Naive-Bayes-prediction-by-retzam-ai

April 17, 2024

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
```

[2]:
```
df = pd.read_csv('injury_data.csv', header=0)
df.head()
```

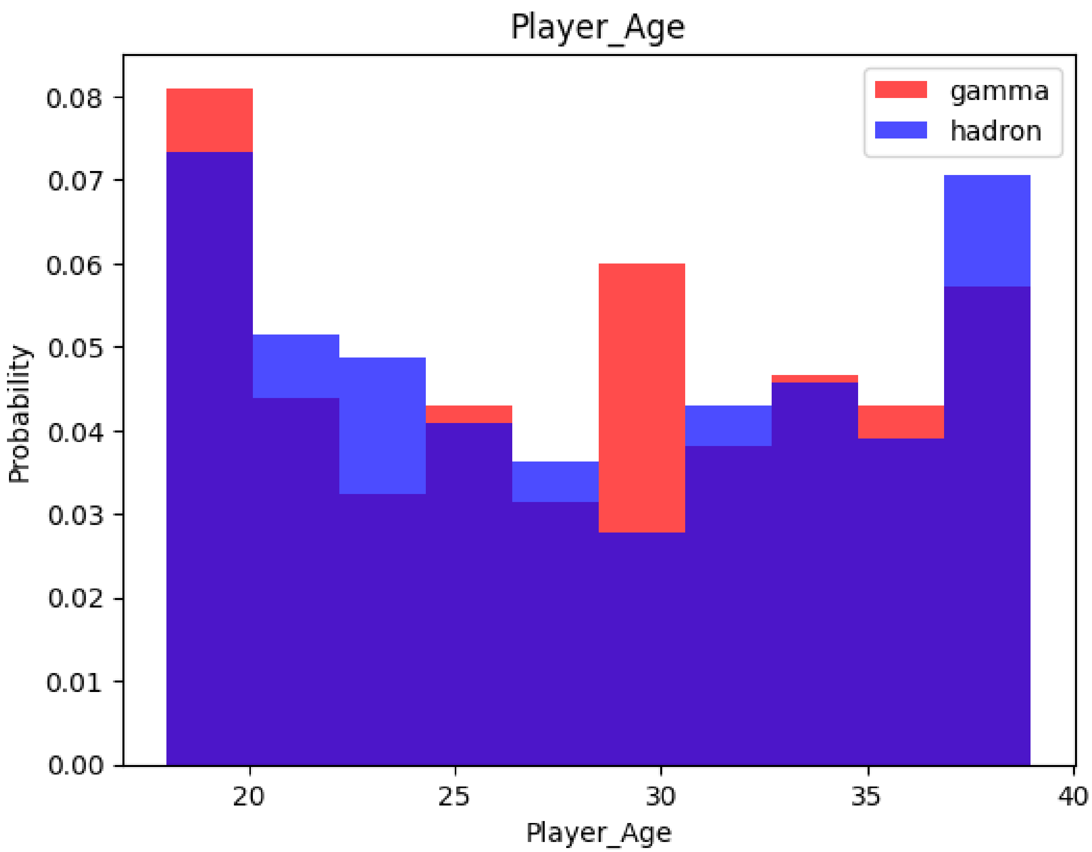
[2]:
```
   Player_Age  Player_Weight  Player_Height  Previous_Injuries  \
0          24      66.251933     175.732429                  1
1          37      70.996271     174.581650                  0
2          32      80.093781     186.329618                  0
3          28      87.473271     175.504240                  1
4          25      84.659220     190.175012                  0

