. **UNIT-3**

Bayesian learning

Maximum likelihood and least squared error hypothesis:Two commonly used approaches to estimate population parameters from a random sample are the maximum likelihood estimation method (default) and the least squares estimation method.

Maximum likelihood estimation method (MLE)

The likelihood function indicates how likely the observed sample is as a function of possible parameter values. Therefore, maximizing the likelihood function determines the parameters that are most likely to produce the observed data. From a statistical point of view, MLE is usually recommended for large samples because it is versatile, applicable to most models and different types of data, and produces the most precise estimates.

Least squares estimation method (LSE)

Least squares estimates are calculated by fitting a regression line to the points from a data set that has the minimal sum of the deviations squared (least square error). In reliability analysis, the line and the data are plotted on a probability plot.

- Bayesian justification for many ANN and other curve fitting methods
- Probability density (necessary for continuous target)
- the random noise variable, , is generated by a Normal probability distribution (i.e., smooth bell-shaped distribution that can be completely characterized by its mean , and standard deviation), where random noise is drawn independently from a Normal distribution with 0 mean
- maximum likelihood hypothesis might not be MAP hypothesis, but if one assume uniform prior probabilities over the hypothesis then it is
- Does noise have normal distribution - noise generated by the sum of very many independent identically distributed random variables will be Normally distributed regardless of the distributions of the individual variables
- in reality different components to noise probably don't have identical distributions
- above analysis only allows noise in the target - for instance when predicting weight based on height and age analysis assumes noise in the measurement of weight but perfect measurements of height and age - analysis is significantly more complicated if this simplifying assumption is removed

Minimum Description Length The minimum description length (MDL) criteria in machine learning says that the best description of the data is given by the model which compresses it the best. Put another way, learning a model for the data or predicting it is about capturing the regularities in the data and any regularity in the data can be used to compress it. Thus, the more we can compress a data, the more we have learnt about it and the better we can predict it.

MDL is also connected to Occam's Razor used in machine learning which states that "other things being equal, a simpler explanation is better than a more complex one." In MDL, the simplicity (or rather complexity) of a model is interpreted as the length of the code obtained when that model is used to compress the data.

The ideal version of MDL is given by the Kolmogorov Complexity, which is defined as the length of the shortest computer program that prints the sequence of observed data and halts. However, the Komogorov

Complexity is uncomputable i.e. it can be shown that there exists no computer program that, for every set of data D, when given D as input, returns the shortest program that prints D). Moreover, for finite length sequences, the best program may depend on the sequence itself. For fruther reading on Kolmogorov Complexity, see Chapter 14 of Thomas and Cover.

Practical versions of MDL are either based on one stage universal codes (known as refined MDL) or two-stage codes (known as crude MDL). The refined MDL suggest that, for a given class of models, pick the model which minimizes the worst case redundancy on the data. This leads to precisely the universal models such as mixture models and normalized maximum likelihood (NML) model, which we discussed in last few lectures:

Mixture model: qγ(x n) = X q∈Qγ π(q) q(x n) where π is chosen to be (nearly) minimax optimal prior for class Qγ

NML model: qγ(x n) = = arg max q∈Qγ q(x n)/ P xn maxq∈Qγ q(x n)

As we discussed earlier, the mixture models are ususally prefered as they amenable to sequential modeling and hence to design of practical arithmetic codes based on those sequential models.

The crude MDL or two-stage code approach is based on the notion that we can specify the descriptive properties of a model for data in two stages - (i) encode the model with some codelength L(q), (ii) encode the data using the model with codelength Lq(x n). Now pick the model which minimizes the total codelength of the two-stage code:

qγ(x n) = arg min q∈Qγ {L(q) + Lq(x n)}

While this procedure can be used for model selection within a class, we will discuss that later. First, we first look at using the two-stage coding approach to select the best model class.

Bayes Optimal Classifier

The Bayes Optimal Classifier is **a probabilistic model that makes the most probable prediction for a new example**. It is described using the Bayes Theorem that provides a principled way for calculating a conditional probability.

Defined as the label produced by the most probable classifier

Computational learning theory

Computational learning theory (CoLT) is a branch of AI concerned with using mathematical methods or the design applied to computer learning programs. It involves using mathematical frameworks for the purpose of quantifying learning tasks and algorithms.

