Bootstrapping

February 21, 2025

1 Bootstrapping

2 Import notebook functions

```
[1]: from notebookfuncs import *
```

2.1 What is Bootstrapping?

Bootstrapping is a resampling technique that helps in estimating the uncertainty of a statistical model.

It includes sampling the original dataset with replacement and generating multiple new datasets of the same size as the original.

Each of these new datasets is then used to calculate the desired statistic, such as the mean or standard deviation.

This process is repeated multiple times, and the resulting values are used to construct a probability distribution for the desired statistic.

This technique is often used in machine learning to estimate the accuracy of a model, validate its performance, and identify areas that need improvement.

2.2 How to Use Bootstrapping in Machine Learning?

Bootstrapping can be used in machine learning in several ways, including the estimation of model performance, model selection, and identifying the most important features in a dataset.

One of the popular use cases of bootstrapping in machine learning is to estimate the accuracy of a classifier, which we will do in this article.

Let's start with an easy example of bootstrapping in Python as a warmup.

2.3 Bootstrapping in Calculating Mean

We will show how bootstrapping can be used to estimate the mean height of students in a school of 10,000 students.

The general approach is to calculate the mean by summing up all heights and dividing the sum by 10,000.

First, generate a sample of 10,000 randomly generated numbers (heights) using the NumPy library.

```
[2]:  import numpy as np
      import pandas as pd
      x = np.random.normal(loc= 185.0, scale=1.0, size=10000)
      printmd("**Here is the output of the mean:**")
      np.mean(x)
```

Here is the output of the mean:

[2]: 185.018910779762

2.4   Machine Learning: What is Bootstrapping?

Now, how can we calculate the height of these students just from 200 students?

In bootstrap sampling, we create several samples of the population by randomly selecting elements from the population with replacements.

In our example, the sample size will be 5. It means there will be 40 samples for the population of 200 students.

With the code below, we will draw 40 samples, size of 5, from the students. (x)

This involves drawing random samples with replacements from the original sample to create 40 new samples.

2.4.1   What do I mean by replacements?

I mean first selecting one of 10,000, forgetting who I chose, then selecting one of 10,000 again.

You can select the same student, but there is a small chance of that. Replacements don't sound that clever anymore, right?

Each new sample has a size of 5, which is smaller than the original sample. The mean of each new sample is calculated and stored in the sample_mean list.

Finally, the mean of all 40 sample means is calculated and represents the estimate of the population mean.

Here is the code.

```
[3]:  import random

      sample_mean = []

      # Bootstrap Sampling
      for i in range(40):
          y = random.sample(x.tolist(), 5)
          avg = np.mean(y)

          sample_mean.append(avg)
      printmd("**Here is the output:**")
      print(np.mean(sample_mean))
```

```
printmd("**Pretty close, isn't it?**")
```

Here is the output:

184.95279460783684

Pretty close, isn't it?

2.5 Bootstrapping in Classification Task With a Decision Tree

Here, we will see the usage of bootstrapping in classification with a decision tree by using the Iris data set. But first, let's see what a decision tree is.

2.6 What Is a Decision Tree?

A decision tree is one of the popular machine learning algorithms that is widely used for both classification and regression problems. It is a tree-based model that involves making predictions by branching out from the root node and making decisions based on certain conditions.

The decision tree classifier is a specific implementation of this algorithm used to perform binary classification tasks.

The main goal of the decision tree classifier is to determine the most important features that contribute to the prediction of a target variable. The algorithm uses a greedy approach to minimize the impurity of the tree by selecting the feature with the highest information gain. The tree continues to split until the data is completely pure or until a stopping criterion is reached.

2.7 What Is the Iris Data Set?

The Iris data set is one of the popular data sets used to evaluate classification tasks.

This data set contains 150 observations of iris flowers, where each observation contains the features such as sepal length, sepal width, petal length, and petal width.

The target variable is the species of the iris plant.

The Iris data set is widely used in machine learning algorithms to evaluate the performance of different models and is also used as an example to demonstrate the concept of decision tree classifiers.

Let's now see how to use bootstrapping and a decision tree in classification.

2.8 Coding of Bootstrapping

The code implements a bootstrapping technique for a machine learning model using the `DecisionTreeClassifier` from the `sci-kit-learn` library.

The first few lines load the Iris dataset and extract the feature data (X) and target data (y) from it.

The bootstrap function takes in the feature data (X), target data (y), and the number of samples (n_samples) to use in bootstrapping.

The function returns a list of trained models and a pandas data frame with precision, recall, F1 score, and the indices used for bootstrapping.

The bootstrapping process is done in a for loop.

For each iteration, the function uses the `np.random.choice` method to randomly select a sample of the feature data (X) and target data (y).

The sample data is then split into training and testing sets using the train_test_split method. The `DecisionTreeClassifier` is trained on the training data and then used to make predictions on the testing data.

The precision, recall, and F1 scores are calculated using the `metrics.precision_score`, `metrics.recall_score`, and `metrics.f1_score` methods from the sci-kit-learn library. These scores are then added to a list for each iteration.

Finally, the results are saved to a pandas data frame with columns for precision, recall, F1 score, the trained models, and the indices used for bootstrapping. The data frame is then returned by the function.

Now, let's see the code.

```python
from sklearn import metrics
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

# Load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target
print(X.shape, y.shape)

def bootstrap(X, y, n_samples=2000):
    models = []
    precision = []
    recall = []
    f1 = []
    indices_x = []
    indices_y = []

    np.random.seed(0)

    for i in range(n_samples):
        index_x = np.random.choice(X.shape[0], size=X.shape[0], replace=True)
        indices_x.append(index_x)
        X_sample = X[index_x, :]

        index_y = np.random.choice(y.shape[0], size=y.shape[0], replace=True)
        indices_y.append(index_y)
        y_sample = y[index_y]

        X_train, X_test, y_train, y_test = train_test_split(
```

```python
            X_sample, y_sample, test_size=0.2, random_state=42
        )

        model = DecisionTreeClassifier().fit(X_train, y_train)
        models.append(model)

        y_pred = model.predict(X_test)
        precision.append(
            metrics.precision_score(y_test, y_pred, average="macro", labels=np.
  ↪unique(y_pred))
        )
        recall.append(metrics.recall_score(y_test, y_pred, average="macro"))
        f1.append(metrics.f1_score(y_test, y_pred, average="macro"))


    # Save the results to a Pandas dataframe
    pred_df = pd.DataFrame(
        {"Precision": precision,
          "Recall": recall,
          "F1": f1,
          "Models": models,
          "Indices_X": indices_x,
          "Indices_Y": indices_y,})
    return models, pred_df

# Now call the function.
models, pred_df = bootstrap(X, y)
# Show the results of the data frame that the function creates.
pred_df.head()

# Now add the index column as the model number,
pred_df['Model Number'] = pred_df.index

# and sort the values by precision.
pred_df.sort_values(by= "Precision", ascending = False).head(5)
```

```
(150, 4) (150,)
```

```
[4]:        Precision     Recall        F1                     Models  \
    244      0.678571   0.688889  0.678644  DecisionTreeClassifier()
    633      0.673611   0.519529  0.503598  DecisionTreeClassifier()
    1626     0.648810   0.690196  0.609604  DecisionTreeClassifier()
    233      0.631373   0.536519  0.460404  DecisionTreeClassifier()
    1490     0.630682   0.511364  0.491135  DecisionTreeClassifier()


                                          Indices_X  \
    244    [96, 98, 47, 55, 26, 110, 125, 131, 66, 94, 74…
```

```
633    [62, 59, 58, 15, 122, 46, 115, 148, 70, 54, 12…
1626   [12, 135, 68, 119, 8, 140, 15, 118, 4, 71, 114…
233    [76, 5, 68, 125, 25, 100, 100, 78, 33, 70, 62,…
1490   [52, 105, 146, 30, 142, 72, 42, 119, 0, 142, 1…


                                          Indices_Y  Model Number
244    [62, 148, 48, 66, 111, 20, 11, 3, 134, 33, 26,…           244
633    [116, 145, 101, 95, 124, 61, 83, 25, 78, 61, 5…           633
1626   [121, 118, 4, 26, 51, 75, 110, 143, 87, 28, 22…          1626
233    [63, 1, 40, 85, 50, 10, 9, 61, 62, 94, 81, 126…           233
1490   [140, 59, 72, 104, 61, 65, 78, 65, 23, 141, 89…          1490
```

To see the results better, we will make some data visualization.

2.9 Bootstrapping Visualization

The code below creates 3 bar graphs to show the performance of bootstrapped models.

The performance of the models is measured by the precision, recall, and F1 scores. So, we will create a data frame called `pred_df`. This data frame stores the scores of the models.

The code creates a figure with 3 subplots.

The first subplot (`ax2`) is a bar graph that shows the precision scores of the top 5 models.

The x-axis shows the precision score, and the y-axis shows the model's index. For each bar, the value is displayed in the center of these 3 figures. The title of this graph is "Precision".

The second subplot (`ax3`) is a bar graph that shows the recall scores of the top 5 models. The x-axis shows the recall score, and the y-axis shows the model's index. The title of this graph is "Recall".

The third subplot (`ax4`) is a bar graph that shows the F1 scores of the top 5 models. The x-axis shows the F1 score, and the y-axis shows the model's index. The title of this graph is "F1".

The overall title of the figure is "Bootstrapped Model Performance Metrics". We will display these images by using `plt.show()`.