   Training_Intensity  Recovery_Time  Likelihood_of_Injury
0            0.457929              5                     0
1            0.226522              6                     1
2            0.613970              2                     1
3            0.252858              4                     1
4            0.577632              1                     1
```

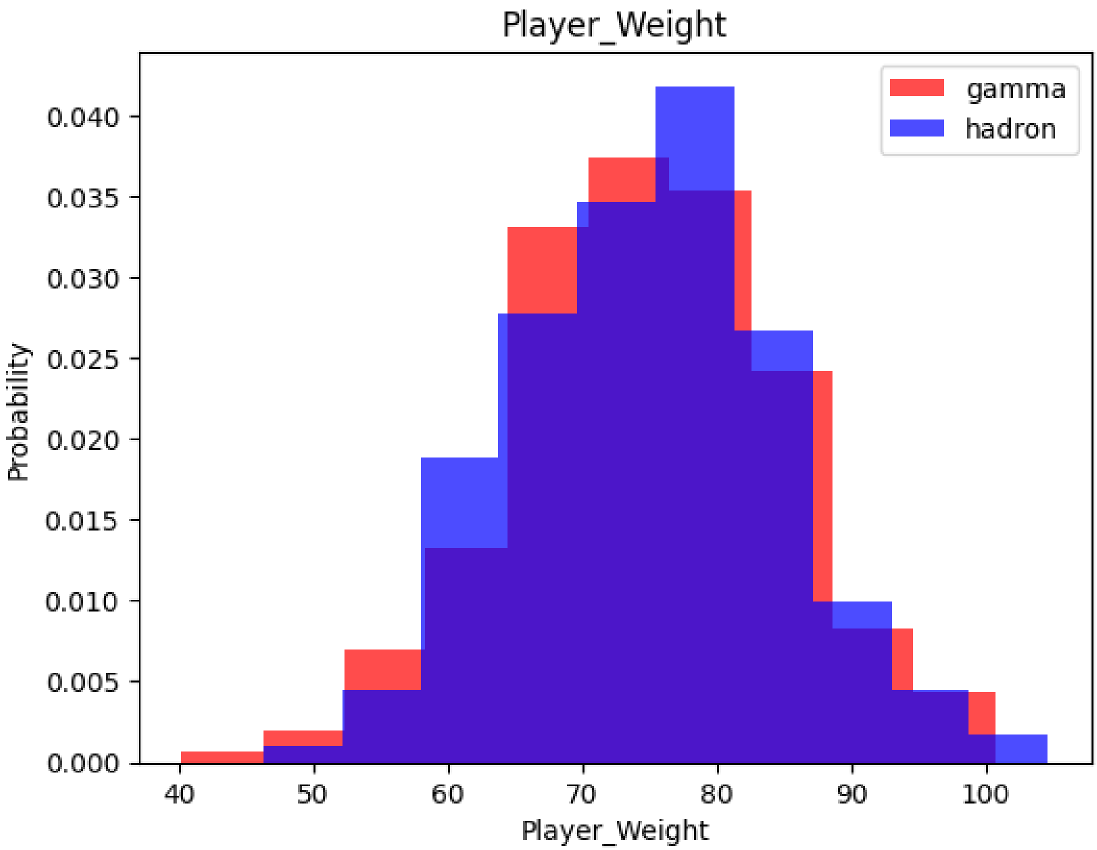
[3]:
```
header = df.columns