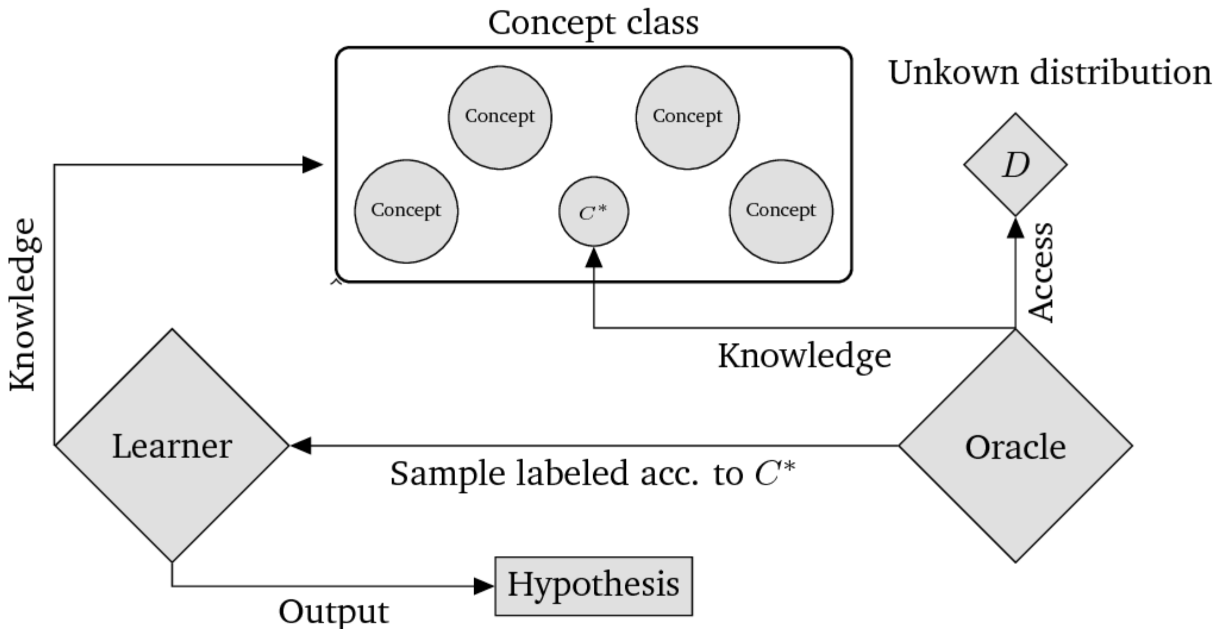
It seeks to use the tools of theoretical computer science to quantify learning problems. This includes characterizing the difficulty of learning specific tasks.

Computational learning theory can be considered to be an extension or sibling of statistical learning theory or SLT for short, that makes use of formal methods for the purpose of quantifying learning algorithms.

- Computational Learning Theory (CoLT): Formal study of learning tasks.
- Statistical Learning Theory (SLT): Formal study of learning algorithms.

This division of learning tasks vs. learning algorithms is arbitrary, and in practice, there is quite a large degree of overlap between these two fields.

Computational learning theory is essentially a sub-field of artificial intelligence (AI) that focuses on studying the design and analysis of machine learning algorithms.



Source: Semantic Scholar

Computational learning theory provides a formal framework in which it is possible to precisely formulate and address questions regarding the performance of different learning algorithms. Thus, careful comparisons of both the predictive power and the computational efficiency of competing learning algorithms can be made. Three key aspects that must be formalized are:

- The way in which the learner interacts with its environment,
- The definition of success in completing the learning task,
- A formal definition of efficiency of both data usage (sample complexity) and processing time (time complexity).

It is important to remember that the theoretical learning models are abstractions of real-life problems. Close connections with experimentalists are useful to help validate or modify these abstractions so that the theoretical results reflect empirical performance. The computational learning theory research has therefore close connections to machine learning research. Besides the model's predictive capability, the computational learning theory also addresses other important features such as simplicity, robustness to variations in the learning scenario, and an ability to create insights to empirically observed phenomena.

Probably learning an Approximately Correct Learning

Rather than just studying different learning algorithms that happen to work well, **computational learning theory** investigates general principles that can be proved to hold for classes of learning algorithms.
Some relevant questions that we can ask about a theory of computational learning include the following:

- Is the learner guaranteed to converge to the correct hypothesis as the number of examples increases?
-

How many examples are required to identify a concept?

-

How much computation is required to identify a concept?
In general, the answer to the first question is "no," unless it can be guaranteed that the examples always eventually rule out all but the correct hypothesis. An adversary out to trick the learner could choose examples that do not help discriminate correct hypotheses from incorrect hypotheses. If an adversary cannot be ruled out, a learner cannot guarantee to find a consistent hypothesis. However, given randomly chosen examples, a learner that always chooses a consistent hypothesis can get arbitrarily close to the correct concept. This requires a notion of closeness and a specification of what is a randomly chosen example.
Consider a learning algorithm that chooses a hypothesis consistent with all of the training examples. Assume a probability distribution over possible examples and that the training examples and the test examples are chosen from the same distribution. The distribution does not have to be known. We will prove a result that holds for all distributions.
The **error of hypothesis**
h∈Hh∈ℋ on instance space II, written error(I,h) error(I,h), is defined to be the probability of choosing an element ii of II such that h(i)≠Y(i)h⊠(i)≠Y(i), where h(i)h⊠(i) is the predicted value of target variable YY on possible example ii, and Y(i)Y(i) is the actual value of YY on example ii. That is,

error(I,h)=P(h(i)≠Y(i)|i∈I).error(I,h)=P(h(i)≠Y(i)|i∈I).

