NBA Players Succes Estimation with Naive Bayes model

May 9, 2025

1 Build a Naive Bayes model

1.1 Introduction

In this activity, you will build your own Naive Bayes model. Naive Bayes models can be valuable to use any time you are doing work with predictions because they give you a way to account for new information. In today's world, where data is constantly evolving, modeling with Naive Bayes can help you adapt quickly and make more accurate predictions about what could occur.

For this activity, you work for a firm that provides insights for management and coaches in the National Basketball Association (NBA), a professional basketball league in North America. The league is interested in retaining players who can last in the high-pressure environment of professional basketball and help the team be successful over time. In the previous activity, you analyzed a subset of data that contained information about the NBA players and their performance records. You conducted feature engineering to determine which features would most effectively predict a player's career duration. You will now use those insights to build a model that predicts whether a player will have an NBA career lasting five years or more.

The data for this activity consists of performance statistics from each player's rookie year. There are 1,341 observations, and each observation in the data represents a different player in the NBA. Your target variable is a Boolean value that indicates whether a given player will last in the league for five years. Since you previously performed feature engineering on this data, it is now ready for modeling.

1.2 Step 1: Imports

1.2.1 Import packages

Begin with your import statements. Of particular note here are `pandas` and from `sklearn`, `naive_bayes`, `model_selection`, and `metrics`.

[1]:
```
# Import relevant libraries and modules.

import pandas as pd
from sklearn import naive_bayes
from sklearn import model_selection
from sklearn import metrics
```

1.2.2 Load the dataset

Recall that in the lab about feature engineering, you outputted features for the NBA player dataset along with the target variable `target_5yrs`. Data was imported as a DataFrame called `extracted_data`. As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the .csv file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

[2]:
```
# RUN THIS CELL TO IMPORT YOUR DATA.
# Load extracted_nba_players_data.csv into a DataFrame called extracted_data.

extracted_data = pd.read_csv('extracted_nba_players_data.csv')
```

1.2.3 Display the data

Review the first 10 rows of data.

[3]:
```
# Display the first 10 rows of data.

### YOUR CODE HERE ###

extracted_data.head(10)
```

[3]:

	fg	3p	ft	reb	ast	stl	blk	tov	target_5yrs	total_points	efficiency
0	34.7	25.0	69.9	4.1	1.9	0.4	0.4	1.3	0	266.4	0.270073
1	29.6	23.5	76.5	2.4	3.7	1.1	0.5	1.6	0	252.0	0.267658
2	42.2	24.4	67.0	2.2	1.0	0.5	0.3	1.0	0	384.8	0.339869
3	42.6	22.6	68.9	1.9	0.8	0.6	0.1	1.0	1	330.6	0.491379
4	52.4	0.0	67.4	2.5	0.3	0.3	0.4	0.8	1	216.0	0.391304
5	42.3	32.5	73.2	0.8	1.8	0.4	0.0	0.7	0	277.5	0.324561
6	43.5	50.0	81.1	2.0	0.6	0.2	0.1	0.7	1	409.2	0.605505
7	41.5	30.0	87.5	1.7	0.2	0.2	0.1	0.7	1	273.6	0.553398
8	39.2	23.3	71.4	0.8	2.3	0.3	0.0	1.1	0	156.0	0.242424
9	38.3	21.4	67.8	1.1	0.3	0.2	0.0	0.7	0	155.4	0.435294

1.3 Step 2: Model preparation

1.3.1 Isolate your target and predictor variables

Separately define the target variable (`target_5yrs`) and the features.

[4]:
```
# Define the y (target) variable.

### YOUR CODE HERE ###
y = extracted_data['target_5yrs']

# Define the X (predictor) variables.

### YOUR CODE HERE ###
X = extracted_data.drop('target_5yrs', axis = 1)
```

Hint 1

Refer to the content about splitting your data into X and y.

Hint 2

In `pandas`, subset your DataFrame by using square brackets `[]` to specify which column(s) to select.

Hint 3

Quickly subset a DataFrame to exclude a particular column by using the `drop()` function and specifying the column to drop.

