Bayesian Modelling

Bayesian modelling is a statistical approach that allows us to build and update models of complex systems using probabilistic methods.

In simple terms, Bayesian modelling is like a game of guessing. You make an initial guess about something, and then you collect some data that might help you to refine your guess. Based on the data, you update your guess and make a new, more informed guess.

For example, suppose you want to predict the outcome of a coin toss. Your initial guess might be that there is a 50/50 chance of the coin landing on heads or tails. You flip the coin and see that it lands on heads. Based on this data, you update your guess and make a new guess that the probability of the coin landing on heads is now higher than 50%.

In Bayesian modelling, we use probability distributions to represent our beliefs or uncertainties about the parameters of a model. We update these probability distributions based on new data, using Bayes' theorem to compute the posterior probability distribution. This allows us to make predictions and estimates based on probabilistic inference.

Bayesian modelling is used in many fields, such as finance, engineering, physics, and biology, to model complex systems and make predictions and decisions based on data and uncertainty.

Posterior And Prior Probability

Posterior probability and prior probability are two important concepts in Bayesian probability theory, which is a branch of statistics that uses Bayes' theorem to update probability estimates based on new data or evidence. Here is an explanation of these two concepts and a real-life example:

Prior probability: Prior probability refers to our initial belief or probability estimate about an event or outcome before we have any new evidence or data. It is usually based on our prior knowledge, experience, or assumptions. For example, if we want to estimate the probability of getting heads in a coin toss, our prior probability would be 0.5, assuming the coin is fair.

Posterior probability: Posterior probability refers to the updated probability estimate of an event or outcome after we have new evidence or data. It is calculated using Bayes' theorem,

which combines the prior probability with the likelihood of the new evidence. For example, if we toss a coin and get heads, our posterior probability of getting heads in the next toss would be higher than 0.5 if we assume that the coin is biased towards heads.
Real-life example: Let's say you want to estimate the probability of a person having diabetes based on their age, gender, weight, and blood sugar level. Your prior probability of a person having diabetes might be based on the prevalence of diabetes in the general population, which is about 8%. However, if you find out that the person is over 50 years old, female, overweight, and has a high blood sugar level, you can update your probability estimate using Bayes' theorem to get a posterior probability that is higher than 8%, reflecting the increased risk factors associated with these characteristics.

In this example, the prior probability would be the initial estimate of the person having diabetes based on general population data, while the posterior probability would be the updated estimate based on the new evidence of age, gender, weight, and blood sugar level.

Bayes Theorem

Suppose you have developed a new diagnostic test for a rare disease that affects 1 in 1000 people. The test is 95% accurate, meaning that if a person has the disease, there is a 95% chance that the test will detect it, and if a person does not have the disease, there is a 95% chance that the test will give a negative result.

You administer the test to a patient who tests positive for the disease. What is the probability that the patient has the disease?

To solve this problem using Bayes' theorem, we need to define the following probabilities:
P(D) = Prior probability of having the disease = 0.001
P(~D) = Prior probability of not having the disease = 0.999
P(Pos|D) = Probability of testing positive given that the patient has the disease = 0.95
P(Neg|~D) = Probability of testing negative given that the patient does not have the disease = 0.95

Using Bayes' theorem, we can calculate the posterior probability of having the disease given a positive test result:
P(D|Pos) = P(Pos|D) * P(D) / [P(Pos|D) * P(D) + P(Pos|~D) * P(~D)]

Denominator represents the total number of people who test positive, regardless of whether they have the disease or not. The numerator represents the subset of people who test positive and have the disease. By dividing the numerator by the denominator, we get the posterior probability of having the disease given a positive test result. This helps us update our prior belief in the probability of having the disease based on the new evidence of the test result.

The denominator in the formula P(D|Pos) = P(Pos|D) * P(D) / [P(Pos|D) * P(D) + P(Pos|~D) * P(~D)] represents the probability of testing positive (Pos) regardless of whether the person has the disease (D) or not (~D). This is called the marginal probability of testing positive, and it is calculated by summing the joint probabilities of testing positive and having the disease and testing positive and not having the disease:

P(Pos) = P(Pos|D) * P(D) + P(Pos|~D) * P(~D)

The denominator represents the total probability of observing a positive result, regardless of whether the true underlying condition is present or not. It serves as a normalizing constant that ensures that the conditional probability on the left-hand side of the equation is a valid probability (i.e., it sums to 1).

The denominator consists of two terms:

P(Pos|D) * P(D): the probability of observing a positive result given that the condition is present, multiplied by the prior probability that the condition is present.