An agent typically does not know PP or Y(i)Y(i) for all ii and, thus, does not actually know the error of a particular hypothesis.
Given ϵ>0ϵ>0, hypothesis hh is **approximately correct** if error(I,h)≤ϵerror(I,h)≤ϵ.
We make the following assumption.
Assumption 7.3.
The training and test examples are chosen independently from the same probability distribution as the population.
It is still possible that the examples do not distinguish hypotheses that are far away from the concept. It is just very unlikely that they do not. A learner that chooses a hypothesis consistent with the training examples is **probably approximately correct** if, for an arbitrary number δδ (0<δ≤10<δ≤1), the algorithm is not approximately correct in at most δδ of the cases. That is, the hypothesis generated is approximately correct at least 1−δ1-δ of the time.
Under the preceding assumption, for arbitrary ϵϵ and δδ, we can guarantee that an algorithm that returns a consistent hypothesis will find a hypothesis with error less than ϵϵ, in
at least 1−δ1-δ of the cases. Moreover, this result does not depend on the probability distribution.
Proposition 7.4.
Given Assumption 7.3, if a hypothesis is consistent with at least
ϵ(ln|H|+ln1δ)1ϵ(ln|$\mathscr{H}$|+ln1δ)

training examples, it has error at most ϵϵ, at least 1−δ1-δ of the time

Proof.
Suppose ϵ>0ϵ>0 and δ>0δ>0 are given. Partition the hypothesis space H$\mathscr{H}$ into
I0=$\mathscr{H}$0= h∈H:error(I,h)≤ϵ}{h∈$\mathscr{H}$:error(I,h)≤ϵ}

I1=$\mathscr{H}$1= h∈H:error(I,h)>ϵ}.{h∈$\mathscr{H}$:error(I,h)>ϵ}.

We want to guarantee that the learner does not choose an element of H1$\mathscr{H}$1 in more than δδ of the cases.
Suppose h∈H1h∈$\mathscr{H}$1, then
(h iswrong for a single example)≥ϵP(h is wrong for a single example)≥ϵ

(h is correct for a single example)≤1−ϵP(h is correct for a single example)≤1-ϵ

(h is correct for m random examples)≤(1−ϵ)m.P(h is correct for m random examples)≤(1-ϵ)m.

Therefore,
contains a hypothesis that is correct for m random examples)P($\mathscr{H}$1 contains a hypothesis that is correct
n random examples)

≤||H1||(1−ϵ)m ≤|$\mathscr{H}$1|(1-ϵ)m

≤||H||(1−ϵ)m ≤|$\mathscr{H}$|(1-ϵ)m

≤|H|e−ϵm ≤|$\mathscr{H}$|e-ϵm

using the inequality (1−ϵ)≤e−ϵ(1-ϵ)≤e-ϵ if 0≤ϵ≤10≤ϵ≤1.
If we ensure that |H|e−ϵm≤δ|ℋ|e-ϵm≤δ, we guarantee that H1ℋ1 does not contain a hypothesis that is correct for mm examples in more than δδ of the cases. So H0H0 contains all of the correct hypotheses in all but δδ of the cases.
Solving for mm gives

ı≥1ϵ(ln|H|+ln1δ)m≥1ϵ(ln|ℋ|+ln1δ)

which proves the proposition. ■
The number of examples required to guarantee this error bound is called the **sample complexity**. The number of examples required according to this proposition is a function of ϵϵ, δδ, and the size of the hypothesis space.

Sample complexity

The **sample complexity** of a machine learning algorithm represents the number of training-samples that it needs in order to successfully learn a target function.

More precisely, the sample complexity is the number of training-samples that we need to supply to the algorithm, so that the function returned by the algorithm is within an arbitrarily small error of the best possible function, with probability arbitrarily close to 1.

There are two variants of sample complexity:

- The weak variant fixes a particular input-output distribution;
- The strong variant takes the worst-case sample complexity over all input-output distributions.

The No free lunch theorem, discussed below, proves that, in general, the strong sample complexity is infinite, i.e. that there is no algorithm that can learn the globally-optimal target function using a finite number of training samples.

However, if we are only interested in a particular class of target functions (e.g, only linear functions) then the sample complexity is finite, and it depends linearly on the VC dimension on the class of target functions.

Sample Complexity for *Finite* Hypothesis Spaces

- Given any *consistent* learner, the number of examples sufficient to assure that any hypothesis will be probably (with probability *(1- δ)*) approximately (within error ε) correct is $m = 1/\varepsilon\ (\ln|H| + \ln(1/\delta))$
- If the learner is *not consistent*, $m = 1/2\varepsilon^2\ (\ln|H| + \ln(1/\delta))$
- Conjunctions of Boolean Literals are also PAC-Learnable and $m = 1/\varepsilon\ (n.\ln 3 + \ln(1/\delta))$
- k-term DNF expressions are not PAC learnable because even though they have polynomial sample complexity, their computational complexity is not polynomial.

Surprisingly, however, k-term CNF is PAC learnable.

Sample Complexity for Infinite Hypothesis Spaces I: VC-Dimension

- The PAC Learning framework has 2 disadvantages:

It can lead to weak bounds

Sample Complexity bound cannot be established for infinite hypothesis spaces

- We introduce new ideas for dealing with these problems:

Definition: A set of instances S is shattered by hypothesis space H iff for every dichotomy of S there exists some hypothesis in H consistent with this dichotomy.

Definition: The Vapnik-Chervonenkisdimension, VC(H), of hypothesis space H defined over instance space X is the size of the largest finite subset of X shattered by H. If arbitrarily large finite sets of X can be shattered by H, then VC(H)=∞

- ***Upper-Bound*** on sample complexity, using the ***VC-Dimension***: $m \geq 1/\varepsilon\ (4\log_2(2/\delta) + 8VC(H)\log_2(13/\varepsilon))$
- ***Lower Bound*** on sample complexity, using the ***VC-Dimension***:

Consider any concept class ***C*** such that $VC(C) \geq 2$, any learner ***L***, and any $0 < \varepsilon < 1/8$, and $0 < \delta < 1/100$. Then there exists a distribution ***D*** and target concept in ***C*** such that if ***L*** observes fewer examples than $\max[1/\varepsilon \log(1/\delta), (VC(C)-1)/(32\varepsilon)]$ then with probability at least δ, ***L*** outputs a hypothesis ***h*** having $error_D(h) > \varepsilon$.

Instance-based learning

The Machine Learning systems which are categorized as **instance-based learning** are the systems that learn the training examples by heart and then generalizes to new instances based on some similarity measure. It is called instance-based because it builds the hypotheses from the training instances. It is also known as **memory-based learning** or **lazy-learning.** The time complexity of this algorithm depends upon the size of training data. The worst-case time complexity of this algorithm is **O (n)**, where n is the number of training instances.

For example, If we were to create a spam filter with an instance-based learning algorithm, instead of just flagging emails that are already marked as spam emails, our spam filter would be programmed to also flag emails that are very similar to them. This requires a measure of resemblance between two emails. A similarity measure between two emails could be the same sender or the repetitive use of the same keywords or something else.

Advantages:

1. Instead of estimating for the entire instance set, local approximations can be made to the target function.
2. This algorithm can adapt to new data easily, one which is collected as we go .

Disadvantages:

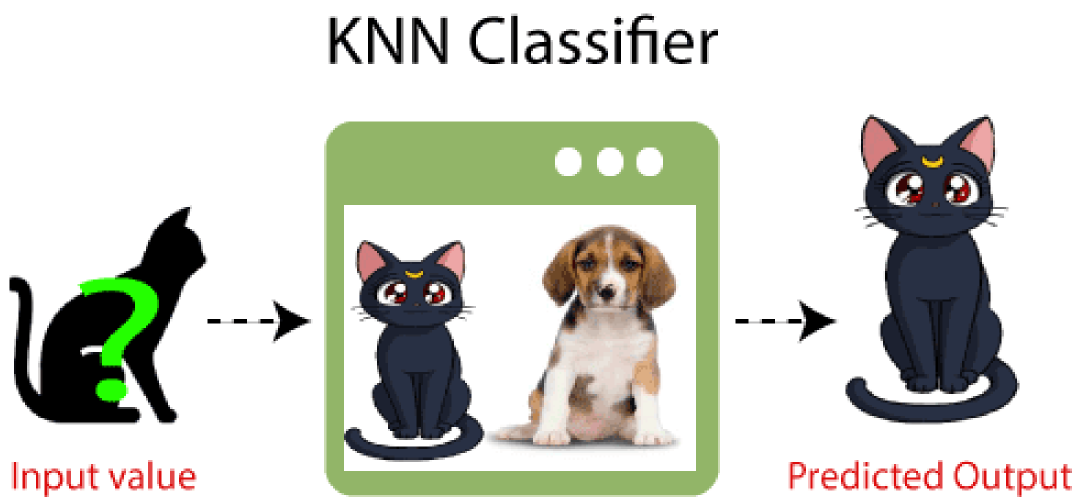
1. Classification costs are high
2. Large amount of memory required to store the data, and each query involves starting the identification of a local model from scratch.

