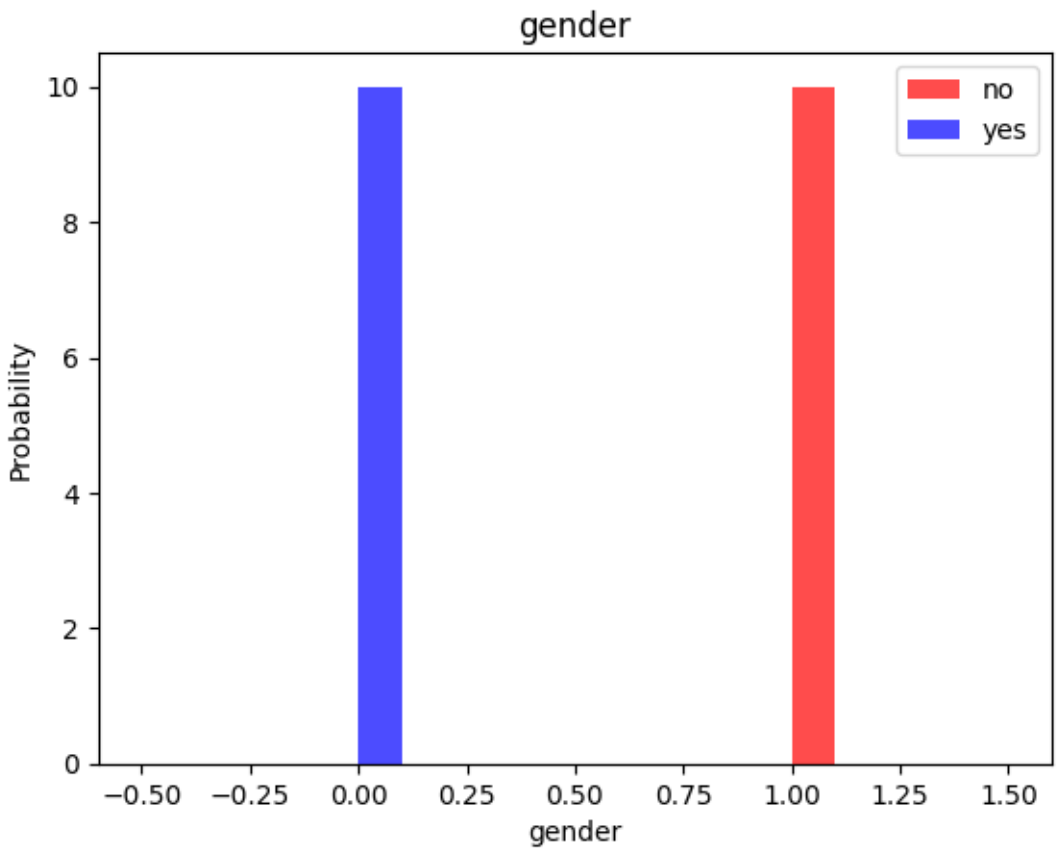
[7]:
```
header = df.columns
header
```

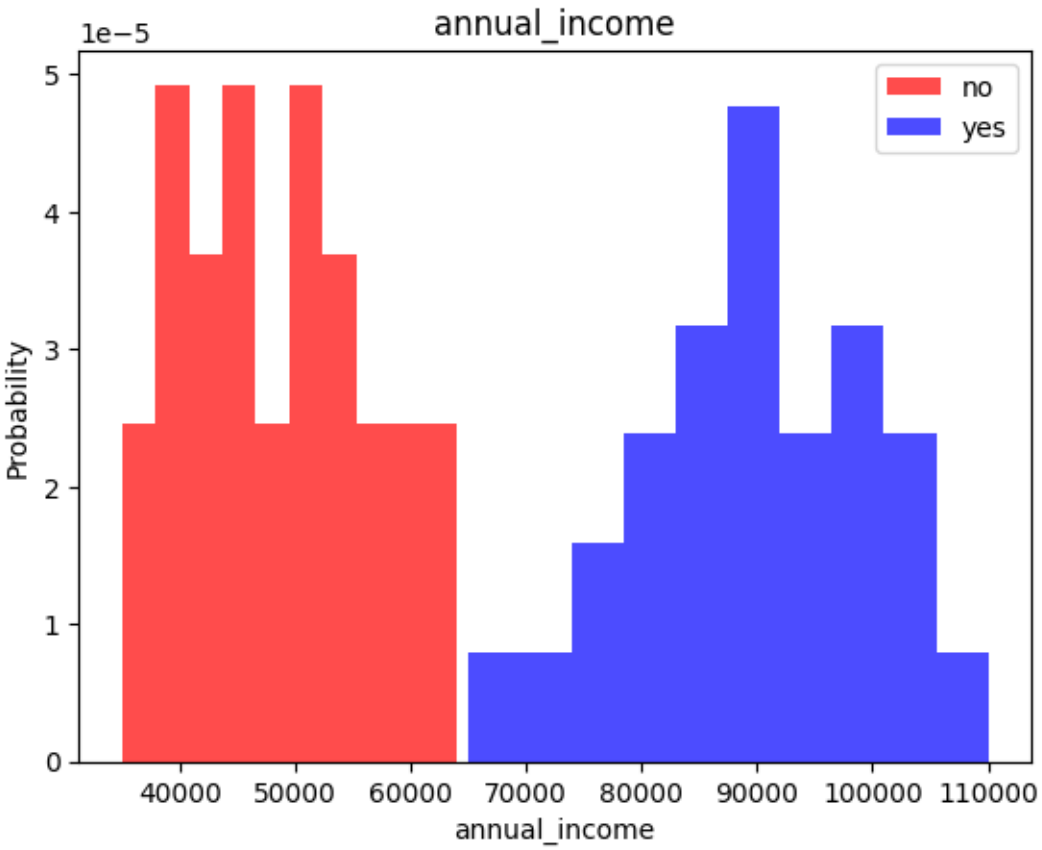
[7]:
```
Index(['age', 'gender', 'annual_income', 'credit_score', 'employed',
       'marital_status', 'no_of_children', 'responded'],
      dtype='object')
```

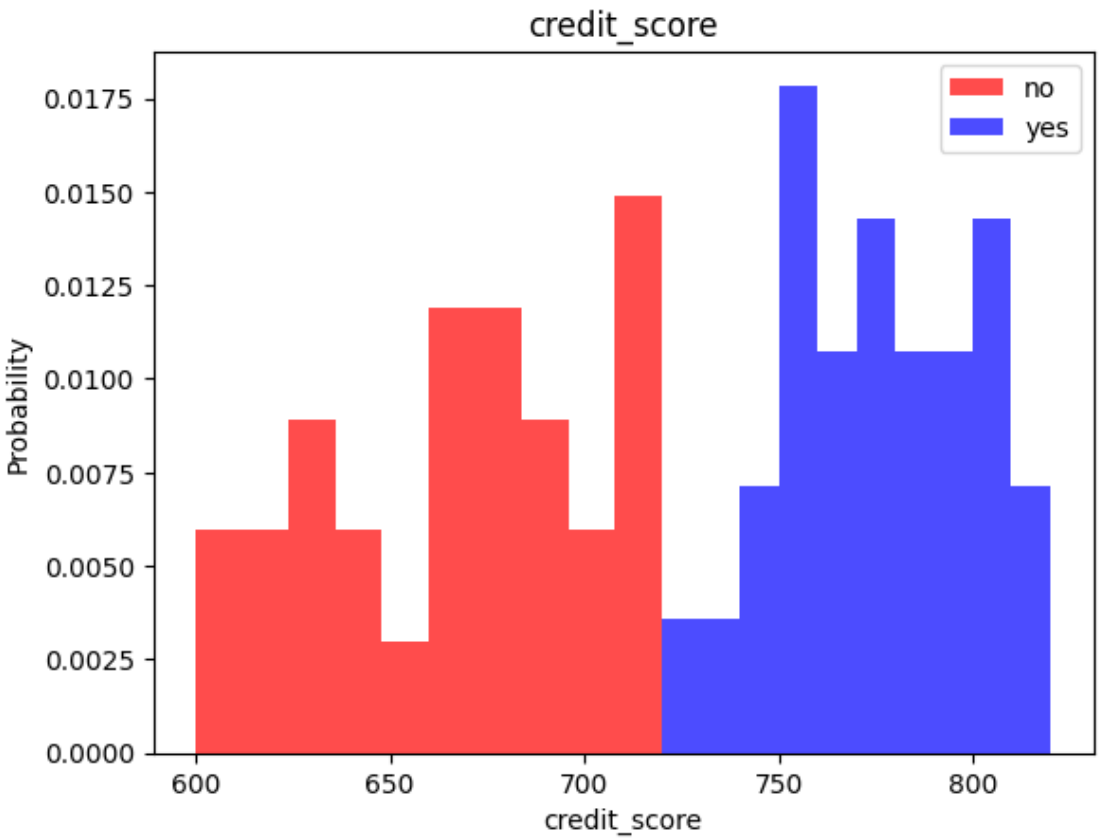
[17]:
```
# We plot a histogram to check which features affect the outcome the most or␣
 ↪the least
# This helps us determine, which features to use in training our model and the␣
 ↪ones to discard

for label in header[:-1]:
```