1.3.2 Display the first 10 rows of your target data

Display the first 10 rows of your target and predictor variables. This will help you get a sense of how the data is structured.

[5]:
```
# Display the first 10 rows of your target data.

### YOUR CODE HERE ###

y.head(10)
```

[5]:
```
0    0
1    0
2    0
3    1
4    1
5    0
6    1
7    1
8    0
9    0
```

```
Name: target_5yrs, dtype: int64
```

Question: What do you observe about the your target variable?

Given that the target variable contains both 1 and 0 indicates that it is binary and requires a model suitable for binary classification.

[6]:
```
# Display the first 10 rows of your predictor variables.

### YOUR CODE HERE ###

X.head(10)
```

[6]:

	fg	3p	ft	reb	ast	stl	blk	tov	total_points	efficiency
0	34.7	25.0	69.9	4.1	1.9	0.4	0.4	1.3	266.4	0.270073
1	29.6	23.5	76.5	2.4	3.7	1.1	0.5	1.6	252.0	0.267658
2	42.2	24.4	67.0	2.2	1.0	0.5	0.3	1.0	384.8	0.339869
3	42.6	22.6	68.9	1.9	0.8	0.6	0.1	1.0	330.6	0.491379
4	52.4	0.0	67.4	2.5	0.3	0.3	0.4	0.8	216.0	0.391304
5	42.3	32.5	73.2	0.8	1.8	0.4	0.0	0.7	277.5	0.324561
6	43.5	50.0	81.1	2.0	0.6	0.2	0.1	0.7	409.2	0.605505
7	41.5	30.0	87.5	1.7	0.2	0.2	0.1	0.7	273.6	0.553398
8	39.2	23.3	71.4	0.8	2.3	0.3	0.0	1.1	156.0	0.242424
9	38.3	21.4	67.8	1.1	0.3	0.2	0.0	0.7	155.4	0.435294

Question: What do you observe about the your predictor variables?

The data indicates that all of the predictor variables are continuous numerical values, so it is important that the model selected is suitable for continuous features.

1.3.3 Perform a split operation on your data

Divide your data into a training set (75% of data) and test set (25% of data). This is an important step in the process, as it allows you to reserve a part of the data that the model has not observed. This tests how well the model generalizes—or performs—on new data.

[7]:
```
# Perform the split operation on your data.
# Assign the outputs as follows: X_train, X_test, y_train, y_test.

### YOUR CODE HERE ###

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,␣
↪test_size=0.25, random_state=0)
```

Hint 1

Refer to the content about splitting your data between a training and test set.

Hint 2

Call the function in the `model_selection` module of `sklearn` on the features and target variable, in order to perform the splitting.

Hint 3

Call the `model_selection.train_test_split()` function, passing in both `features` and `target`, while configuring the appropriate `test_size`.

Assign the output of this split as `X_train`, `X_test`, `y_train`, `y_test`.

1.3.4 Print the shape of each output

Print the shape of each output from your train-test split. This will verify that the split operated as expected.

[8]:
```
# Print the shape (rows, columns) of the output from the train-test split.

# Print the shape of X_train.

### YOUR CODE HERE ###

print(X_train.shape)

# Print the shape of X_test.

### YOUR CODE HERE ###

print(X_test.shape)

# Print the shape of y_train.

### YOUR CODE HERE ###

print(y_train.shape)

# Print the shape of y_test.

### YOUR CODE HERE ###

print(y_test.shape)
```

```
(1005, 10)
(335, 10)
(1005,)
(335,)
```

Hint 1

Call the attribute that DataFrames in `pandas` have to get the number of rows and number of columns as a tuple.

Hint 2

Call the `shape` attribute.

Question: How many rows are in each of the outputs?

Each training DataFrame contains 1,005 rows, while each test DataFrame contains 335 rows. Additionally, there are 10 columns in each X DataFrame, with only one column in each y DataFrame.

Question: What was the effect of the train-test split?

The effect of the train-test split showed an approximately 75% training and 25% test split.

1.4 Step 3: Model building

Question: Which Naive Bayes algorithm should you use?

Using the assumption that your features are normally distributed and continuous, the Gaussian Naive Bayes algorithm is most appropriate for your data. While your data may not perfectly adhere to these assumptions, this model will still yield the most usable and accurate results.