Some of the instance-based learning algorithms are :

1. K Nearest Neighbor (KNN)
2. Self-Organizing Map (SOM)
3. Learning Vector Quantization (LVQ)
4. Locally Weighted Learning (LWL)

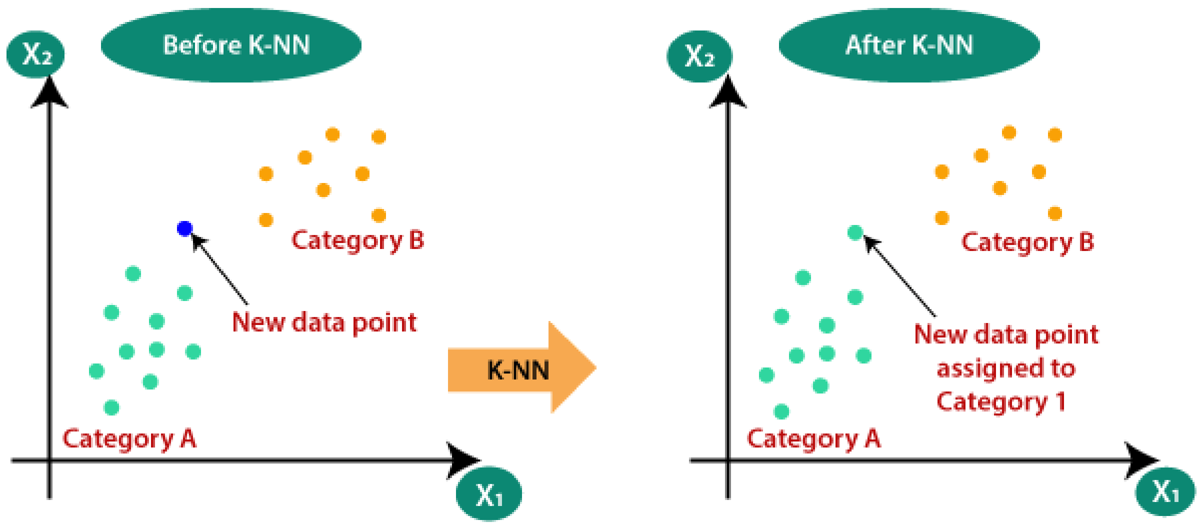
K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



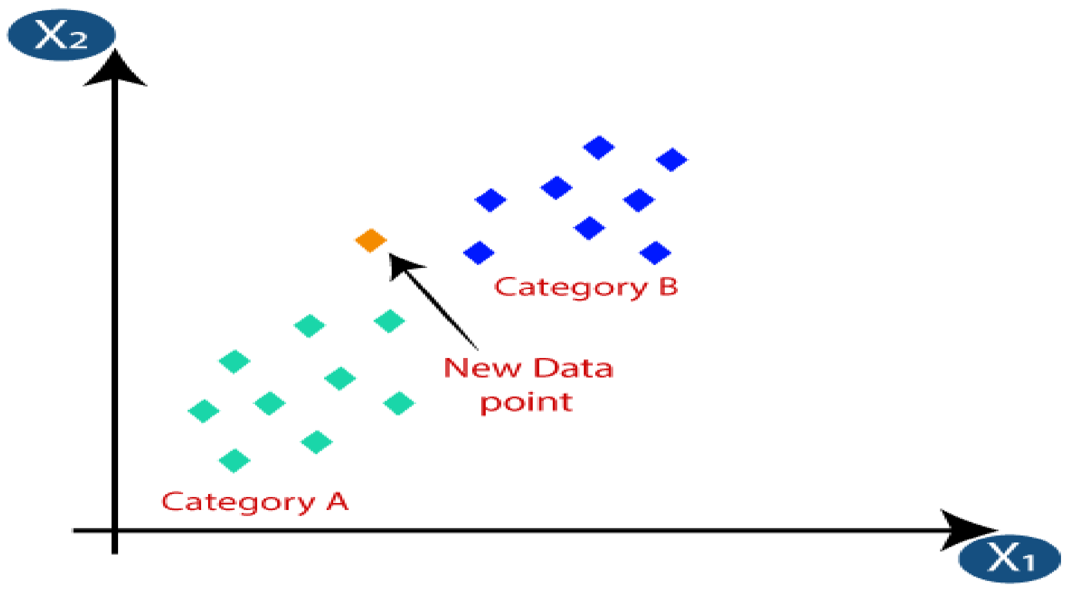
How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

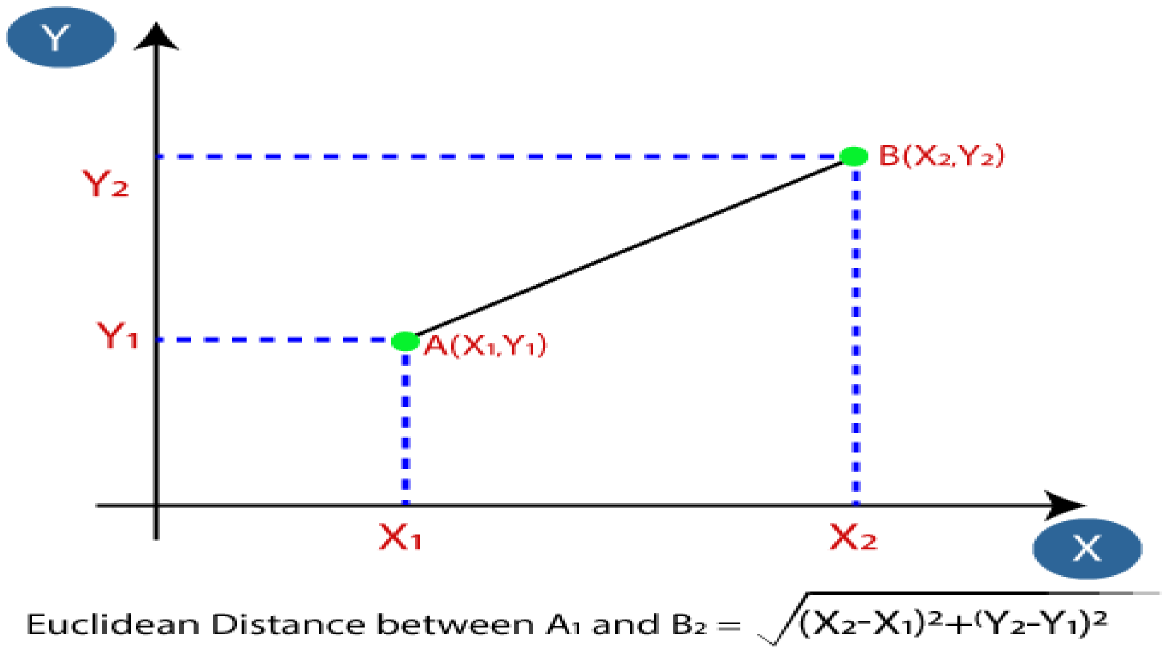
- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

ADVERTISEMENT BY ADRECOVER

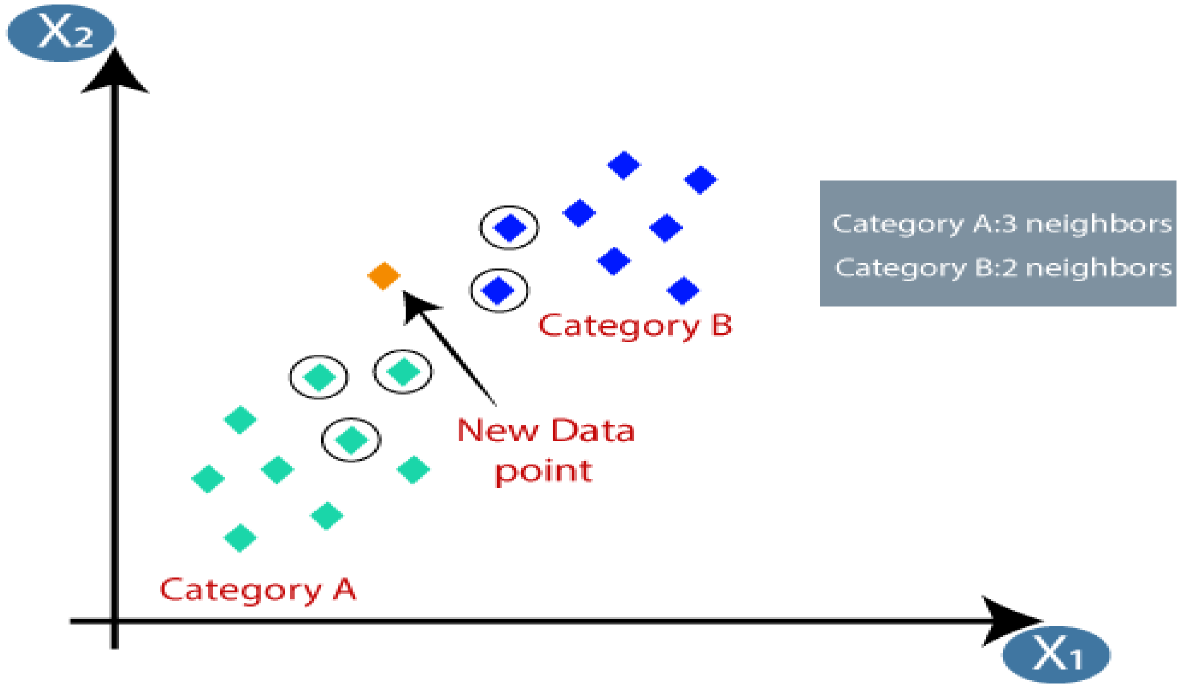
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

ADVERTISEMENT BY ADRECOVER

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Locally Weighted Regression

Model-based methods, such as neural networks and the mixture of Gaussians, use the data to build a parameterized model. After training, the model is used for predictions and the data are generally discarded. In contrast, ``memory-based" methods are non-parametric approaches that explicitly retain the training data, and use it each time a prediction needs to be made. Locally weighted regression (LWR) is a memory-based method that performs a regression around a point of interest using only training data that are ``local" to that point. One recent study demonstrated that LWR was suitable for real-time control by constructing an LWR-based system that learned a difficult juggling task [Schaal & Atkeson 1994].

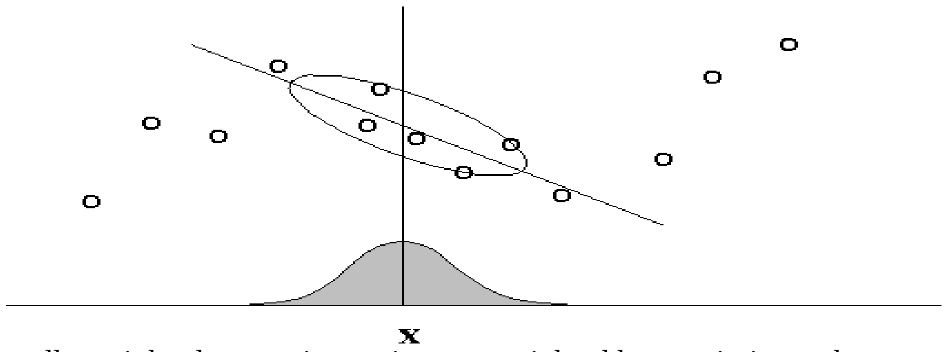


Figure 2: In locally weighted regression, points are weighted by proximity to the current **x** in question using a kernel. A regression is then computed using the weighted points

We consider here a form of locally weighted regression that is a variant of the LOESS model [Cleveland et al. 1988]. The LOESS model performs a linear regression on points in the data set, weighted by a kernel centered at **x** (see Figure 2). The kernel shape is a design parameter for which there are many possible choices: the original LOESS model uses a ``tricubic" kernel; in our experiments we have used a Gaussian

$$h_i(x) \equiv h(x - x_i) = \exp(-k(x - x_i)^2),$$

where **k** is a smoothing parameter. In Section 4.1 we will describe several methods for automatically setting **k**.

For brevity, we will drop the argument **x** for $h_i(x)$, and define $n = \sum_i h_i$. We can then write the estimated means and covariances as:

$$\mu_x = \frac{\sum_i h_i x_i}{n},\ \sigma_x^2 = \frac{\sum_i h_i (x_i - \mu_x)^2}{n},\ \sigma_{xy} = \frac{\sum_i h_i (x_i - \mu_x)(y_i - \mu_y)}{n}$$

$$\mu_y = \frac{\sum_i h_i y_i}{n},\ \sigma_y^2 = \frac{\sum_i h_i (y_i - \mu_y)^2}{n},\ \sigma_{y|x}^2 = \sigma_y^2 - \frac{\sigma_{xy}^2}{\sigma_x^2}.$$

We use the data covariances to express the conditional expectations and their estimated variances:

$$\hat{y} = \mu_y + \frac{\sigma_{xy}}{\sigma_x^2}(x - \mu_x), \qquad \sigma_{\hat{y}}^2 = \frac{\sigma_{y|x}^2}{n^2}\left(\sum_i h_i^2 + \frac{(x-\mu_x)^2}{\sigma_x^2}\sum_i h_i^2 \frac{(x_i - \mu_x)^2}{\sigma_x^2}\right) \qquad (10)$$

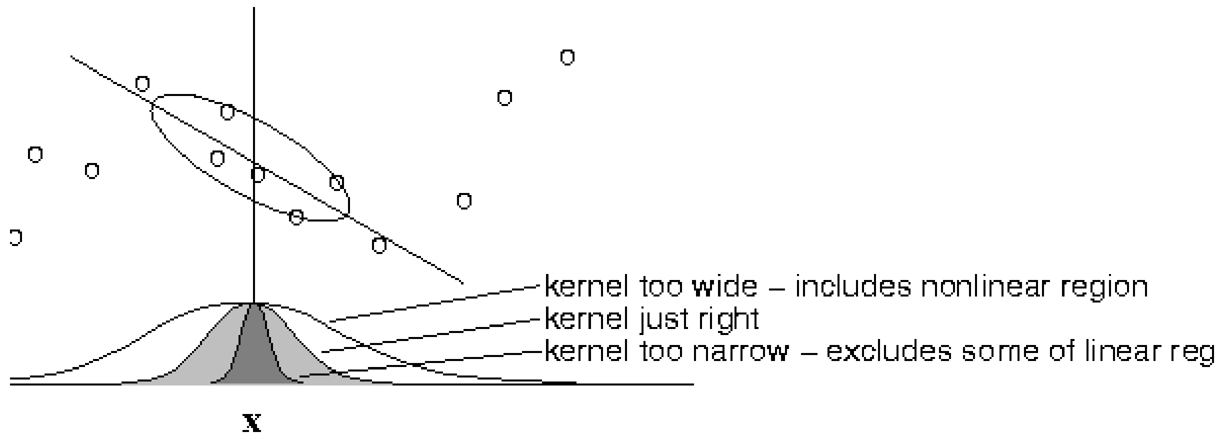

kernel too wide – includes nonlinear region
kernel just right
kernel too narrow – excludes some of linear reg
x

Figure 3: The estimator variance is minimized when the kernel includes as many training points as

can be accommodated by the model. Here the linear LOESS model is shown. Too large a kernel includes points that degrade the fit; too small a kernel neglects points that increase confidence in the fit

Radial Basis Function Kernel

Radial Basis Kernel is a kernel function that is used in machine learning to find a non-linear classifier or regression line.

What is Kernel Function?
Kernel Function is used to transform n-dimensional input to m-dimensional input, where m is much higher than n then find the dot product in higher dimensional efficiently. The main idea to use kernel is: A linear classifier or regression curve in higher dimensions becomes a Non-linear classifier or regression curve in lower dimensions.

Mathematical Definition of Radial Basis Kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Radial Basis Kernel

where *x, x'* are vector point in any fixed dimensional space.
But if we expand the above exponential expression, It will go upto infinite power of *x* and *x'*, as expansion of e^x contains infinite terms upto infinite power of *x* hence it involves terms upto infinite powers in infinite dimension.
If we apply any of the algorithms like perceptron Algorithm or linear regression on this kernel, actually we would be applying our algorithm to new infinite-dimensional datapoint we have created. Hence it will give a hyperplane in infinite dimensions, which will give a very strong non-linear classifier or regression curve after returning to our original dimensions.

$$a_1 X^{inf} + a_2 X^{inf-1} + a_3 X^{inf-2} + \ldots + a_{inf} X + C$$

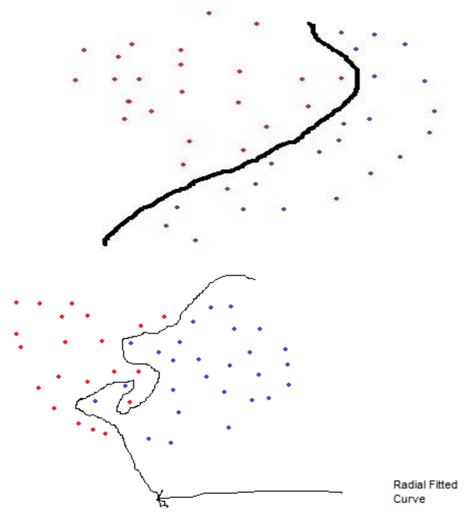
polynomial of infinite power

So, Although we are applying linear classifier/regression it will give a non-linear classifier or regression line, that will be a polynomial of infinite power. And being a polynomial of infinite power, Radial Basis kernel is a very powerful kernel, which can give a curve fitting any complex dataset.

Why Radial Basis Kernel Is much powerful?

The main motive of the kernel is to do calculations in any d-dimensional space where $d > 1$, so that we can get a quadratic, cubic or any polynomial equation of large degree for our classification/regression line. Since Radial basis kernel uses exponent and as we know the expansion of e^x gives a polynomial equation of infinite power, so using this kernel, we make our regression/classification line infinitely powerful too.

Some Complex Dataset Fitted Using RBF Kernel easily:



Case Based Reasoning (CBR) Classifier

As we know **Nearest Neighbour classifiers** stores training tuples as points in Euclidean space. But **Case-Based Reasoning classifiers (CBR)** use a database of problem solutions to solve new problems. It stores the tuples or cases for problem-solving as complex symbolic descriptions.

How CBR works?

When a new case arrises to classify, a Case-based Reasoner(CBR) will first check if an identical training case exists. If one is found, then the accompanying solution to that case is returned. If no identical case is found, then the CBR will search for training cases having components that are similar to those of the new case. Conceptually, these training cases may be considered as neighbours of the new case. If cases are represented as graphs, this involves searching for subgraphs that are similar to subgraphs within the new case. The CBR tries to combine the solutions of the neighbouring training cases to propose a solution for the new case. If compatibilities arise with the individual solutions, then backtracking to search for other solutions may be necessary. The CBR may employ background knowledge and problem-solving strategies to propose a feasible solution.

Applications of CBR includes:

1. Problem resolution for customer service help desks, where cases describe product-related diagnostic problems.
2. It is also applied to areas such as engineering and law, where cases are either technical designs or legal rulings, respectively.
3. Medical educations, where patient case histories and treatments are used to help diagnose and treat new patients.

Challenges with CBR

- Finding a good similarity metric (eg for matching subgraphs) and suitable methods for combining solutions.
- Selecting salient features for indexing training cases and the development of efficient indexing techniques.

CBR becomes more intelligent as the number of the trade-off between accuracy and efficiency evolves as the number of stored cases becomes very large. But after a certain point, the system's efficiency will suffer as the time required to search for and process relevant cases increases.

Remarks on lazy and eager learning.

Lazy learner:

1. Just store Data set **without** learning from it
2. Start classifying data when it receive **Test data**
3. So it takes less time learning and more time classifying data

Eager learner:

1. When it receive data set it starts classifying (learning)
2. Then it does not wait for test data to learn
3. So it takes long time learning and less time classifying data

Hint : In supervised learning: Some examples are :
Lazy : K - Nearest Neighbour, Case - Based Reasoning
Eager : Decision Tree, Naive Bayes, Artificial Neural Networks