P(Pos|~D) * P(~D): the probability of observing a positive result given that the condition is not present, multiplied by the prior probability that the condition is not present.

These two terms represent the probability of observing a positive result under both the presence and absence of the condition, respectively. They are weighted by the prior probabilities of the presence and absence of the condition, respectively.

= 0.95 * 0.001 / [0.95 * 0.001 + 0.05 * 0.999]
= 0.019

So, the probability that the patient actually has the disease given a positive test result is only 1.9%. This is much lower than the initial probability of 0.1%, indicating that the positive test result is not very strong evidence for the presence of the disease.
This example shows how Bayes' theorem can be used to update our beliefs or probabilities based on new evidence, and how the accuracy of a diagnostic test depends not only on its sensitivity and specificity but also on the prevalence of the disease in the population being tested.

Numerical Example

- Suppose a factory produces two types of products: Type A and Type B.
- Historically, 70% of the products produced are Type A and 30% are Type B.
- The factory has two machines that produce these products, Machine 1, and Machine 2.
- Machine 1 produces 95% Type A products and 5% Type B products, while Machine 2 produces 80% Type A products and 20% Type B products.
- One of the products is randomly selected from the factory and it is found to be a Type A product. What is the probability that it was produced by Machine 1?

To solve this problem using Bayes' theorem, we need to define the following probabilities:

P(M1) = Prior probability that the product was produced by Machine 1 = 0.5 (assuming an equal chance of being produced by each machine)

P(M2) = Prior probability that the product was produced by Machine 2 = 0.5 (assuming an equal chance of being produced by each machine)

P(A|M1) = Probability that the selected product is Type A given that it was produced by Machine 1 = 0.95

P(A|M2) = Probability that the selected product is Type A given that it was produced by Machine 2 = 0.8

Using Bayes' theorem, we can calculate the posterior probability that the product was produced by Machine 1 given that it is a Type A product:

$P(M1|A) = P(A|M1) * P(M1) / [P(A|M1) * P(M1) + P(A|M2) * P(M2)]$

$= 0.95 * 0.5 / [0.95 * 0.5 + 0.8 * 0.5]$

$= 0.542$

So, the probability that the product was produced by Machine 1 given that it is a Type A product is 54.2%. This indicates that Machine 1 is more likely to have produced the product, but there is still some uncertainty due to the fact that both machines have some probability of producing Type A products.

Naïve Bayes vs Bayes Theorem

Bayes' theorem and Naive Bayes theorem are related but different concepts.

Bayes' theorem is a mathematical formula that describes the relationship between conditional probabilities of two events. It allows us to update our beliefs or probabilities about the occurrence of an event based on new evidence or information.

Naive Bayes theorem, on the other hand, is a specific application of Bayes' theorem to solve classification problems in machine learning. In Naive Bayes, we make a naive assumption that the features (or attributes) of the data are independent of each other, which simplifies the calculation of the posterior probability distribution. This makes the algorithm computationally efficient and often leads to good results in practice.

For example, in text classification, Naive Bayes can be used to predict the category of a given text (such as spam or ham email) based on the words that appear in the text. We calculate the posterior probability of each category given the words in the text using Bayes' theorem, and then choose the category with the highest probability as the predicted category.

In summary, Bayes' theorem is a mathematical formula that describes the relationship between conditional probabilities, while Naive Bayes theorem is a specific application of Bayes' theorem to solve classification problems in machine learning.

Naïve Bayes Example:

Suppose we have a dataset of fruits labelled as either apples or oranges based on their color and diameter. We want to predict whether a new fruit is an apple, or an orange based on its color and diameter.
The training data looks like this:

Fruit	Color	Diameter	Type
1	Red	3	Apple
2	Red	3	Apple
3	Red	1	Orange
4	Yellow	1	Orange
5	Yellow	2	Orange
6	Yellow	3	Apple

Using Naive Bayes, we can calculate the probability of the new fruit being an apple given its color and diameter.

P (Apple | Color = Red, Diameter = 2) = P (Color = Red | Apple) * P (Diameter = 2 | Apple) * P(Apple) / P (Color = Red, Diameter = 2)

P (Color = Red | Apple) = 2/3 = 0.67
P (Diameter = 2 | Apple) = 0
P(Apple) = 3/6 = 0.5

P (Color = Red, Diameter = 2) = P (Color = Red | Apple) * P (Diameter = 2 | Apple) * P(Apple) + P (Color = Red | Orange) * P (Diameter = 2 | Orange) * P(Orange)
P (Color = Red, Diameter = 2) = 0.67 * 0 + 0.33 * 0.33 * 0.5

P (Apple | Color = Red, Diameter = 2) = 0.67 * 0 * 0.5 / (0.67 * 0 + 0.33 * 0.33 * 0.5)

Therefore, the probability of the new fruit being an apple given its color and diameter is 0.

Based on our Naive Bayes model, we predict that the new fruit is likely to be an orange.

Real Life Example of probabilistic modeling and inference

One real-life example of probabilistic modeling and inference is in weather forecasting. Weather forecasting is a complex process that involves predicting the probability of various weather conditions occurring in the future.

In weather forecasting, probabilistic modeling is used to represent the uncertainty in the data and estimate the probability of different weather conditions occurring. Meteorologists use a variety of models to make these predictions, including Bayesian networks and Hidden Markov models.

Bayesian inference is used to update the probabilities based on new data, such as current weather conditions and satellite imagery. This allows meteorologists to make more accurate predictions about future weather conditions.

For example, suppose a meteorologist wants to predict the probability of rain tomorrow. They might use a Bayesian network to model the relationships between different weather variables, such as temperature, humidity, and air pressure. The network would estimate the probability of rain based on these variables.

As new data becomes available, such as a drop in air pressure or an increase in humidity, the meteorologist can update the probabilities using Bayesian inference. This would allow them to make a more accurate prediction of the probability of rain tomorrow.

Overall, probabilistic modeling and inference are essential tools in weather forecasting, allowing meteorologists to make more accurate predictions and help people prepare for future weather conditions.

Introduction to k-NN Algorithm

k-Nearest Neighbors is a non-parametric machine learning algorithm used for classification and regression tasks. It is a simple yet effective algorithm that can be used for a wide range of problems.

The k-NN algorithm works by finding the k-nearest neighbours of a new data point and assigning it to the class that occurs most frequently among those neighbors. The value of k is a hyperparameter that needs to be set prior to training the algorithm.

Example of k-NN Algorithm

Let's consider a simple example to understand how the k-NN algorithm works for classification tasks.

Suppose we have a dataset of 50 flowers, where each flower has two features: the length and width of its petals. The dataset is labeled with three different classes: Iris Setose, Iris Versicolour, and Iris Virginica.

Now, suppose we want to classify a new flower based on its petal length and width. We can use the k-NN algorithm to find the k-nearest neighbors of the new flower in the dataset and assign it to the class that occurs most frequently among those neighbors.

For example, if we set k=5, the k-NN algorithm would find the five nearest neighbors of the new flower in the dataset based on their petal length and width. Let's say that three of these

neighbors belong to the Iris Setosa class and two of them belong to the Iris Versicolour class. In this case, the k-NN algorithm would classify the new flower as Iris Setosa, since it is the most common class among the five nearest neighbors.

Advantages and Disadvantages of k-NN Algorithm

Advantages

1. k-NN is a simple and easy-to-understand algorithm that does not require any assumptions about the underlying distribution of the data.
2. It can be used for both classification and regression tasks.
3. k-NN is a non-parametric algorithm, which means it does not make any assumptions about the functional form of the data.

Disadvantages

1. The main drawback of k-NN is its computational complexity, especially when dealing with large datasets.
2. The choice of the value of k can have a significant impact on the performance of the algorithm.
3. The k-NN algorithm is sensitive to irrelevant features in the dataset.

Example

Suppose we have a dataset of labeled flowers with two features, petal length and petal width. We have 6 flowers in our dataset with the following labels and features:

Flower	Petal length	Petal width	Label
1	1.4	0.2	setosa
2	1.3	0.2	setosa
3	4.7	1.4	versicolor
4	4.9	1.5	versicolor
5	5.1	1.9	virginica
6	5.7	2.3	virginica

Now, let's say we have a new flower with petal length 5.0 and petal width 1.6, and we want to classify it based on its features using k-NN algorithm with k=3.

First, we need to calculate the Euclidean distance between the new flower and each flower in our dataset:

Flower	Petal length	Petal width	Distance
1	1.4	0.2	sqrt((5.0-1.4)^2+(1.6-0.2)^2)
2	1.3	0.2	sqrt((5.0-1.3)^2+(1.6-0.2)^2)
3	4.7	1.4	sqrt((5.0-4.7)^2+(1.6-1.4)^2)
4	4.9	1.5	sqrt((5.0-4.9)^2+(1.6-1.5)^2)
5	5.1	1.9	sqrt((5.0-5.1)^2+(1.6-1.9)^2)
6	5.7	2.3	sqrt((5.0-5.7)^2+(1.6-2.3)^2)

Finally, we classify the new flower by taking a majority vote among the labels of the k nearest neighbors.

SVM

Support Vector Machines (SVM) are a type of supervised learning algorithm used for classification and regression analysis. The basic idea behind SVM is to find a hyperplane that best separates the classes in the feature space.

In a binary classification problem, we have a set of input data points with corresponding labels that belong to one of two classes (positive or negative). The goal of SVM is to find a hyperplane that maximizes the margin between the two classes.

The margin is the distance between the hyperplane and the closest data points from each class. The hyperplane that maximizes the margin is the one that best separates the classes and is called the maximum margin hyperplane.

Mathematically, we can represent the maximum margin hyperplane as a linear equation:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

or

$$\mathbf{w}^T\mathbf{x}=0$$

where w is the weight vector perpendicular to the hyperplane, x is the input data point, and b is the bias term. The goal of SVM is to find the optimal values of w and b that maximize the margin while correctly classifying the data points.

Both w^T x = 0 and wx + b = 0 are equations that define a decision boundary in a linear classifier.

The equation w^T x = 0 is the equation of a hyperplane that passes through the origin. Here, w is a weight vector and x is the input vector. The decision boundary separates the space into two regions, where one region corresponds to one class and the other region corresponds to another class.

On the other hand, the equation wx + b = 0 is the equation of a hyperplane that does not necessarily pass through the origin. Here, w is a weight vector, x is the input vector, and b is a bias term. The decision boundary separates the space into two regions, where one region corresponds to one class and the other region corresponds to another class.

In practice, the equation wx + b = 0 is more commonly used in SVM (Support Vector Machines) to define the decision boundary. This is because it allows for greater flexibility in the placement of the decision boundary, as the bias term b can be adjusted to move the hyperplane along the direction of the weight vector w.

To solve this optimization problem, SVM introduces the concept of support vectors, which are the data points closest to the hyperplane. These support vectors define the margin and are used to determine the optimal hyperplane.

The objective function of SVM can be expressed as:

$$\text{argmax}(w^*, b^*) \ \frac{2}{\|w\|} \text{ such that } y_i\left(\vec{w}.\vec{X} + b\right) \geq 1$$

Solving this optimization problem involves finding the Lagrange multipliers of the constraint equation. The optimal values of w and b can then be computed using the support vectors and their corresponding Lagrange multipliers.

In summary, SVM finds the optimal hyperplane that maximizes the margin between the classes by solving an optimization problem that minimizes the norm of the weight vector subject to constraints that ensure correct classification and a margin of at least 1.

Why Optimal Hyperplane?

There are several reasons to find the optimal hyperplane in SVM:

Better Generalization: The optimal hyperplane tends to generalize better to new, unseen data compared to suboptimal hyperplanes. This is because the optimal hyperplane is less sensitive to noise in the data and captures the true underlying structure of the data. It helps in generalization by Reducing Overfitting

Improved Separation: The optimal hyperplane provides better separation between the two classes, resulting in better classification accuracy. This is especially true in situations where the data is not linearly separable, as the optimal hyperplane can be chosen to fit the data better than suboptimal hyperplanes.

Robustness to Outliers: The optimal hyperplane is less affected by outliers in the data than suboptimal hyperplanes. This is because the optimal hyperplane is chosen to maximize the margin between the classes, and outliers are typically far from the decision boundary.

Impact of Outlier on SVM:

Outliers have the capability to make your model poor. The margin will shrink and the decision boundary will be sub-optimal resulting in poor classification.

In the presence of outliers, you need to use a more general version of the Support Vector Machine that is with soft margins.

Interpretability: The optimal hyperplane provides an interpretable model that can help understand the relationship between the input variables and the class labels. This is because the optimal hyperplane is chosen to be the simplest one that separates the two classes, which can lead to a more interpretable model.

Kernels

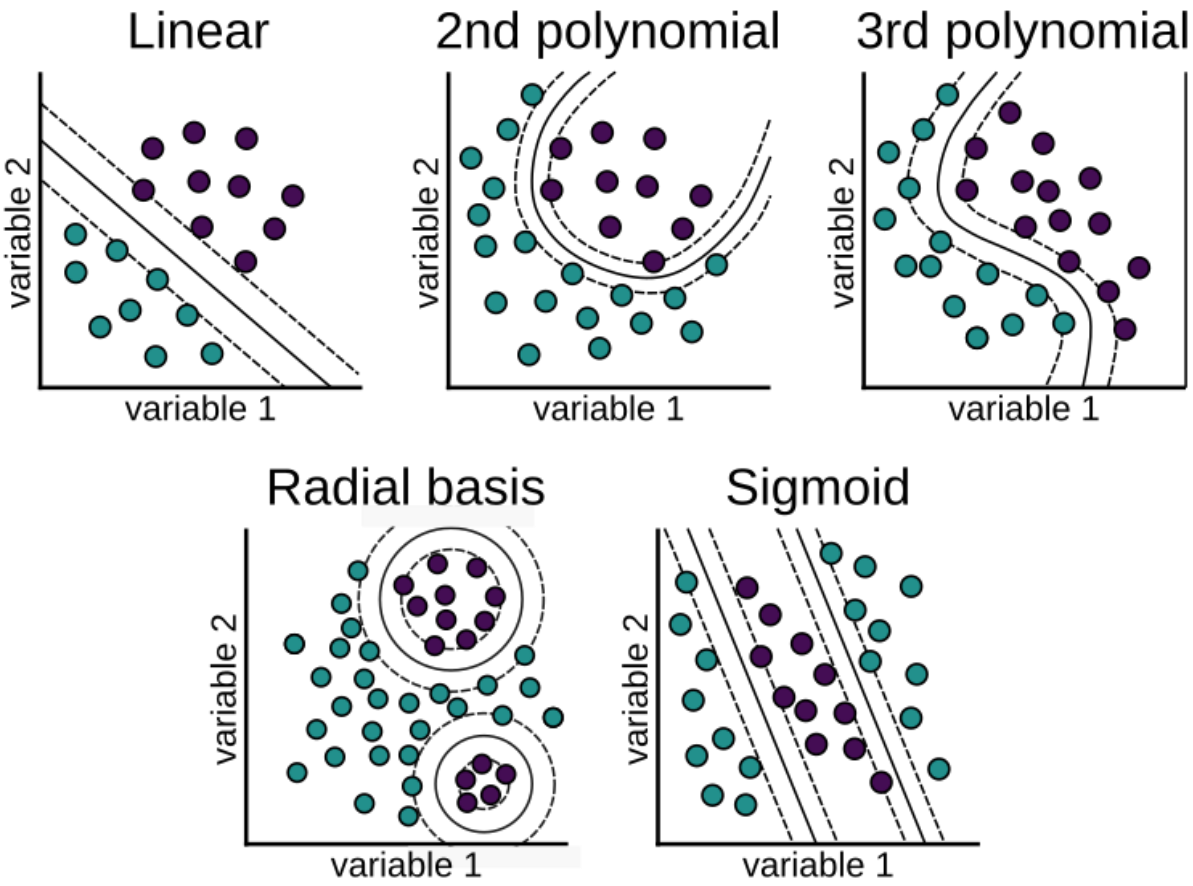
There are several functions SVM uses to perform this task. Some of the most common ones are:

1. **Polynomial Kernel Function**: This transforms data points by using the dot product and transforming data to an ‘n-dimension’, n could be any value from 2, 3 et cetera, i.e the transformation will be either a squared product or higher. Therefore representing data in higher-dimensional space using the new transformed points.

2. **The Radial Basis Function(RBF)**: This function behaves like a ‘weighted nearest neighbor model’. It transforms data by representing in infinite dimensions, then using the weighted nearest neighbor (observation with the most influence on the new data point) for classification. The Radial function can be either Gaussian or Laplace. This is dependent on a hyperparameter known as gamma. This is the most commonly used kernel.

3. **The Sigmoid Function**: also known as the hyperbolic tangent function(Tanh), finds more application in neural networks as an activation function. This function is used in image classification.

4. **The Linear Kernel**: Used for linear data. This just simply represents the data points using a linear relationship.



SAMPLES

Name of the Kernel	Mathematical Formula
Linear	$k(x,y) = x^T.y$
Polynomial	$k(x,y) = (x^T, y)^P or k(x,y)$ $= (x^T.y + 1)^p$ where p is the polynomial degree
RBF(Gaussian)	$\phi(x) = \exp(-\frac{x^2}{2\sigma^2}), \sigma > 0$

- **The C-parameter**: This is a regularization parameter used to prevent overfitting. It is inversely related to the Margin, such that if a larger C value is chosen, the margin is smaller, and if a smaller C value is chosen the margin is larger. It aids with the trade-off between bias and variance. SVM just like most machine learning algorithms has to deal with this as well.