# MACHINE LEARNING ASSIGNMENT-2

Nikhil Dhanaraj 2021A7PS0427H

PS Varshith 2021A8PS1660H

Sricharan Reddy Bollampalli 2021A7PS0379H

## PART A - Predicting Whether Income of an Individual is Greater Than or Less Than $50K Using Naïve Bayes Classifier Machine Learning Model.

### Concept Behind The Model

Naïve Bayes is a classification algorithm based on Bayes' theorem which

describes the probability of an event based on prior knowledge of conditions

that might be related to the event.

Naïve Bayes is called "naïve" because it assumes that all

attributes are independent of each other, which is usually not true in

real-world data. However, this simplification enables the algorithm to work

efficiently and accurately in many practical situations.

The algorithm is called "Bayes" because

it involves calculating the probabilities of class membership based on Bayes'

theorem, which is:

**BAYES THEOREM**

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

where:

- P(A|B) is the posterior probability of A given B
- P(B|A) is the likelihood probability of B given A
- P(A) is the prior probability of A
- P(B) is the prior probability of B

In the context of classification, B represents the input data (i.e., the attributes of a sample), and A represents the class label. The algorithm computes the posterior probabilities of each class label given the input data and selects the class label with the highest probability as the predicted class.

To compute the posterior probabilities, Naive Bayes assumes that the likelihood probability of B given A can be estimated by the product of the probabilities of each attribute given the class label. This assumption is called the "naïve" assumption because it assumes independence among attributes.

In practice, Naïve Bayes can be applied to both binary and multi-class classification problems, and it is particularly suitable for high-dimensional datasets with many attributes. It has been shown to perform well in many real-world applications, such as email spam filtering, sentiment analysis, and medical diagnosis.

## Implementing The Model

The Adult dataset contains demographic information about individuals, such as age, education, occupation, and income. The goal is to predict whether an individual's income is above or below $50k per year.

The code starts by importing necessary libraries such as numpy and pandas.

A class NaiveBayes is defined, which has methods to calculate prior probabilities and conditional probabilities, fit the model on training data, predict the class of new data, and calculate accuracy, precision, and recall.

The calculate_prior method calculates the prior probability of each class by dividing the number of instances of that class by the total number of instances.

The calculate_conditional_probabilities method calculates the conditional probabilities of each feature given each class using the Laplace smoothing technique to avoid zero probabilities.

The predict_class method predicts the class of a new instance by multiplying the prior probability with the conditional probability for each feature.

The fit method fits the model on training data by calculating the prior and conditional probabilities.

The predict method predicts the class of new instances using the predict_class method and returns the predictions.

The accuracy, precision, and recall methods calculate the corresponding metrics using the true labels and predicted labels.

Lastly, the code reads the Adult dataset and splits it into training and testing sets. It then trains the Naive Bayes classifier on the training set and tests it on the testing set. It calculates the accuracy, precision, recall, and F1 score of the classifier and prints them out. The F1 score is the harmonic mean of precision and recall, and it is a measure of the classifier's accuracy that takes into account both false positives and false negatives.

## Observations From The Model

Without Applying Any Smoothening Techniques:-

| Seed | Accuracy | Precision | Recall | F1_Score |
|---|---|---|---|---|
| 44 | 82.81 | 50.04 | 63.93 | 56.14 |
| 454 | 82.72 | 46.17 | 66.91 | 56.64 |
| 4454 | 77.19 | 72.24 | 30.23 | 42.62 |
| 334 | 83.62 | 53.57 | 63.4 | 58.08 |
| 94 | 83.2 | 44.76 | 69.05 | 54.32 |
| 954 | 76.97 | 72.47 | 30.15 | 42.58 |
| 9 | 82.95 | 52.83 | 62.45 | 57.24 |
| 49 | 77.19 | 73.3 | 29.95 | 42.52 |
| 499 | 77.08 | 71.52 | 30.88 | 43.14 |
| 899 | 77.33 | 73.63 | 29.68 | 42.4 |
| 699 | 77.07 | 72.92 | 30.76 | 42.26 |
| 7669 | 77.47 | 73.43 | 30.9 | 43.5 |

|  |  |  |  |  |
|---|---|---|---|---|
| 569 | 84.15 | 50.85 | 66.3 | 57.54 |
| 5 | 83.05 | 47.99 | 66.51 | 55.54 |
| 12 | 76.98 | 73.83 | 29.85 | 42.5 |

The average values for accuracy, precision , recall and F1 score are

|  |
|---|
| Avg accuracy = 79.98 |

|  |
|---|
| Avg precision = 63.236 |

|  |
|---|
| Avg recall = 42.46 |

|  |
|---|
| Avg F1 score = 45.688 |

Applying Additive Smoothening Techniques:-

| Seed | Accuracy | Precision | Recall | F1_Score |
|---|---|---|---|---|
| 36 | 82.24 | 41.87 | 68.93 | 52.1 |
| 96 | 77.08 | 73.07 | 30.02 | 42.56 |
| 1435 | 76.68 | 73.21 | 29.67 | 42.22 |
| 9678 | 82.58 | 42.75 | 68.47 | 52.64 |
| 219 | 76.97 | 71.82 | 30.8 | 43.1 |
| 439 | 77.38 | 73.01 | 30.99 | 43.5 |
| 569 | 77.83 | 75.39 | 30.11 | 43.04 |
| 9988 | 77.27 | 72.52 | 30.84 | 43.28 |

|  |  |  |  |  |
|---|---|---|---|---|
| 456 | 82.57 | 53.73 | 60.74 | 57.02 |
| 879 | 82.35 | 44.14 | 68.46 | 40.84 |
| 659 | 76.72 | 73.77 | 28.23 | 40.84 |
| 876 | 80.47 | 34.55 | 72.63 | 46.81 |
| 654 | 77.15 | 73.04 | 29.63 | 42.22 |
| 3478 | 76.91 | 72.17 | 28.79 | 42.16 |
| 9878 | 76.64 | 73.5 | 28.59 | 41.16 |

The average values for accuracy, precision , recall and F1 score are

Avg accuracy = 81.024

Avg precision = 66.277

Avg recall = 40.743

Avg F1 score = 47.032

## Inference From The Model

Based on the data provided, there is a difference in the average values of accuracy, precision, recall, and F1 score between the two data sets (with and without additive techniques). The average values of accuracy and precision appear to have decreased slightly with the addition of the method, while the average values of recall and F1 score appear to have reduced more significantly

## Comparison with Logistic Regression

The average accuracy(79%) of LR in comparison to Naïve Bayes classifier was the same. But the difference between the two models is shown in the values of precision(79%), recall(100%) and f1 score(88%). These 3 values were considerably greater than that of the Naïve Bayes classifier.

## Comparison with K-Nearest Neighbours

The average accuracy(79%) of KNN in comparison to Naïve Bayes classifier gave similar results although Naïve classifier with smoothening gave better results. KNN gave an average precision of 60.07% and an average recall of 77.28% which were considerably higher than that of the Naïve Bayes classifier.

# PART B - Recognizing Handwritten Digits Using Artificial Neural Network Machine Learning Model.

## Concept Behind The Model

ANN stands for Artificial Neural Network, a machine learning algorithm.

ANNs are composed of layers of artificial neurons connected by weighted edges. Each neuron receives input from its connected neurons, performs a mathematical operation on the input, and then passes the result to its interconnected neurons in the next layer. This process is repeated until the output layer produces a result.

Each neuron receives input from other neurons in the previous layer, processes that input, and produces an output that is transmitted to the next layer. The connections between neurons are modelled by weights, determining the strength of the transmitted signal.

Training an ANN involves adjusting the weights between neurons based on the input data and desired output. Training is typically done using an algorithm called backpropagation, which calculates the error between the predicted output and the desired output and adjusts the weights to minimize that error.

## Implementing The Model

Here we are using ann to classify handwritten digits, and we do so using tensorflow.

We split the data into training and testing data and then created the ANN architecture using TensorFlow. You can use the tf.keras.Sequential API to create the layers of the ANN.

Using tensorflow, we Train the ANN using the training data set. During training, the weights of the ANN are adjusted to minimize the error between the predicted output and the actual output. You can use the compile() and fit() functions in TensorFlow to train the model.

We then Validate the trained ANN using the testing data set. The accuracy of the ANN on the testing data set should be calculated using the evaluate() function.

We can then change the number of layers and other parameters to get the 15 required models for the assignment.

We then validated the models using testing data and wrote a function for accuracy to use as a performance metric.

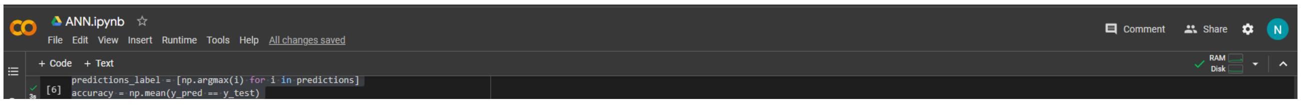
## Observations From The Model

| number | model | accuracy |
|---|---|---|
| 1 | two hidden layers | 94.78% |
| 2 | three hidden layers | 95.89% |
| 3 | three hidden layers with 100 neurons | 95.70% |
| 4 | three hidden layers with 150 neurons | 95.43% |
| 5 | two hidden layers with relu and tanh | 94.20% |
| 6 | two hidden layers with sigmoid and relu | 91.02% |
| 7 | two hidden layers with sigmoid and tanh | 91.39% |

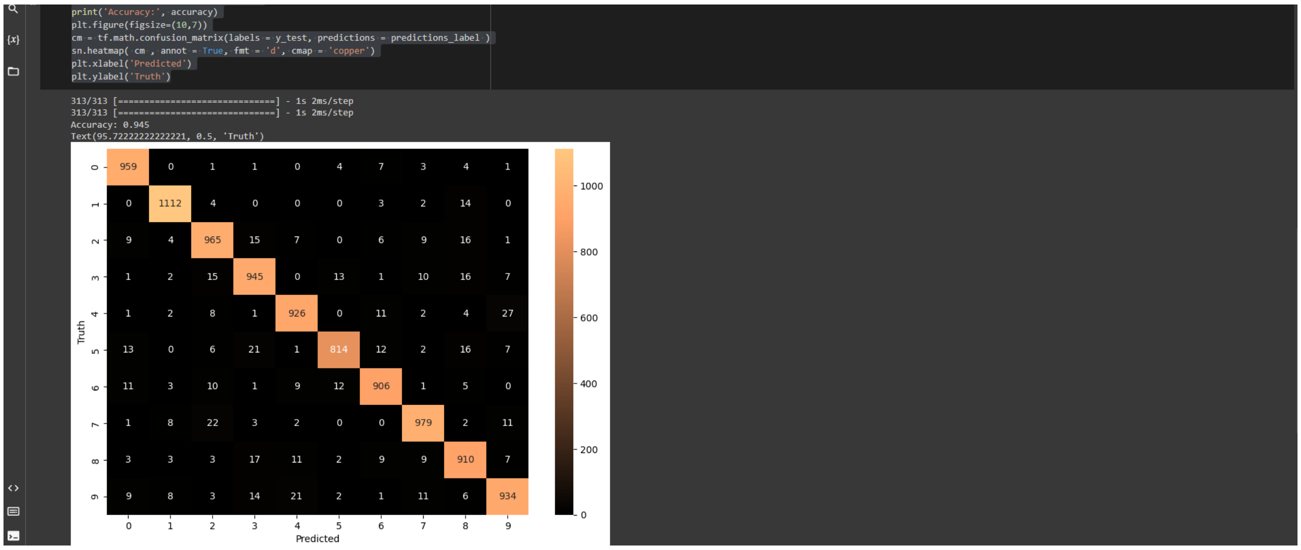
| | | |
|---|---|---|
| 8 | three hidden layers with relu, sigmoid and tanh | 91.98% |
| 9 | three hidden layers with relu, sigmoid and tanh with 100 neurons | 92.04% |
| 10 | three hidden layers with only relu and 150 neurons | 95.54% |
| 11 | three hidden layers with only tanh and 150 neurons | 93.48% |
| 12 | three hidden layers with only relu and 100 neurons | 95.52% |
| 13 | three hidden layers with only tanh and 100 neurons | 93.97% |
| 14 | two hidden layers with relu and tanh and 100 neurons | 94.65% |
| 15 | two hidden layers with sigmoid and tanh with 100 neurons | 91.34% |

## Confusion Matrix

This is the confusion matrix for the ANN model which had two hidden layers with 100 neurons and relu as the activation function for both of them.



```
[6] predictions_label = [np.argmax(i) for i in predictions]
    accuracy = np.mean(y_pred == y_test)
```



```
print('Accuracy:', accuracy)
plt.figure(figsize=(10,7))
cm = tf.math.confusion_matrix(labels = y_test, predictions = predictions_label )
sn.heatmap( cm , annot = True, fmt = 'd', cmap = 'copper')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
Accuracy: 0.945
Text(95.72222222222221, 0.5, 'Truth')
```

| Truth \ Predicted | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 959 | 0 | 1 | 1 | 0 | 4 | 7 | 3 | 4 | 1 |
| 1 | 0 | 1112 | 4 | 0 | 0 | 0 | 3 | 2 | 14 | 0 |
| 2 | 9 | 4 | 965 | 15 | 7 | 0 | 6 | 9 | 16 | 1 |
| 3 | 1 | 2 | 15 | 945 | 0 | 13 | 1 | 10 | 16 | 7 |
| 4 | 1 | 2 | 8 | 1 | 926 | 0 | 11 | 2 | 4 | 27 |
| 5 | 13 | 0 | 6 | 21 | 1 | 814 | 12 | 2 | 16 | 7 |
| 6 | 11 | 3 | 10 | 1 | 9 | 12 | 906 | 1 | 5 | 0 |
| 7 | 1 | 8 | 22 | 3 | 2 | 0 | 0 | 979 | 2 | 11 |
| 8 | 3 | 3 | 3 | 17 | 11 | 2 | 9 | 9 | 910 | 7 |
| 9 | 9 | 8 | 3 | 14 | 21 | 2 | 1 | 11 | 6 | 934 |

## Inference From The Model

As we can see relu as the activation function gave the best results.

Three hidden layers gave better results than two hidden layers and 100 neurons gave better classification results.

Tanh gave better results than sigmoid activation layer.

We used the three hidden layers with 100 neurons with relu as the activation function model to obtain the best results.