Here is the code.

```python
[5]: import matplotlib.pyplot as plt

     # Create a figure and subplots
     fig, (ax2, ax3, ax4) = plt.subplots(1, 3, figsize=(14, 8))

     best_of = pred_df.sort_values(by="Precision", ascending=False).head(5)

     # Create the first graph Precision
     best_of.plot(
         kind="barh",
         x="Model Number",
         y="Precision",
```

```python
        color="mediumturquoise",
        ax=ax2,
        legend=False,
)
ax2.set_xlabel(
        "Precision Score",
        fontstyle="italic",
        fontsize=14,
        fontweight="bold",
        y=1.1,
)
ylabel = "Model\nName"
ax2.set_ylabel(ylabel, fontsize=16)
ax2.set_title("Precision", fontsize=16, fontstyle="italic")
for index, value in enumerate(best_of["Precision"]):
        ax2.text(
                value / 2,
                index,
                str(round(value, 2)),
                ha="center",
                va="center",
                fontsize=14
        )
best_of = pred_df.sort_values(by="Recall", ascending=False).head(5)

# Create the second graph Recall
best_of.plot(
        kind="barh",
        x="Model Number",
        y="Recall",
        color="orange",
        ax=ax3,
        legend=False,
)
ax3.set_xlabel(
        "Recall Score",
        fontstyle="italic",
        fontsize=14,
        fontweight="bold",
)
ax3.set_ylabel(ylabel, fontsize=16)
ax3.set_title("Recall", fontsize=16, fontstyle="italic")
for index, value in enumerate(best_of["Recall"]):
        ax3.text(
                value / 2,
                index,
                str(round(value, 2)),
```

```python
        ha="center",
        va="center",
        fontsize=14
    )

# Create the third graph F1
best_of = pred_df.sort_values(by="F1", ascending=False).head(5)
best_of.plot(
    kind="barh",
    x="Model Number",
    y="F1",
    color="darkorange",
    ax=ax4,
    legend=False,
)
ax4.set_xlabel(
    "F1 Score",
    fontstyle="italic",
    fontsize=14,
    fontweight="bold",
)
ax4.set_ylabel(ylabel, fontsize=16)
ax4.set_title("F1", fontsize=16, fontstyle="italic")

for index, value in enumerate(best_of["F1"]):
    ax4.text(
        value / 2,
        index,
        str(round(value, 2)),
        ha="center",
        va="center",
        fontsize=14
    )
# Fit the figure
plt.tight_layout()

plt.suptitle(
    "Bootstrapped Model Performance Metrics",
    fontsize=18,
    y=1.05,
    fontweight="bold"
)

# Show the figure
plt.show()
```
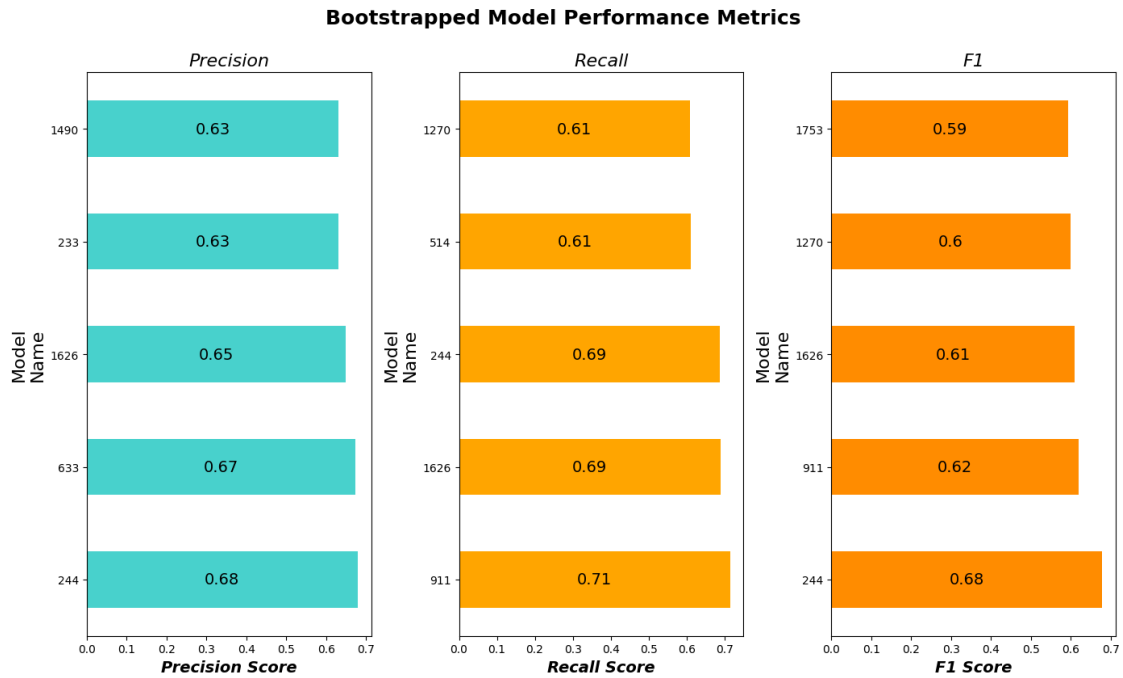
Bootstrapped Model Performance Metrics

Source: https://www.kdnuggets.com/2023/03/bootstrapping.html

`[6]:`
```
allDone();
```

<IPython.lib.display.Audio object>