header
```

[3]:
```
Index(['Player_Age', 'Player_Weight', 'Player_Height', 'Previous_Injuries',
       'Training_Intensity', 'Recovery_Time', 'Likelihood_of_Injury'],
      dtype='object')
```

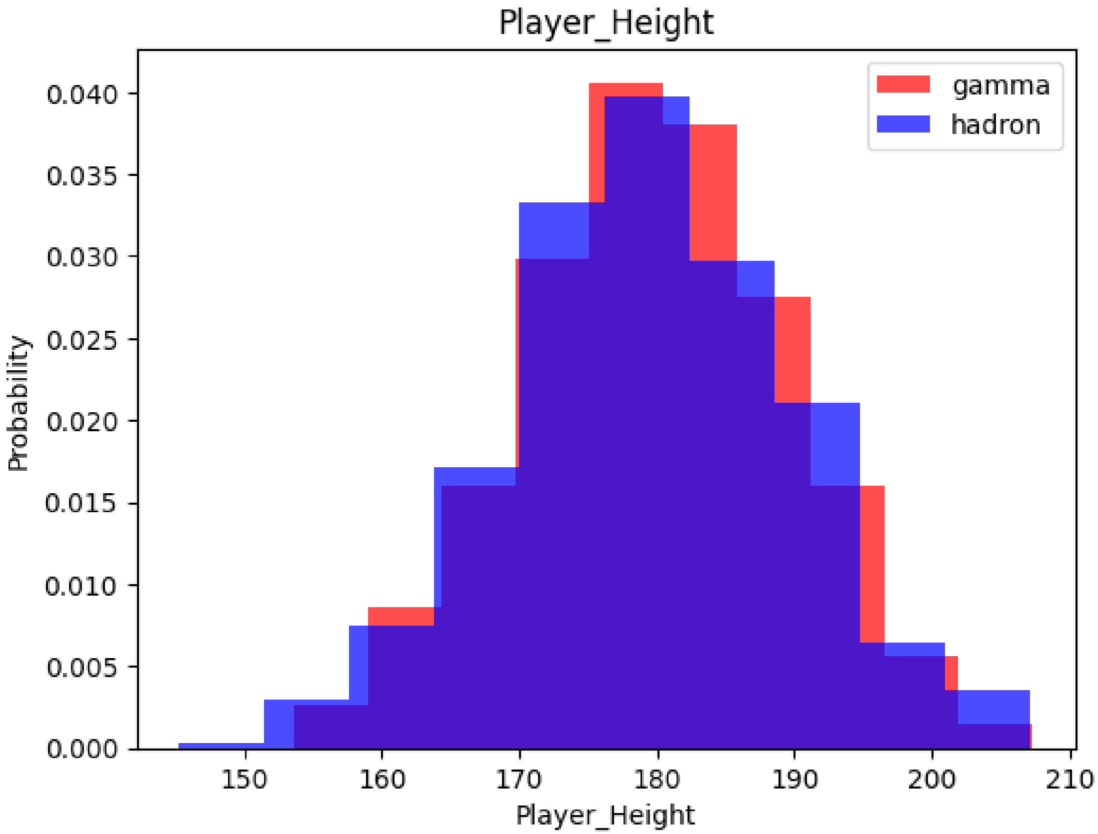
[4]:
```
# We plot a histogram to check which features affect the outcome the most or␣
↪the least
# This helps us determine, which features to use in training our model and the␣
↪ones to discard