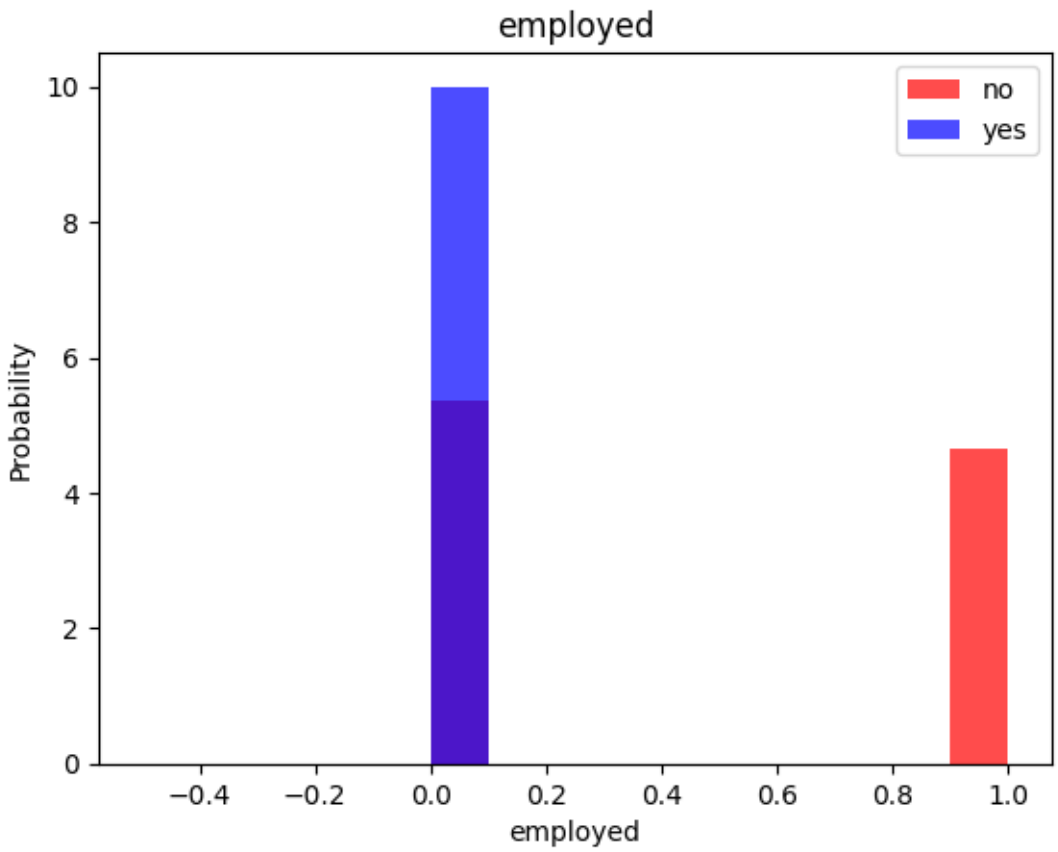
```
plt.hist(df[df['responded'] == 0][label], color = 'red', label='no', alpha=0.
↪7, density=True)
plt.hist(df[df['responded'] == 1][label], color = 'blue', label='yes',␣
↪alpha=0.7, density=True)
plt.title(label)
plt.ylabel('Probability')
plt.xlabel(label)
plt.legend()
plt.show()
```

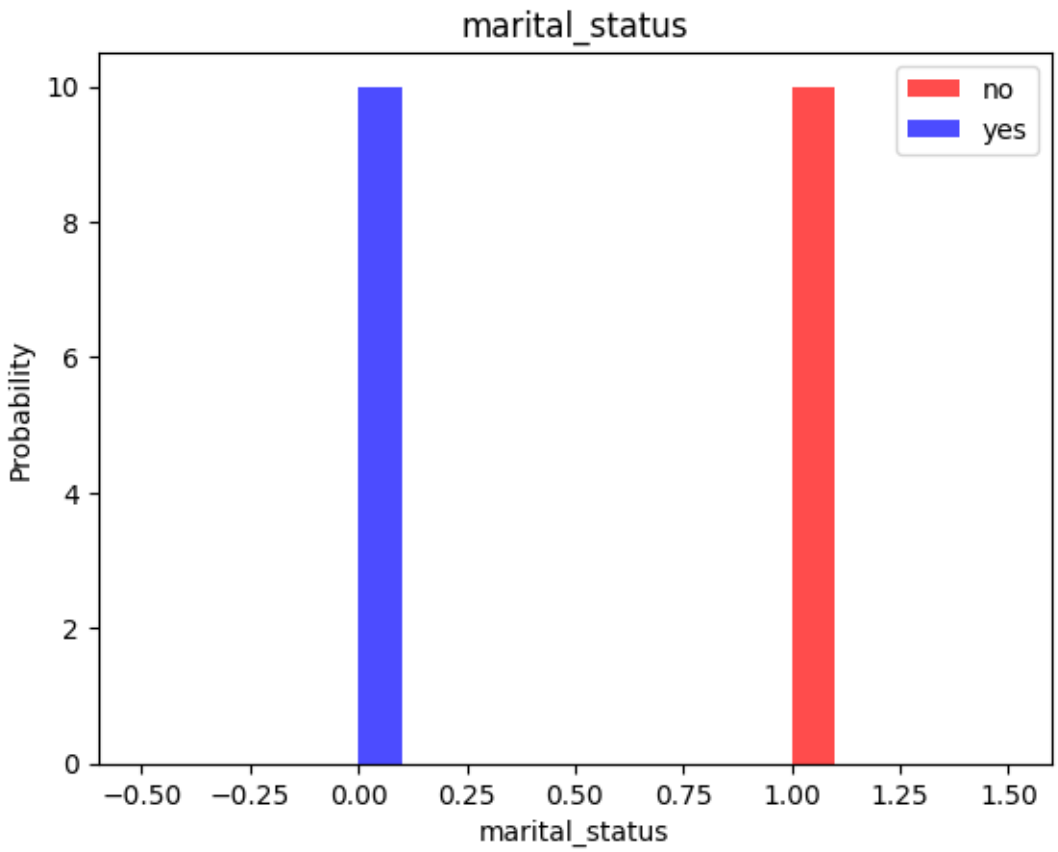


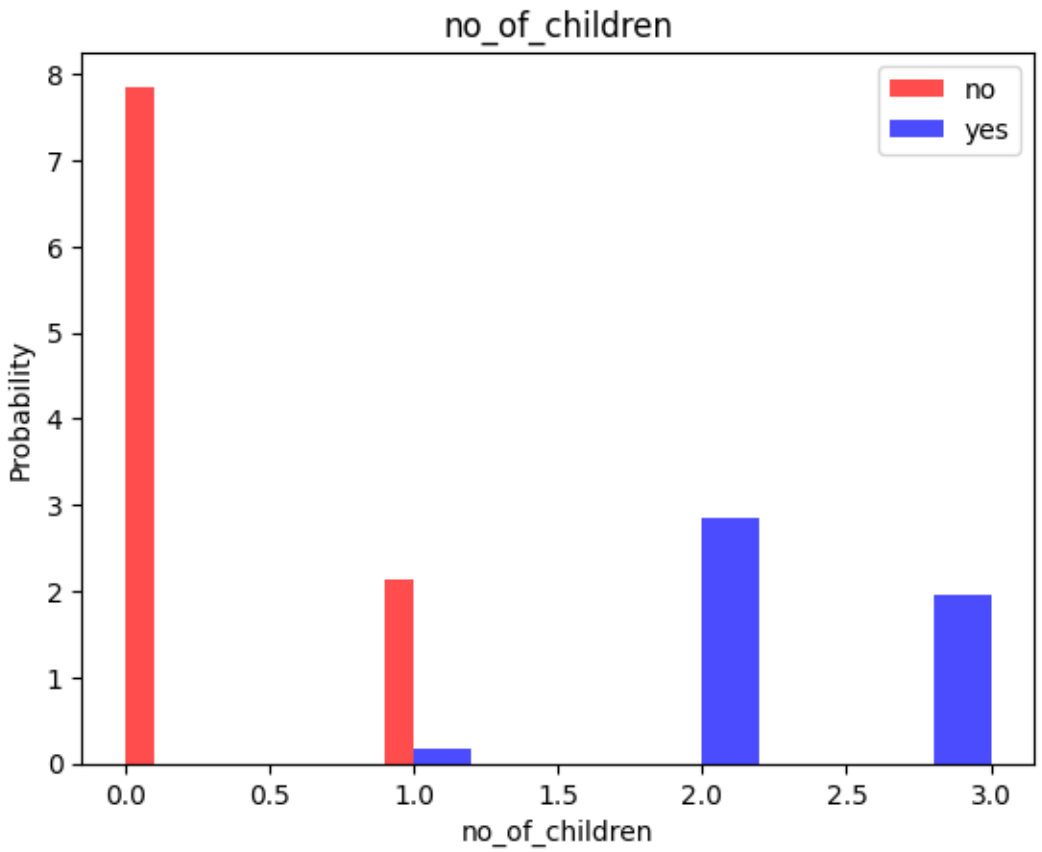

gender
no
yes
Probability
0
2
4
6
8
10
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
gender


annual_income
1e−5
5
4
3
2
1
0
Probability
no
yes
40000
50000
60000
70000
80000
90000
100000
110000
annual_income


credit_score
no
yes
Probability
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
600
650
700
750
800
credit_score


employed
no
yes
Probability
10
8
6
4
2
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
employed


marital_status
no
yes
Probability
10
8
6
4
2
0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
marital_status



[9]:
```
train, test = np.split(df.sample(frac=1), [int(0.8 *len(df))])
```

[10]:
```
# 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
```

[11]:
```
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]:
```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

[14]:
```
svm_model = SVC()
svm_model = svm_model.fit(X_train, y_train)
```

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

[15]:
```
array([1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1])
```

[16]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       1.00      1.00      1.00         7
```

```
    accuracy                           1.00        12
   macro avg       1.00      1.00      1.00        12
weighted avg       1.00      1.00      1.00        12
```

[ ]: