Machine Learning and Deep Learning

Prediction algorithm using naive Bayes approach

Geovani Teca
Cracow 202

1. Introduction

Machine learning is the concept that a computer program can learn from data and produce output without human intervention. Deep learning is a subset of machine learning which is a concept that a machine through an algorithm uses data to train itself to perform certain tasks, such as predict an outcome, image or voice recognition. Machine learning is divided into three main fields: Supervised learning, unsupervised learning and reinforcement learning. The last two are out of the scope of this project, the focus of the project is supervised learning which is the process that consists of teaching a machine to predict an outcome based on known previous outcomes provided as input to train the model. The supervised learning is also divided into two fields: Regression which expects the model to predict a numerical value and classification which predicts the class where the outcome very probably belongs to. The entire picture is presented in figure 1.

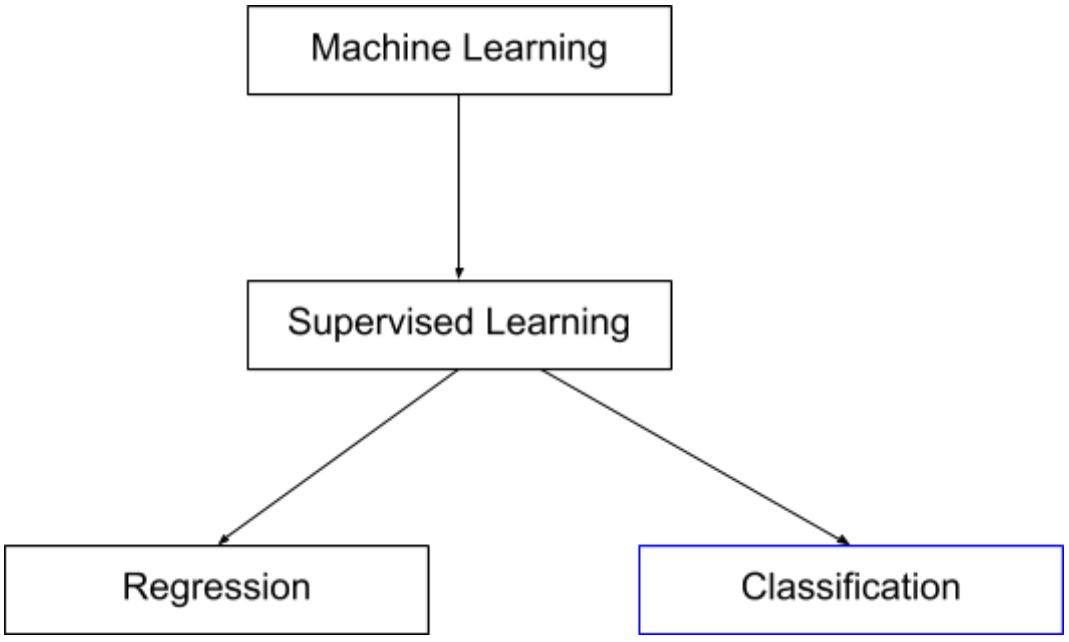

Figure 1. Classification model under the machine learning hierarchy

The goal of the project is to present an implementation of a machine learning, supervised learning using classification model to predict the outcome of events based on naive Bayes probability.

2. Classification Model With Naive Bayes Algorithm

Naive Bayes is a classification technique with an assumption of independence among predictors, it is assumed that the probability of the occurrence event Y is independent from the event X that already has occurred.

Applying in the machine learning field the algorithm is used to predict to which class a set of data belongs to, based on the input training data already provided to the machine. The machine is provided input data and classes (outcomes) to train the model, and after the training is provided with a set of new data the machine predicts the class that the input data belongs to. The figure 2 illustrates the classification model using naive Bayes algorithms.

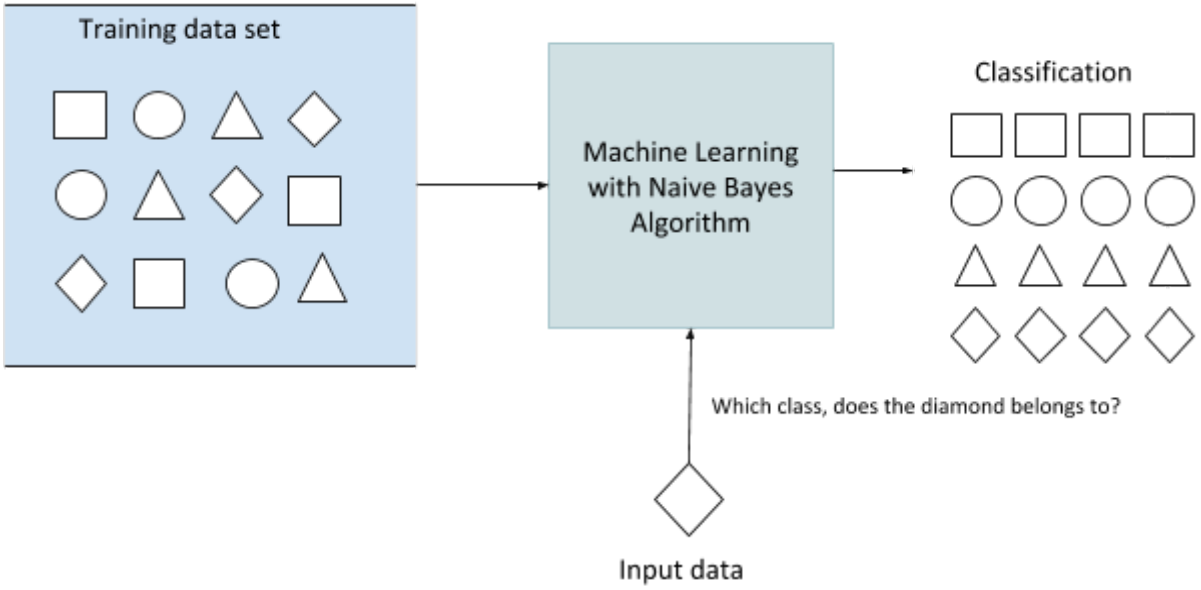

Figure 2. Naive Bayes classification

The mathematics of Naive Bayes algorithm computed for the project is as follows.

$$P(class|data) = \frac{P(data|class) * P(class)}{P(data)} \quad (1)$$

Where:

$P(class|data)$ − $Posteria$: $The\ probability\ of\ a\ data\ belong\ to\ a\ class$

$P(data|class)$ − $Class\ conditional\ probability$: $The\ known\ outcomes$

$P(class)$ − $Prior$: $The\ prior\ belief,\ that\ the\ outcome\ has\ occurred$

$P(data)$ − $Prior\ probability\ of\ data,\ which\ is\ not\ calculate\ in\ Naive\ Bayes$

For a model with different classes (outcomes), the equation (1) is rewritten as:

$$P(class|data) = \frac{P(data|class_1) * P(data|class_2) * \ldots * P(data|class_n) * P(class)}{P(data)} \quad (2)$$

Since $P(data)$ is not calculated in Naive Bayes, assumed to not have direct impact in the calculation, the equation is reduce to predict the class with the maximum probability to:

$$P(class) = argmax_{class}\, P(data|class_1) * P(data|class_2) * \ldots * P(data|class_n) \quad (3)$$

The $P(data|class)$ is a normal distribution and is calculated using Gaussian distribution probability:

$$P(data|class) = \frac{1}{\sigma_{data} * \sqrt{2*\pi}} * exp(-\frac{(data - \mu_{data})^2}{2*\sigma_{data}^2}) \quad (4)$$

3. Project description

The Naive Bayes algorithm for classification is implemented in Python 3 programming language, using the following libraries:

- math
- csv
- random

To the project is attached the files:

- **main.py**: The file containing the naive bayes algorithm classification and accuracy calculator.
- **The data set**: The files which contain the source data to train the model, the files to train the model were csv files generated from https://www.mockaroo.com.

The advantage of the script is such data is not limited only to the experimented data set, the code is able to work with any data set provided in the csv file.

In order to implement the Naive Bayes in Python the following steps are needed:

1. Load the data set in a csv format file

   The data set is stored in csv (comma separated values) format, where data is separated by comma sign to represent column. The file is opened, read and each row is stored as element of the list object, where the index represents the row and the column values are stored under the given index.

```
7   # Function Purpose: Open the CSV data file
8   # @input: file name
9   # @output: The list with all the row of dataset
10  def load_csv(filename):
11      dataset = list()
12      with open(filename, 'r') as file:
13          csv_reader = reader(file)
14          for row in csv_reader:
15              if not row:
16                  continue  # skip empty rows with no data
17              dataset.append(row)
18      return dataset
```

2. Classify the data set by class, this step is selecting the classes that represent all the possible outcomes from the data set.

   The data set along with the classes are separated in a dictionary data type where the classes (the last index of dataset list object) are the key and all the features in the row are stored in a list object to become the value of the dictionary.

```
59  def split_dataset_by_class(dataset):
60      class_and_features_dict = dict()
61      for i in range(len(dataset)):
62          current_row = dataset[i]
63          class_value = current_row[-1]
64          if (class_value not in class_and_features_dict):
65              class_and_features_dict[class_value] = list()
66          class_and_features_dict[class_value].append(current_row)
67      return class_and_features_dict
```

3. Calculate the statistical parameter of data set (mean and standard deviation of data set)

   The mean value of the dataset is calculated as the sum of all the elements divided by the number of elements in the dataset.

```
72  def calculate_mean_value(features_values):
73      num_of_samples = float(len(features_values))
74      class_mean_value = sum(features_values) / num_of_samples
75      return class_mean_value
```

The standard deviation is calculated as the square root of variance of the data set.

```
80  def calculate_stdev(features_values):
81      average_value = calculate_mean_value(features_values)
82      num_of_samples = float(len(features_values)-1)
83      variance = sum([(x-average_value)**2 for x in features_values]) / num_of_samples
84      return sqrt(variance)
```

4. Calculate the probability function using Gaussian distribution

   In this step the equation 4 is computed for each class, the probability of each class is calculated taking into account the mean, standard deviation of the underline class.

```
97  def calculate_probability(features_values, mean, stdev):
98      exponent = exp(-((features_values - mean)**2 / (2 * stdev**2 )))
99      posteria = (1 / (sqrt(2 * pi) * stdev)) * exponent
100     return posteria
```

5. Calculate the classes probability (text is placed due to the low quality of screenshot)

   Based on the input data after training the model, the input data is evaluated and then is calculated the probability of all classes in the model, based on the input data (the feature values) provided as parameter row.

```
def calculate_class_probability(aggregated_dataset_by_class, row):
  class_row = 0
  dataset_len_index = 2
  total_rows = sum([aggregated_dataset_by_class[class_name][class_row][dataset_len_index]
          for class_name in aggregated_dataset_by_class])
  probabilities = dict()
  for class_value, class_aggregated_dataset_by_class in aggregated_dataset_by_class.items():
    probabilities[class_value] =
aggregated_dataset_by_class[class_value][class_row][dataset_len_index] / (total_rows)
    for i in range(len(class_aggregated_dataset_by_class)):
      mean, stdev, count = class_aggregated_dataset_by_class[i]
      probabilities[class_value] *= calculate_probability(row[i], mean, stdev)
  return probabilities
```

6. Predict the result

The function calculates from the returned calculated probability of all the class, which class has the highest value of probability, meaning that based on the input data the model can then classify to which class the input data belongs to.

```
132  def predict_outcome(training_data, test_data):
133      probabilities = calculate_class_probability(training_data, test_data)
134      best_class, best_prob = None, -1
135      for class_value, probability in probabilities.items():
136          if best_class is None or probability > best_prob:
137              best_prob = probability
138              best_class = class_value
139      return best_class
```

7. Calculate the algorithm accuracy

   The algorithm accuracy is calculated based on the comparison between how many outcomes the model predicts correctly compared to the known outcome.

```
154  def get_accuracy(test_data, prediction):
155      correct = 0
156      for i in range(len(test_data)):
157          if test_data[i][-1] == prediction[i]: # compare prediction with the column outcome
158              correct = correct + 1
159
160      accuracy = (correct/float(len(test_data)))*100.0
161      return accuracy
```

4. Simulation results

After writing the Naive Bayes algorithm to training the mode, two different dataset are selected and attached to the project files:

- **players_stats.csv**: Football Players stats, to allow prediction in which classification in a season a player would end up, the input data is represented with the following features: *[Games Played, overall Assists, Goals scored, Trophies won].*

- **purschase_gender.csv**: The data about the purchase of items and the amount of money spent, to allow prediction of gender based on the quantity of items bought which gender made the purchase. the input data is represented with the following features: *[Amount Spent, Hats, Jeans, T-shirts, Dress, Shoes, Jewellery, Gender].*

In order to test the different data set, the filename and the input data variable have to be changed to proper values.

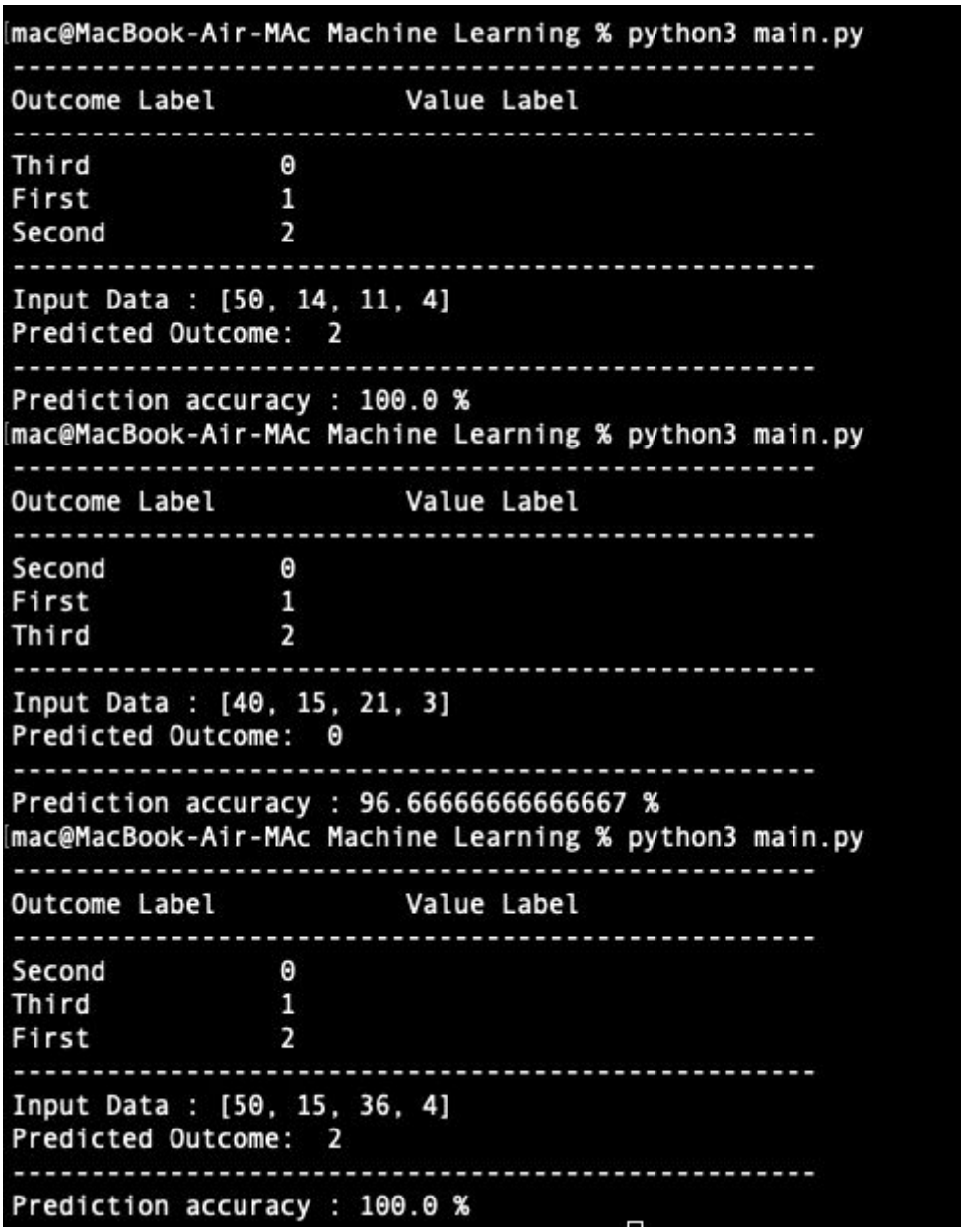

Figure 3. Prediction for the best player winner based on the stats over the season

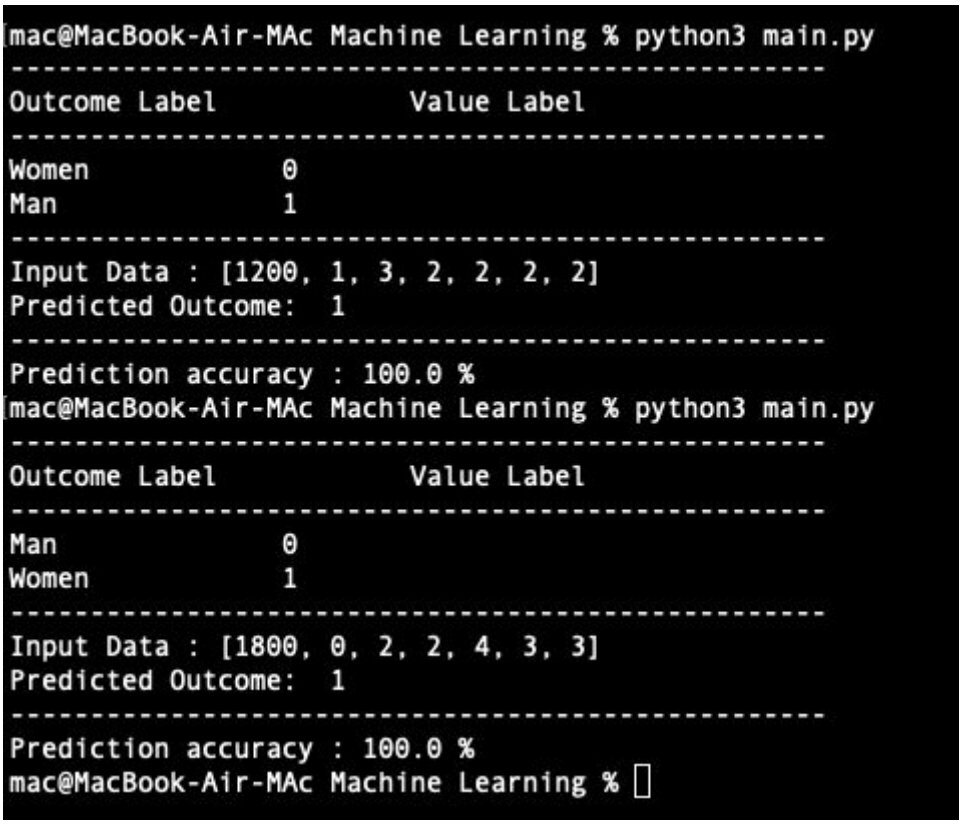

Figure 4. Gender prediction based on the amount spent and quantity of item bought by a client

5. Conclusion

Naive Bayes algorithm for model a machine learning concept is a simple but very useful process that is widely applied in different fields of our lives, and the outcomes from prediction prove to be accurate in most of situations, leading to measures that improve revenue for business, prevent diseases for health care, impact on decision in stock market in finances and for sure finds the best application in the remaining fields, with a downside that the Naive Bayes assume that events are independent from each other which in real life might not necessary is proved to be true in reality.