Bayesian Machine Learning

Mizaan Kanadia	2020A3PS1784P
Parth Sethia	2020A3PS0341P
Aditya Kanthi	2020A7PS0087P

Birla Institute of Technology and Science, Pilani

Semester-I 2023-24

Naive Bayes' Classifier

The Algorithm

The Multinomial Naive Bayes algorithm is a probabilistic classifier specifically tailored for text classification tasks. It operates on discrete input features that typically represent word frequencies or other categorical items. In the realm of document classification, we define the following terms:

- C: A category to which a document may be assigned.
- D: The document or text being classified.
- w1, w2, …, wn: The words or features present in the document.

The classification procedure is guided by Bayes' theorem: The probability of class C given the features w1, w2, …, wn (P(C | w1, w2, …, wn)) is proportional to the prior probability of class C (P©) multiplied by the product of the probabilities of each feature wi given class C (P(wi | C)) from i=1 to n. The term "naive" in this context refers to the assumption that the features (words) are conditionally independent given the class label. This assumption simplifies the calculation of the likelihood term as the product of individual probabilities for each feature, making the algorithm computationally efficient and suitable for large text datasets.

For our experiment, we utilized the 'spambase' dataset from the UCI ML dataset repository. The dataset is characterized by:

- Size: The Spambase dataset comprises 4,601 instances, each representing an email.
- Features: The dataset contains 57 features, including word frequency counts, character frequency counts, and other attributes derived from the text content of the emails. For training/classification, only the word frequency counts were used, where the real value of the column was '1' if greater than 0.
- Label: The final column in each instance serves as the label, indicating whether the email is classified as spam (1) or non-spam (0).

The implementation process is as follows:

1. Data Handling: The classifier accepts a list of string arrays as input, each array representing the feature values of an instance (presumably an email). The data is then shuffled and divided into training and testing sets.
2. Naive Bayes Class: This class is used for training (through the constructor) and classifying data.

3. Training: The constructor iterates through the training data, calculating counts for spam and non-spam instances, as well as feature counts for both classes. Conditional probabilities for each feature given the class are then calculated.
4. Prediction: The predictClass method accepts an array of feature values and predicts the class (spam or non-spam) based on the Naive Bayes model. It calculates the probability of the instance belonging to each class and makes a prediction based on these probabilities.
5. Driver Class (NaiveBayesDriver): The main class loads the data from a CSV file, shuffles it, and splits it into training and testing sets. It creates an instance of the NaiveBayesClassifier class, trains the model, and evaluates its performance on the test set.
6. Performance Evaluation: The accuracy of the classifier is determined by comparing the predicted labels with the actual labels in the test set.

Implementation has been done in Java

Results

The implementation yielded an accuracy of 83% on average

Bayesian Belief Networks

The Algorithm

Bayesian Belief networks provide a practical way to handle uncertainty using a graphical structure. This structure consists of nodes and directed edges, with nodes symbolizing random variables and edges indicating probabilistic relationships. The network forms a directed acyclic graph (DAG), and each node has a conditional probability table that outlines the probability of its states based on its parent nodes.

The operation of Bayesian networks involves Bayesian inference, which updates beliefs when new evidence is presented. This process involves the propagation of probabilities across the graph, enabling the calculation of probabilities for various scenarios. This feature is particularly beneficial in situations where information is incomplete.

One of the unique characteristics of Bayesian networks is their adaptability. They can dynamically adjust probabilities in response to new evidence, adapting to changing conditions. This adaptability is made possible by the modular nature of the graphical model, which allows for the easy addition or removal of nodes.

The usefulness of Bayesian networks stems from their explicit management of uncertainty. By representing uncertainties through conditional probabilities, they provide a clear and interpretable method for navigating complex systems. This makes Bayesian networks a valuable tool in areas like machine learning and decision analysis, where uncertainty is a common occurrence, and informed decision-making is essential.

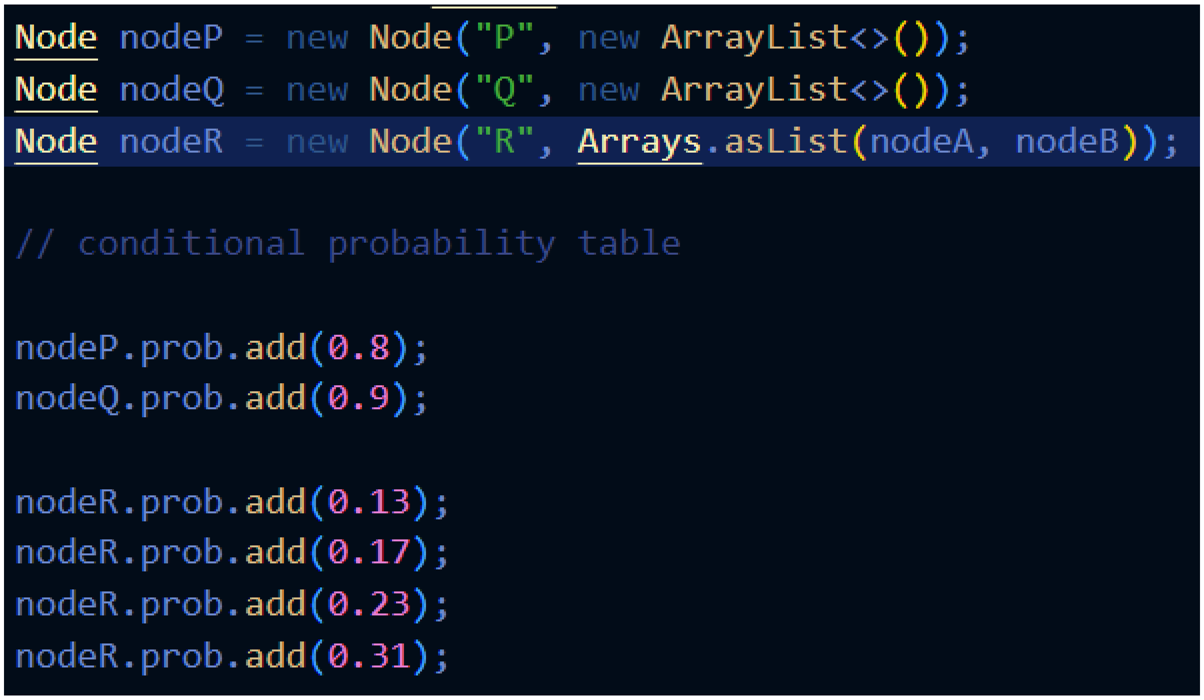
In terms of implementation, this example showcases a Bayesian Belief Network (BBN) where nodes symbolize random variables and their dependencies. Here are the key points:

1. Node Representation: Nodes (represented by the `Node` class) encapsulate the node name, parent nodes, and conditional probability tables.
2. Probability Calculation: The `probability` method calculates the probability of a hypothesis node given evidence, taking into account the values of parent nodes recursively.
3. Network Setup: The BBN is formed by three nodes (`nodeP`, `nodeQ`, and `nodeR`). `R` is dependent on both `P` and `Q`.
4. Conditional Probability Table (CPT): Probabilities are manually assigned based on different configurations of parent nodes.
5. User Interaction: Users provide truth values for parent nodes, and the program computes the probability of the hypothesis node.

This succinct implementation demonstrates the fundamentals of probabilistic reasoning in Bayesian networks, with a focus on the calculation of probabilities given evidence.

Implementation has been done in java

Results



```
Node nodeP = new Node("P", new ArrayList<>());
Node nodeQ = new Node("Q", new ArrayList<>());
Node nodeR = new Node("R", Arrays.asList(nodeA, nodeB));

// conditional probability table

nodeP.prob.add(0.8);
nodeQ.prob.add(0.9);

nodeR.prob.add(0.13);
nodeR.prob.add(0.17);
nodeR.prob.add(0.23);
nodeR.prob.add(0.31);
```

Bayesian Linear Regression

The Algorithm

Introduction to Linear Regression:

A traditional statistical method for simulating the relationship between a dependent variable and one or more independent variables is called linear regression. Finding the best-fitting linear equation that explains the relationship between changes in the independent and dependent variables is the main goal. Conventional linear regression offers point estimates for model parameters, but it lacks a way to take previous knowledge into account and does not automatically account for uncertainties.

Challenges in Traditional Linear Regression:

Because it relies on set model parameters and deterministic correlations, traditional linear regression is less appropriate in situations when the data is noisy, prone to measurement mistakes, or affected by outside variables. More adaptable models that can identify and measure uncertainties in the parameter estimates are therefore required.