for label in header[:-1]:
  plt.hist(df[df['Likelihood_of_Injury'] == 1][label], color = 'red',␣
↪label='gamma', alpha=0.7, density=True)
```

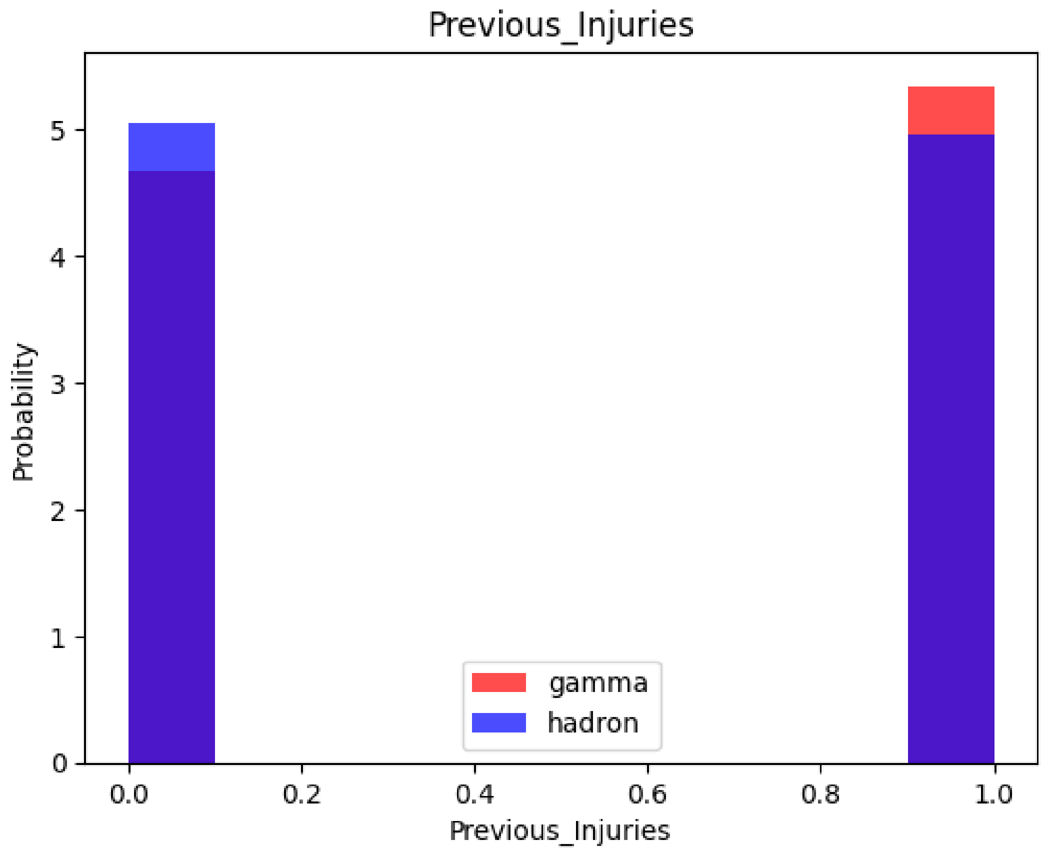
```
plt.hist(df[df['Likelihood_of_Injury'] == 0][label], color = 'blue',␣
↪label='hadron', alpha=0.7, density=True)
plt.title(label)
plt.ylabel('Probability')
plt.xlabel(label)
plt.legend()
plt.show()
```

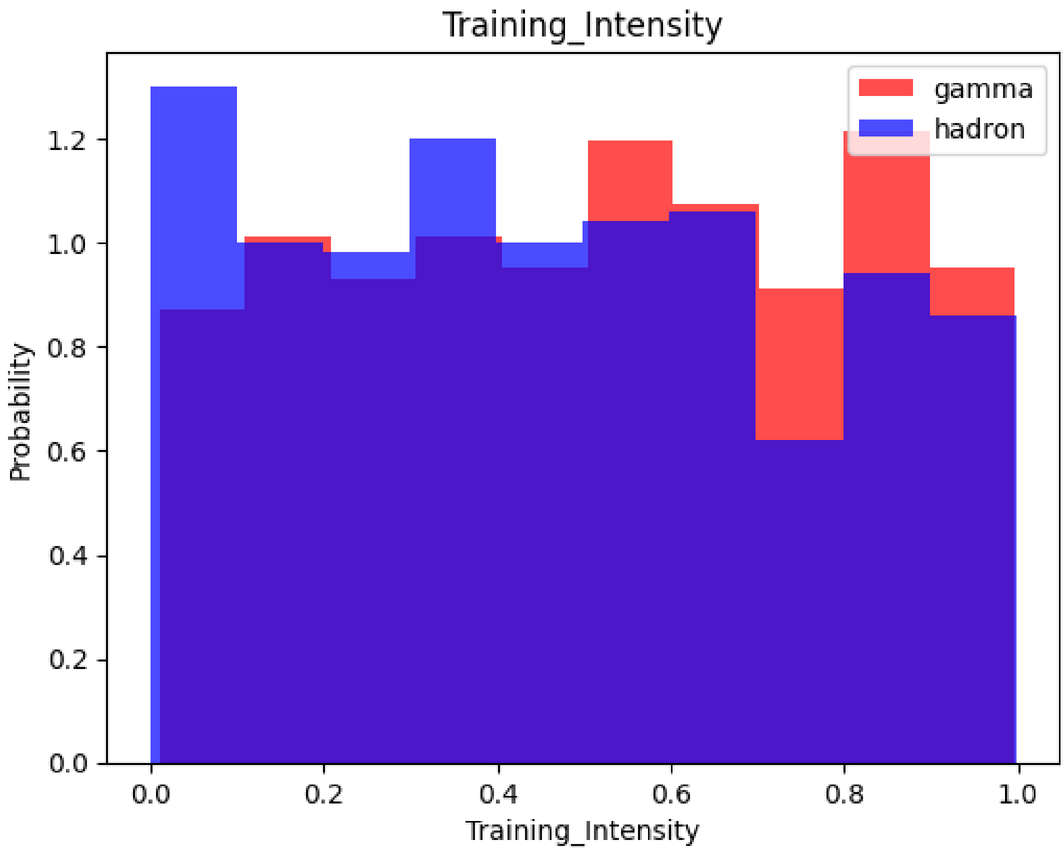


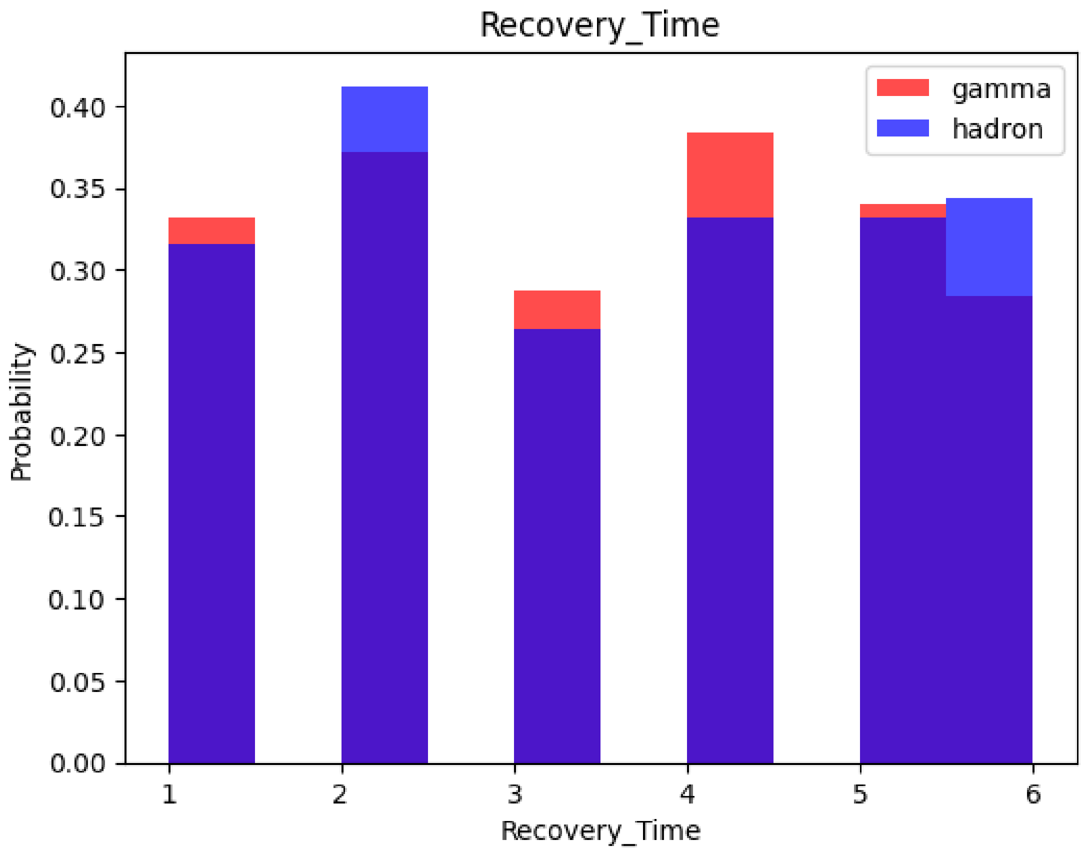


Player_Weight
gamma
hadron
Probability
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0.040
40
50
60
70
80
90
100
Player_Weight


Player_Height
gamma
hadron
Probability
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
150
160
170
180
190
200
210
Player_Height


Previous_Injuries
Probability
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0
Previous_Injuries
gamma
hadron


Training_Intensity
gamma
hadron
Probability
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Training_Intensity



[5]:
```python
train, test = np.split(df.sample(frac=1), [int(0.8 *len(df))])
```

[6]:
```python
# 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
```

[7]:
```
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)
```

[9]:
```
# We'll use Gaussian Naive Bayes implementation from sklearn

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
```

[10]:
```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

[10]: GaussianNB()

[11]:
```
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

[11]: GaussianNB()

[12]:
```
y_pred = nb_model.predict(X_test)
y_pred
```

[12]:

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

[13]:

```
# Check model performance with classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.52      0.52      0.52        97
           1       0.55      0.55      0.55       103

    accuracy                           0.54       200
   macro avg       0.53      0.53      0.53       200
weighted avg       0.53      0.54      0.53       200
```

[ ]: