

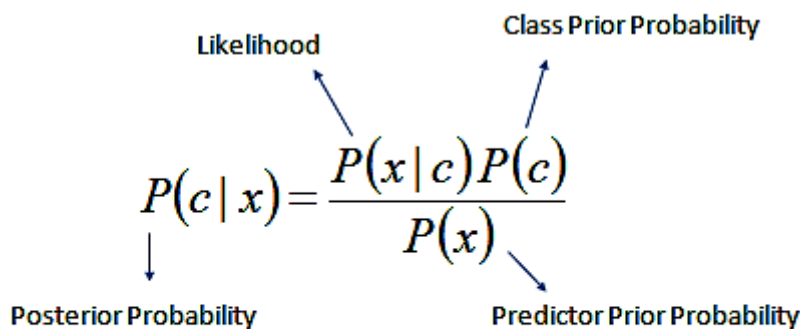
# Machine Learning Assignment - 2 report

By:

Udit Gupta (ID:2020B4A72368H) Aditya R Goyal(ID:2019B2A81443H) Nimish Agrawal (ID:2020B3A71857H) ]

## Naive Bayes

Naive Bayes is a simple probabilistic algorithm that is used for classification tasks in machine learning. It works on the principle of Bayes' theorem, which states that the probability of a hypothesis given the observed evidence is proportional to the probability of the evidence given the hypothesis. In other words, it calculates the probability of each class given the feature values of an instance and then selects the class with the highest probability as the predicted class. Naive Bayes is called "naive" because it assumes that all features are independent of each other, which is often not true in real-world problems.



The diagram shows the formula for the posterior probability in Naive Bayes classification. The formula is  $P(c | x) = \frac{P(x | c)P(c)}{P(x)}$ . Arrows point from labels to parts of the formula: 'Likelihood' points to  $P(x | c)$ , 'Class Prior Probability' points to  $P(c)$ , 'Posterior Probability' points to  $P(c | x)$ , and 'Predictor Prior Probability' points to  $P(x)$ .

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

In this implementation, we have used the Naive Bayes algorithm to classify the income of adults based on various demographic and socio-economic factors such as age, education level, occupation, etc. The dataset used in this implementation is the Adult Dataset, which is publicly available on the UCI Machine Learning Repository.

### Steps

1. Remove missing values and preprocess dates by dropping rows with missing data represented as "?"
2. Convert categorical variables to integer type for easy analysis, including mapping the income target variable to 1 if income > 50k and 0 otherwise.
3. Split the dataset into train and test sets with a 67% - 33% split.
4. Implement a Gaussian Naive-Bayes classifier, including calculating the prior probability of each class, conditional probability of each feature given each class, and predicting the class of a given instance.
5. Evaluate and compare the classifier's performance with KNN and logistic regression by calculating recall, precision, accuracy, and F1 score based on 50 different training and testing splits.

In the given dataset, the target variable represents whether an individual makes less than or equal to \$50K per year or more than \$50K per year.

**Prior probability** refers to the probability of an instance belonging to a certain class before considering any features. In the context of this dataset, prior probability would be the probability of an individual making less than or equal to \$50K per year or more than \$50K per year, without considering any of their features. Prior probability is calculated as the number of instances belonging to each class divided by the total number of instances in the training set.

**Conditional probability**, on the other hand, is the probability of a certain feature value given a class. In the context of this dataset, conditional probability would be the probability of a certain feature value (such as age or education) given that an individual belongs to a certain income class (less than or equal to \$50K per year or more than \$50K per year). These conditional probabilities are calculated for each feature for each class in the training set.

The Naive Bayes algorithm uses these prior and conditional probabilities to predict the class of new instances. The algorithm assumes that each feature is conditionally independent given the class. Given a new instance with certain feature values, the algorithm calculates the probability of the instance belonging to each class using the prior and conditional probabilities and predicts the class with the highest probability.

The prior probabilities of each class are calculated, and the conditional probabilities of each feature given each class are estimated. A smoothing factor is added to prevent the probabilities from being zero.

A loglikelihood function is defined to calculate the log-likelihood of an instance belonging to each class, and a predict function is defined to predict the class of an instance based on the maximum log-likelihood.

The code then makes predictions on the test set and calculates the accuracy. The predicted labels are converted to floats, and the confusion matrix is calculated. The true positives, true negatives, false positives, and false negatives are then computed from the confusion matrix. The precision, recall, and F1-score are calculated from these values.

#### **confusion matrix:**

**Accuracy:** The percentage of instances that are correctly predicted. In this case, the accuracy is  $(TP + TN) / (TP + TN + FP + FN)$

**Precision:** The percentage of instances that are predicted as positive that are actually positive. In this case, the precision is  $TP / (TP + FP)$

**Recall** (also known as Sensitivity): The percentage of positive instances that are correctly predicted as positive. In this case, the recall is  $TP / (TP + FN)$

**F1 Score:** A weighted average of precision and recall, where a score of 1.0 represents perfect precision and recall, and 0.0 represents the worst possible score. In this case, the F1 score is  $2 * (precision * recall) / (precision + recall)$

#### **Results:**

Naive Bayes without smoothing:

Mean Accuracy of 10 Random splits	0.80482
Variance of accuracy of 10 random splits	8.347x 10 <sup>-6</sup>

Mean Precision of 10 random splits	0.6824
Variance precision of 10 random splits	$8.307 \times 10^{-5}$
Mean Recall of 10 random splits	0.39439
Variance Recall of 10 random splits	$5.87031 \times 10^{-5}$
Mean F1 Score of 10 random splits	0.49984
Variance F1 score of 10 random splits	$5.362271 \times 10^{-5}$

Naive Bayesian with smoothing:

Mean Accuracy of 10 Random splits	0.80452
Variance of accuracy of 10 random splits	$8.8235 \times 10^{-6}$
Mean Precision of 10 random splits	0.67848
Variance precision of 10 random splits	$7.912477 \times 10^{-5}$
Mean Recall of 10 random splits	0.39606
Variance Recall of 10 random splits	$5.18398 \times 10^{-5}$
Mean F1 Score of 10 random splits	0.500137
Variance F1 score of 10 random splits	$5.5234 \times 10^{-5}$

K-Nearest Neighbours:

Mean Accuracy of 10 Random splits	0.80437
Variance of accuracy of 10 random splits	$1.4163 \times 10^{-5}$
Mean Precision of 10 random splits	0.68241
Variance precision of 10 random splits	0.00018129
Mean Recall of 10 random splits	0.393700
Variance Recall of 10 random splits	$8.295969 \times 10^{-5}$
Mean F1 Score of 10 random splits	0.49929
Variance F1 score of 10 random splits	0.0001011027

Logistic Regression:

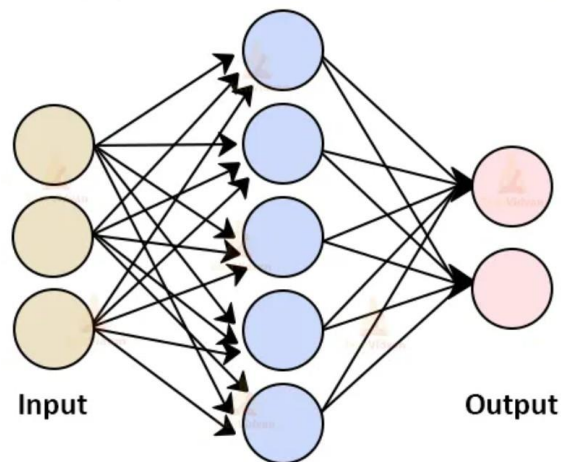
Mean Accuracy of 10 Random splits	0.80455
Variance of accuracy of 10 random splits	$1.4163 \times 10^{-5}$
Mean Precision of 10 random splits	0.68241
Variance precision of 10 random splits	0.00018129
Mean Recall of 10 random splits	0.39370011
Variance Recall of 10 random splits	$8.295969 \times 10^{-5}$
Mean F1 Score of 10 random splits	0.49929
Variance F1 score of 10 random splits	0.0001011027

*Comparison:*

Comparing the average F1 score of the four models, the Naive Bayes model with smoothing is the best model.

# Building a Basic Neural Network for Image Classification

Neural networks are a type of machine learning algorithm modeled after the human brain. They consist of layers of interconnected nodes called neurons that perform computations on input data. Neural networks are commonly used in image classification tasks, where the goal is to classify images into different categories based on their visual features.

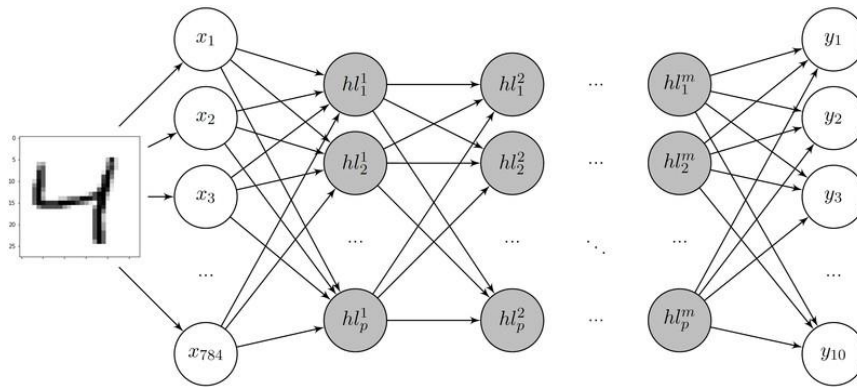


Steps to build a neural network for image classification:

1. Data preparation
2. Building the network architecture: The architecture of the neural network is defined by the number of layers, the number of neurons in each layer, and the activation functions used in each layer.
3. Training the network: This is done by adjusting the weights and biases of the neurons in the network using an optimization algorithm such as stochastic gradient descent.
4. Evaluating the network: Once the network has been trained, it is evaluated using the validation and testing sets of images.
5. Fine-tuning the network

## Methodology

The code uses the popular MNIST dataset, which consists of 60,000 training images and 10,000 test images of handwritten digits from 0 to 9. The aim is to build a neural network that can correctly classify these images into their respective classes.



First, the code displays an example image from the dataset using the matplotlib library to help understand the structure of the data.

Next, the dataset is preprocessed by dividing each pixel value by 255 to normalize them to be between 0 and 1. This helps the neural network converge faster and makes it less sensitive to variations in the pixel values.

The neural network architecture is defined using the Keras library's Sequential model. The input layer is a flattened version of the 28x28 pixel images, followed by two hidden layers. The activation function for the first two layers is ReLU, which is a non-linear function that helps the model learn more complex relationships between the input and output. The last layer uses a sigmoid activation function, which maps the output of the previous layer to a probability value between 0 and 1 for each of the 10 possible classes.

The model is then compiled using the `sparse_categorical_crossentropy` loss function, which is appropriate for multi-class classification problems. The optimizer used is Adam, which is a popular optimization algorithm that adapts the learning rate during training. The evaluation metric used is accuracy, which is a common metric for classification problems.

The model is trained on the training set for 5 epochs with a validation **split of 0.2, meaning that 20% of the training data** is used for validation. The history object contains the loss and accuracy values for both the training and validation sets at each epoch during training, which is used to plot the training history.

After training, the model's performance is evaluated on the test set by predicting the class labels for the test images using the `predict()` method. The `accuracy_score()` function from the scikit-learn library is used to calculate the accuracy of the model on the test set.

Layer	No. of Neurons	Activation Function	Accuracy
2	100, 50	tanh, tanh	0.9561
2	100, 50	tanh, sigmoid	0.9628
2	100, 50	tanh, relu	0.9609
2	100, 50	sigmoid, sigmoid	0.9651
2	100, 50	sigmoid, relu	0.9572
2	100, 50	relu, relu	0.9723
3	50, 50,50	tanh, tanh, tanh	0.9451
3	50, 50,50	tanh, relu, tanh	0.9506
3	50, 50,50	relu, tanh, sigmoid	0.9629
3	50, 50,50	relu, tanh, relu	0.9581
3	50, 50,50	relu, relu, relu	0.9616
3	50, 50,50	relu,relu,sigmoid	0.9658
3	50, 50,50	relu,sigmoid,relu	0.956
3	50, 50,50	relu,sigmoid,sigmoid	0.9611
3	50, 50,50	sigmoid, sigmoid, sigmoid	0.9394

The average accuracy of 2 layered models      0.9624

The average accuracy of 3 layered models      0.9556

The average accuracy of all models      0.9583

### Comparison:

On comparing the results, the following can be concluded:

1. The best two hidden-layered model is when we used is ReLU function at layers having 100 and 50 neurons, respectively, with an accuracy of 0.9723.
2. The best three hidden-layered model is when we used the ReLU, relu, sigmoid function at the hidden layers having 50, 50, and 50 neurons, respectively, with an accuracy of 0.9658.
3. The best models (including 2 and 3 hidden layers) is when using two hidden-layered ReLU activation functions with 100 and 50 neurons, respectively. It achieved an accuracy of 0.9723.
4. It's been observed that compared to the other two activation functions, the relu activation function results in a higher accuracy pro-vided Ceteris Paribus holds. It could be due to the fact that relu results in a faster convergence because it has a definite slope (0 for -ve values and constant for +ve values).

confusion matrix:

The following confusion matrices are the most accurate.

- The diagonal elements represent the number of times the corresponding digit was correctly classified by the model.
- The element (i, j), where  $i \neq j$ , means that the element was actually 'digit- i' but predicted as 'digit- j'.