Hint 1

Refer to the content about different implementations of the Naive Bayes to determine which is appropriate in this situation.

Hint 2

Note that you are performing binary classification.

Hint 3

You can identify the appropriate algorithm to use because you are performing a binary classification and assuming that the features of your model follow a normal distribution.

1.4.1 Fit your model to your training data and predict on your test data

By creating your model, you will be drawing on your feature engineering work by training the classifier on the `X_train` DataFrame. You will use this to predict `target_5yrs` from `y_train`.

Start by defining `nb` to be the relevant algorithm from `sklearn.naive_bayes`. Then fit your model to your training data. Use this fitted model to create predictions for your test data.

[9]:
```
# Assign `nb` to be the appropriate implementation of Naive Bayes.

### YOUR CODE HERE ###

nb = naive_bayes.GaussianNB()

# Fit the model on your training data.

### YOUR CODE HERE ###
```

```
nb.fit(X_train, y_train)

# Apply your model to predict on your test data. Call this "y_pred".

### YOUR CODE HERE ###

y_pred = nb.predict(X_test)
```

Hint 1

Refer to the content about constructing a Naive Bayes.

Hint 2

The appropriate implementation in this case is `naive_bayes.GaussianNB()`. Fit this model to your training data and predict on your test data.

Hint 3

Call `fit()`and pass your training feature set and target variable. Then call `predict()` on your test feature set.

1.5 Step 4: Results and evaluation

1.5.1 Leverage metrics to evaluate your model's performance

To evaluate the data yielded from your model, you can leverage a series of metrics and evaluation techniques from scikit-learn by examining the actual observed values in the test set relative to your model's prediction. Specifically, print the accuracy score, precision score, recall score, and f1 score associated with your test data and predicted values.

[10]:
```
# Print your accuracy score.

### YOUR CODE HERE ###

print('accuracy score:'), print(metrics.accuracy_score(y_test, y_pred))

# Print your precision score.

### YOUR CODE HERE ###

print('precision score:'), print(metrics.precision_score(y_test, y_pred))

# Print your recall score.

### YOUR CODE HERE ###

print('recall score:'), print(metrics.recall_score(y_test, y_pred))
```

```
# Print your f1 score.

### YOUR CODE HERE ###

print('f1 score:'), print(metrics.f1_score(y_test, y_pred))
```

```
accuracy score:
0.6895522388059702
precision score:
0.8405797101449275
recall score:
0.5858585858585859
f1 score:
0.6904761904761905
```

[10]: (None, None)

Hint 1

Refer to the content about model evaluation for detail on these metrics.

Hint 2

The `metrics` module in `sklearn` has a function for computing each of these metrics.

Hint 3

Call `accuracy_score()`, `precision_score()`, `recall_score()`, and `f1_score()`, passing `y_test`, and `y_pred` into each function.

Question: What is the accuracy score for your model, and what does this tell you about the success of the model's performance?

The accuracy score for this model is 0.6896, or 69.0% accurate.

Question: Can you evaluate the success of your model by using the accuracy score exclusively?

In classification problems, accuracy is useful to know but may not be the best metric by which to evaluate this model. While accuracy is often the most intuitive metric, it is a poor evaluation metric in some cases. In particular, if you have imbalanced classes, a model could appear accurate but be poor at balancing false positives and false negatives.

Question: What are the precision and recall scores for your model, and what do they mean? Is one of these scores more accurate than the other?

Precision and recall scores are both useful to evaluate the correct predictive capability of a model because they balance the false positives and false negatives inherent in prediction.

The model shows a precision score of 0.8406, suggesting the model is quite good at predicting true positives—meaning the player will play longer than five years—while balancing false positives. The recall score of 0.5859 shows worse performance in predicting true negatives—where the player will not play for five years or more—while balancing false negatives.These two metrics combined can give a better assessment of model performance than accuracy does alone.

Question: What is the F1 score of your model, and what does this score mean?

The F1 score balances the precision and recall performance to give a combined assessment of how well this model delivers predictions. In this case, the F1 score is 0.6905, which suggests reasonable predictive power in this model.