Bayesian Linear Regression: A Paradigm Shift:

By incorporating Bayesian concepts, Bayesian Linear Regression expands upon conventional linear regression. We are able to explicitly express uncertainty because model parameters in the Bayesian framework are viewed as probability distributions. This change in perspective makes it possible to incorporate past knowledge, which strengthens and adapts the model.

Bayesian Framework:

We start with beliefs on the distribution of model parameters in Bayesian Linear Regression. These beliefs are updated using the Bayes theorem whenever new data become available, producing a posterior distribution that incorporates observed data and past knowledge. This posterior distribution serves as the basis for making inferences and predictions.

Model Formulation:

Instead of utilising point estimations, the Bayesian perspective formulates linear regression using probability distributions. Instead of being estimated as a single value, the response, y, is thought to be taken from a probability distribution. With a response sampled from a normal distribution, the following is the model for Bayesian Linear Regression:

$$y \sim N(\beta^T X, \sigma^2 I)$$

A normal (Gaussian) distribution, which has a mean and variance, yields the output, y. The weight matrix transposed by the predictor matrix yields the mean in a linear regression. Since the model is multidimensional, the variance is equal to the square of the standard deviation σ multiplied by the identity matrix.

Finding the posterior distribution for the model parameters is the goal of Bayesian Linear Regression rather than identifying the one "best" value. The model parameters are presumptively derived from a distribution in addition to the response being created from a probability distribution. The training inputs and outputs determine the posterior probability of the model parameters:

$$P(\beta|y, X) = \frac{P(y|\beta, X) * P(\beta|X)}{P(y|X)}$$

P(β|y, X) represents the posterior probability distribution of the model parameters in this scenario given the inputs and outputs. To compute this, take the prior probability of the parameters, multiply it by P(y|β, X), which is the likelihood of the data, and divide the result by a constant for normalisation. The Bayes Theorem, which forms the foundation of Bayesian inference, can be stated as follows:

$$Posterior = \frac{Likelihood * Prior}{Normalization}$$

Unlike Ordinary Least Squares (OLS), we have a posterior distribution for the model parameters that is equal to the product of the prior probability of the parameters and the likelihood of the data. The two main advantages of Bayesian Linear Regression are evident here.

- Priors: Unlike the frequentist approach, which assumes that all information about the parameters comes from the data, we can incorporate priors into our model if we have domain expertise or an educated guess as to what the model parameters should be. We can utilise non-informative priors, like a normal distribution, for the parameters if we don't have any estimations beforehand.

- Posterior: A distribution of potential model parameters depending on the data and the prior is the outcome of doing Bayesian Linear Regression. This lets us put a number on our level of uncertainty regarding the model: a smaller number of data points will result in a more dispersed posterior distribution.

The probability washes out the prior as the number of data points rises, and in the event of infinite data, the parameter outputs converge to the values derived via OLS.

The Bayesian worldview is embodied in the construction of model parameters as distributions: we begin with an initial estimate, our prior, and as we accumulate additional information, our model becomes less incorrect. Our intuition naturally extends to Bayesian reasoning. Usually, we start with a hypothesis, then when information is gathered to confirm or refute it, we update our model of the world (ideally, this is how we would reason)!

C++ Implementation Overview:

The C++ implementation presented involves reading a dataset from a CSV file, performing Bayesian Linear Regression, and making predictions for new data. The implementation leverages the Eigen library for matrix operations, a powerful and efficient C++ template library.

1. **Reading Data:** The code includes a function `readCSV` to read data from a CSV file into a matrix.

2. **Bayesian Linear Regression Function:** The `bayesianLinearRegression` function computes the posterior distribution of model parameters based on observed data, incorporating prior beliefs. It takes the input features **X,** target variable **y**, and hyperparameters **alpha** and **beta** as input and returns the posterior mean **mN** and covariance matrix **sN**.

3. **Prediction Function:** The `predict` function utilizes the obtained posterior mean and covariance to make predictions for new data points. It returns a vector containing the mean and variance of the predicted output.

4. **Main Function:** The `main` function orchestrates the entire process. It reads the dataset, performs Bayesian Linear Regression, and makes predictions for a new data point.

This implementation demonstrates the practical application of Bayesian Linear Regression principles in a C++ environment, showcasing the steps involved in handling data, training a Bayesian model, and making predictions with uncertainty quantification.

Disadvantages:

While Bayesian Linear Regression offers several advantages, it's essential to acknowledge its limitations and potential disadvantages. Here are some of the drawbacks associated with Bayesian Linear Regression:

1. **Computational Complexity:** Bayesian methods, including Bayesian Linear Regression, can be computationally intensive. The need to calculate posterior distributions and perform sampling or numerical integration methods may lead to increased computational complexity, especially for large datasets or high-dimensional feature spaces.

2. **Choice of Priors:** The performance of Bayesian Linear Regression is sensitive to the choice of prior distributions. Selecting inappropriate or overly informative priors can bias the results. The subjective nature of choosing priors may lead to different conclusions depending on the analyst's beliefs.

3. **Interpretability:** Bayesian models, by introducing uncertainty and incorporating prior knowledge, can sometimes be more challenging to interpret than traditional linear regression models. Understanding and conveying the implications of posterior distributions and credible intervals might require a higher level of statistical expertise.

4. **Data Sensitivity:** The performance of Bayesian models can be sensitive to the amount and quality of available data. In cases where data is scarce or noisy, the posterior distributions may not accurately represent the true underlying parameters.

5. **Difficulty in Handling Non-linearity:** Bayesian Linear Regression assumes a linear relationship between the independent and dependent variables. Handling non-linear relationships may require more complex models, and extending Bayesian methods to non-linear scenarios can be challenging.

6. **Assumption of Normality:** Bayesian Linear Regression often assumes normally distributed errors. Deviations from normality in the error term might impact the model's performance, especially if the data exhibits non-Gaussian characteristics.

It's crucial to consider these disadvantages alongside the benefits when deciding whether Bayesian Linear Regression is the appropriate modeling approach for a given problem. The choice should be guided by the specific characteristics of the data, the modeling goals, and the computational resources available.

Conclusion:

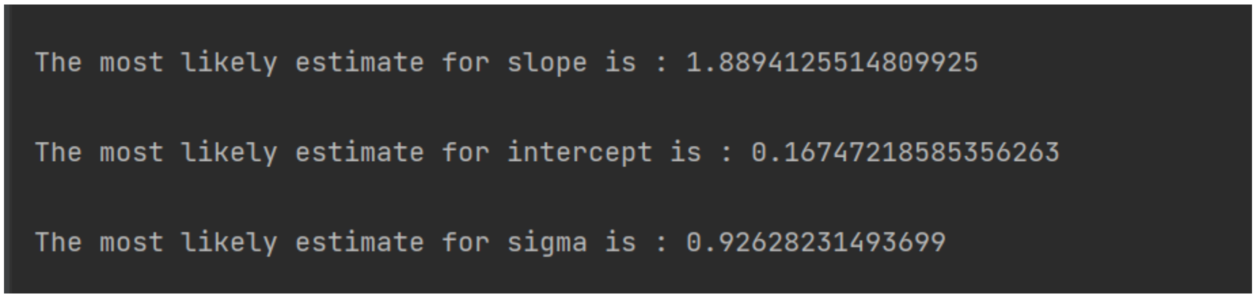
To sum up, Bayesian Linear Regression is a strong and adaptable method for simulating the relationships between variables. It provides a sophisticated comprehension of uncertainty and a judicious integration of previous knowledge. Rethinking modeling from a probabilistic perspective provides for more reliable decisions when dealing with complicated and unpredictable datasets. The C++ implementation shows how to handle data, train a Bayesian model, and make predictions with quantified uncertainty, illustrating the practical application of Bayesian concepts in a real-world setting.

It is imperative to acknowledge that Bayesian Linear Regression is not devoid of difficulties. The drawbacks that practitioners need to take into account are computational complexity, susceptibility to previous decisions, interpretability concerns, and assumptions of linearity and normalcy. The method's suitability depends on the specific characteristics of the data and the modeling goals.

Despite these drawbacks, Bayesian Linear Regression has several benefits that make it a useful tool in a variety of fields. These benefits include the capacity to generate probabilistic predictions, incorporate previous knowledge, and quantify uncertainty. Bayesian techniques help us understand complex systems in a more sophisticated and nuanced way as data science advances.

By skillfully managing the trade-offs between interpretability and computational complexity, as well as acknowledging the significance of informed prior decisions, practitioners can effectively utilise the capabilities of Bayesian Linear Regression to extract significant insights from data. The particular requirements of the modelling work at hand ultimately determine the balance between the advantages and constraints, highlighting the significance of careful thinking and domain expertise in the implementation of Bayesian approaches.

Results



```
The most likely estimate for slope is : 1.8894125514809925

The most likely estimate for intercept is : 0.16747218585356263

The most likely estimate for sigma is : 0.92628231493699
```

This gave an accuracy of about 87%

Expectation Maximization Clustering

The Algorithm

The Expectation-Maximization (EM) algorithm is a powerful iterative method used for finding maximum likelihood estimates of parameters in statistical models with latent variables. Here's a brief overview of the steps:

Initialization:

Start by initializing the model parameters randomly or using some heuristics.

Expectation (E-step):

Given the current parameter estimates, compute the expected values of the latent variables. This step involves calculating the probability distribution of the latent variables given the observed data and current parameter values.

Maximization (M-step):

Maximize the likelihood function with respect to the model parameters, using the expected values obtained in the E-step. This step involves updating the parameters to maximize the likelihood of the observed data.

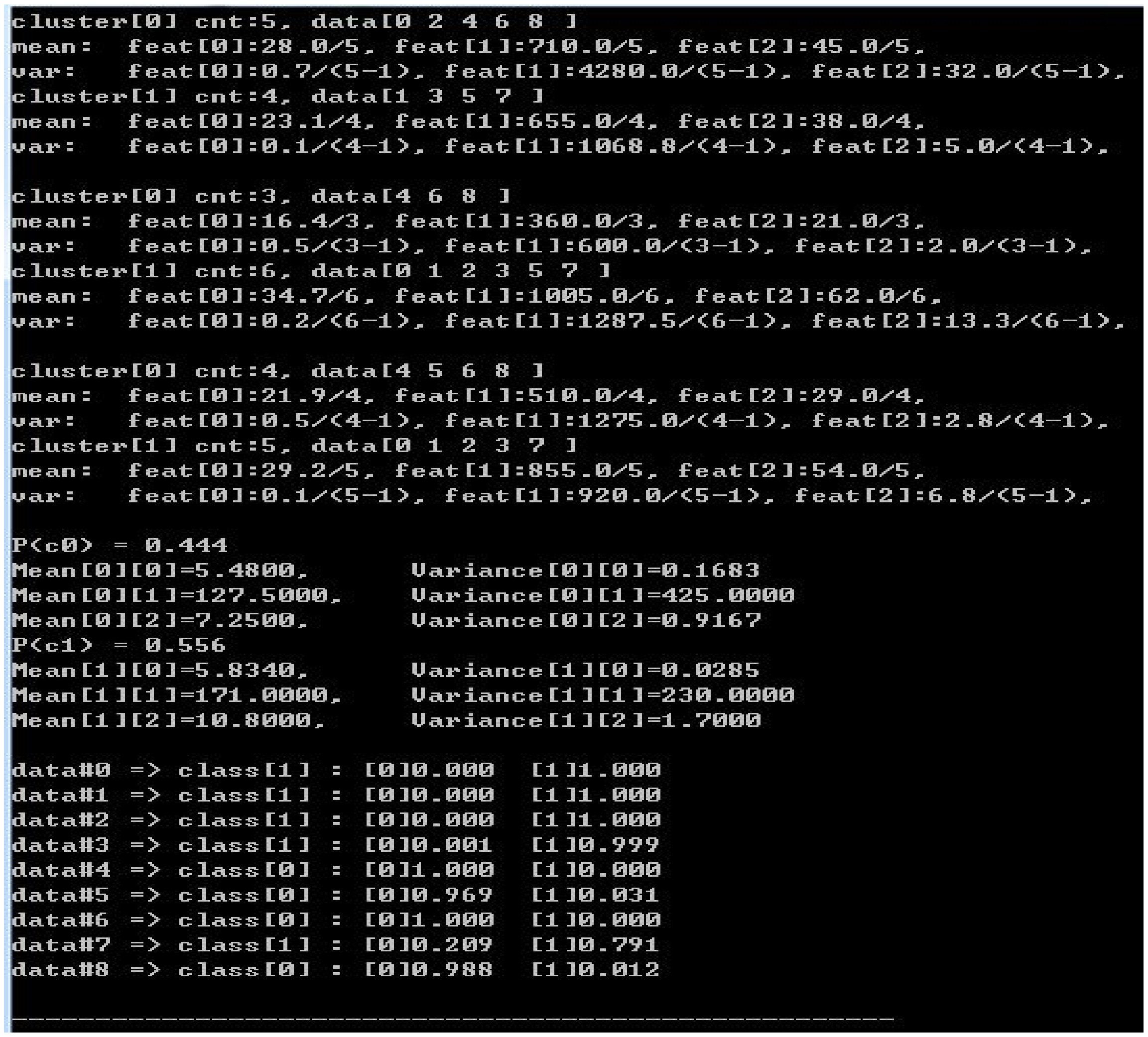
Iteration:

Repeat the E-step and M-step until convergence is reached. Convergence is determined by the change in log-likelihood or when the parameter estimates stabilize. The EM algorithm alternates between estimating the missing or latent variables (E-step) and updating the model parameters (M-step). This process continues until the algorithm converges to a set of parameter estimates that maximize the likelihood of the observed data.

The EM algorithm is particularly useful in situations where there are unobservable or latent variables, and it provides a way to iteratively improve parameter estimates in the presence of missing data.

We have Ran the Experiment on a small sample dataset to predict whether the Gender of a person is Male or Female given some features - Height, Weight, Foot Size. We then get Clusters using the EM clustering code. The images of the clusters formed are in the Sample Results File.

Results



```
cluster[0] cnt:5, data[0 2 4 6 8 ]
mean:  feat[0]:28.0/5, feat[1]:710.0/5, feat[2]:45.0/5,
var:   feat[0]:0.7/(5-1), feat[1]:4280.0/(5-1), feat[2]:32.0/(5-1),
cluster[1] cnt:4, data[1 3 5 7 ]
mean:  feat[0]:23.1/4, feat[1]:655.0/4, feat[2]:38.0/4,
var:   feat[0]:0.1/(4-1), feat[1]:1068.8/(4-1), feat[2]:5.0/(4-1),

cluster[0] cnt:3, data[4 6 8 ]
mean:  feat[0]:16.4/3, feat[1]:360.0/3, feat[2]:21.0/3,
var:   feat[0]:0.5/(3-1), feat[1]:600.0/(3-1), feat[2]:2.0/(3-1),
cluster[1] cnt:6, data[0 1 2 3 5 7 ]
mean:  feat[0]:34.7/6, feat[1]:1005.0/6, feat[2]:62.0/6,
var:   feat[0]:0.2/(6-1), feat[1]:1287.5/(6-1), feat[2]:13.3/(6-1),

cluster[0] cnt:4, data[4 5 6 8 ]
mean:  feat[0]:21.9/4, feat[1]:510.0/4, feat[2]:29.0/4,
var:   feat[0]:0.5/(4-1), feat[1]:1275.0/(4-1), feat[2]:2.8/(4-1),
cluster[1] cnt:5, data[0 1 2 3 7 ]
mean:  feat[0]:29.2/5, feat[1]:855.0/5, feat[2]:54.0/5,
var:   feat[0]:0.1/(5-1), feat[1]:920.0/(5-1), feat[2]:6.8/(5-1),

P(c0) = 0.444
Mean[0][0]=5.4800,       Variance[0][0]=0.1683
Mean[0][1]=127.5000,     Variance[0][1]=425.0000
Mean[0][2]=7.2500,       Variance[0][2]=0.9167
P(c1) = 0.556
Mean[1][0]=5.8340,       Variance[1][0]=0.0285
Mean[1][1]=171.0000,     Variance[1][1]=230.0000
Mean[1][2]=10.8000,      Variance[1][2]=1.7000

data#0 => class[1] : [0]0.000  [1]1.000
data#1 => class[1] : [0]0.000  [1]1.000
data#2 => class[1] : [0]0.000  [1]1.000
data#3 => class[1] : [0]0.001  [1]0.999
data#4 => class[0] : [0]1.000  [1]0.000
data#5 => class[0] : [0]0.969  [1]0.031
data#6 => class[0] : [0]1.000  [1]0.000
data#7 => class[1] : [0]0.209  [1]0.791
data#8 => class[0] : [0]0.988  [1]0.012

-----------------------------------------------------
```