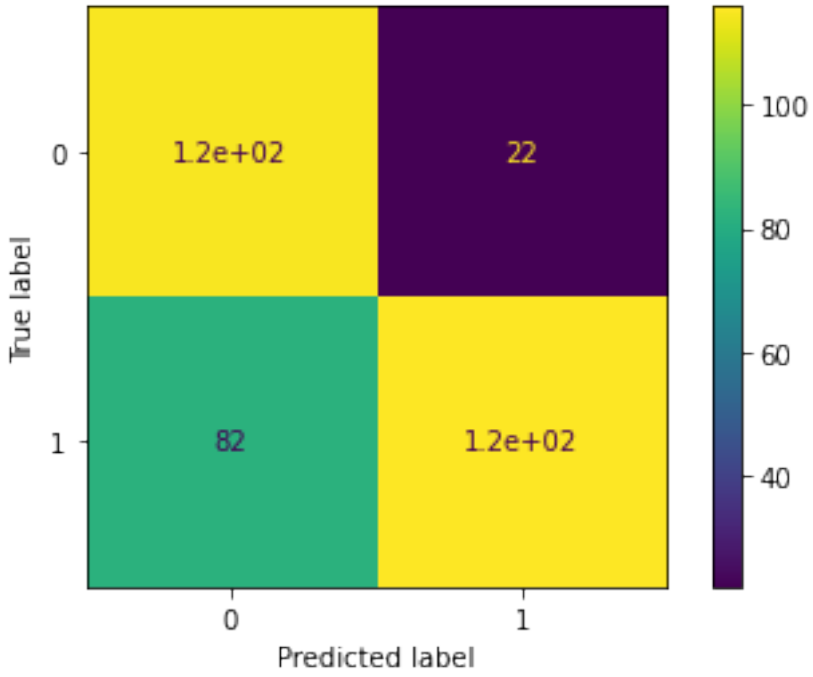
1.5.2 Gain clarity with the confusion matrix

Recall that a confusion matrix is a graphic that shows your model's true and false positives and negatives. It helps to create a visual representation of the components feeding into the metrics.

Create a confusion matrix based on your predicted values for the test set.

[11]:
```
# Construct and display your confusion matrix.

# Construct the confusion matrix for your predicted and test values.

### YOUR CODE HERE ###

cm = metrics.confusion_matrix(y_test, y_pred)

# Create the display for your confusion matrix.

### YOUR CODE HERE ###

disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=nb.
↪classes_)

# Plot the visual in-line.

### YOUR CODE HERE ###

disp.plot()
```

[11]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7fac65475150>
```



Hint 1

The `metrics` module has functions to create a confusion matrix.

Hint 2

Call `confusion_matrix`, passing in `y_test` and `y_pred`. Then, utilize `ConfusionMatrixDisplay()` to display your confusion matrix.

Question: What do you notice when observing your confusion matrix, and does this correlate to any of your other calculations?

- The top left to bottom right diagonal in the confusion matrix represents the correct predictions, and the ratio of these squares showcases the accuracy.
- The concentration of true positives stands out relative to false positives. This ratio is why the precision score is so high (0.8406).
- True negatives and false negatives are closer in number, which explains the worse recall score.

1.6 Considerations

What are some key takeaways that you learned from this lab? - The evaluation of the model is important to inform if the model has delivered accurate predictions. - Splitting the data was important for ensuring that there was new data for the model to test its predictive performance. - Each metric provided an evaluation from a different standpoint, and accuracy alone was not a strong way to evaluate the model. - Effective assessments balance the true/false positives versus true/false negatives through the confusion matrix and F1 score.

How would you present your results to your team? - Showcase the data used to create the prediction and the performance of the model overall. - Review the sample output of the features and the confusion matrix to indicate the model's performance. - Highlight the metric values, emphasizing the F1 score.

How would you summarize your findings to stakeholders? - The model created provides some value in predicting an NBA player's chances of playing for five years or more. - Notably, the model performed better at predicting true positives than it did at predicting true negatives. In other words, it more accurately identified those players who will likely play for more than five years than it did those who likely will not.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged