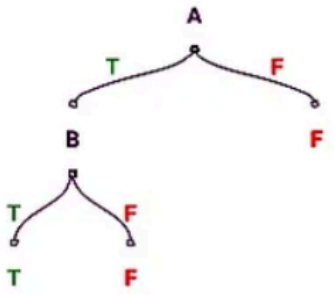
1.Decision tree:

Lets see decision tree with this simple example, It is normal "AND' operation problem, where 'A', 'B' are features and "A and B" are corresponding labels.

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T



Source Hackerearth

If A=F then result=F

If A=T and B=T, then result=T

If A=T and B=F, then result = F

Entropy

Entropy is amount of information is needed to accurately describe the some sample.

$$Entropy = -\sum_{i=1}^{n} p_i * \log(p_i)$$

Gini index / Gini impurity

Gini index is measure of inequality in sample.

$$Gini\,index = 1 - \sum_{i=1}^{n} p_i^2$$

Decision Tree Algorithm

Decision tree algorithm is a tree where nodes represents features(attributes), branch represents decision(rule) and leaf nodes represents outcomes(discrete and continuous).

2. KNN

Working of K-Nearest Neighbour

KNN algorithm stores all available cases and classifies new data based on the majority class of its nearest neighbors. Value of **K** in KNN refers to the number of nearest neighbors to consider when performing classification.

K parameter is critical because:

- If **K** is too small, the model may be sensitive to noise in the dataset.
- If **K** is too large, the classification might be too generalized, and nuances in the data may be overlooked.

Distance between data points is measured using a distance metric, such as **Euclidean distance**, to find the nearest neighbors.

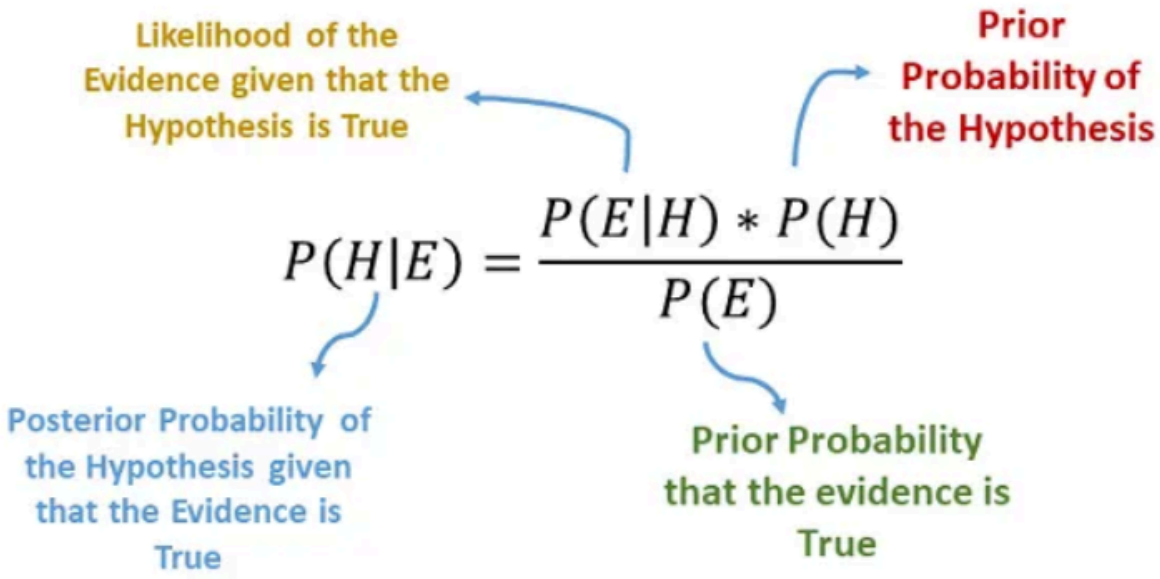
people Angelina will lie whose k factor is 3 and age is 5. So we have to find out the distance using **Euclidean distance formula**:

$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ to find the distance between any two points.

To calculate the distance between **Angelina** and other individuals in the dataset:

$$d = \sqrt{(age_2 - age_1)^2 + (gender_2 - gender_1)^2}$$

3.Naive Bayes



Using the chain rule, the likelihood P(X | y) can be decomposed as:

$$P(X|y) = P(x_1, x_2, \ldots, x_n|y)$$

$$= P(x_1|x_2, \ldots, x_n, y) * P(x_2|x_3, \ldots, x_n, y) \ldots P(x_n|y)$$

but because of the Naive's conditional independence assumption, the conditional probabilities are independent of each other.

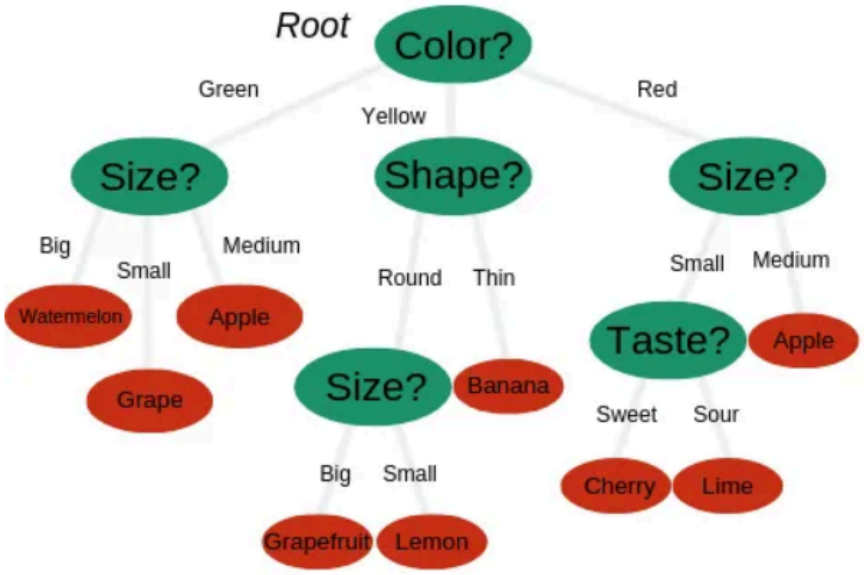
$$P(X|y) = P(x_1|y) * P(x_2|y) \ldots . P(x_n|y)$$

Thus, by conditional independence, we have:

$$P(y|X) = \frac{P(x_1|y) * P(x_2|y) \ldots . P(x_n|y) * P(y)}{P(x_1) * P(x_2) \ldots . P(x_n)}$$

4.Random Forest

The Random Forest Algorithm is composed of different decision trees, each with the same nodes, but using different data that leads to different leaves. It merges the decisions of multiple decision trees in order to find an answer, which represents the average of all these decision trees.



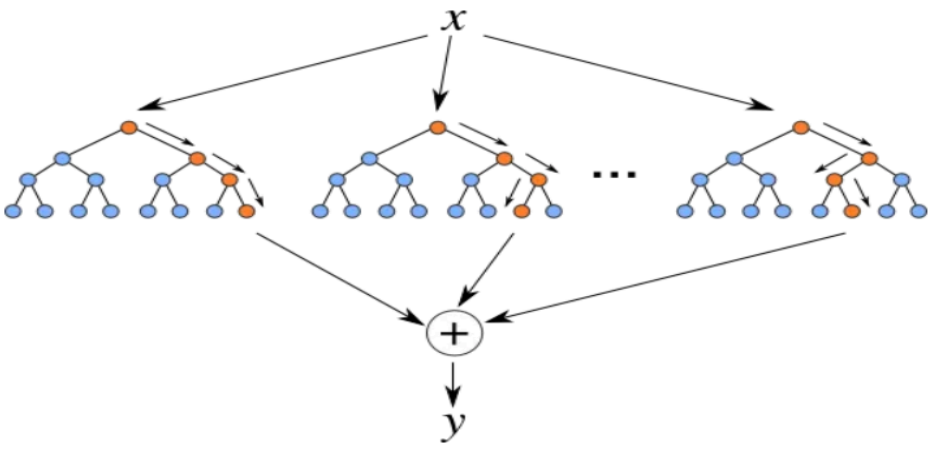
The random forest algorithm is a supervised learning model; it uses labeled data to "learn" how to classify unlabeled data. This is the opposite of the K-means Cluster algorithm, which we learned in a past article was an unsupervised learning model. The Random Forest Algorithm is used to solve both regression and classification problems, making it a diverse model that is widely used by engineers.

Pros:

- Used for regression and classification problems, making it a diverse model.
- Prevents overfitting of data.
- Fast to train with test data.

Cons:

- Slow in creating predictions once model is made.
- Must beware of outliers and holes in the data.



Classification

Before applying the Random Forest Algorithm on these results you will need to perform something called one-hot encoding. This entails assigning a number to a categorical variable in order to apply mathematics to the problem.

Label Encoding

Product Purchased	Categorical #	Purchase Price
Toilet Paper	1	$4.00
Shampoo	2	$6.50
Conditioner	3	$8.50

One Hot Encoding

Toilet Paper	Shampoo	Cleanser	Price
1	0	0	$4.00
0	1	0	$6.50
0	0	1	$8.50

After the data is one-hot encoded, the mathematics can be applied and the Random Forest Algorithm can come to a conclusion. If the algorithm concludes that most people in this target market are not potential customers, it may be a good idea for the company to rethink their product with these types of people in mind.

Before applying the Random Forest Algorithm on these results you will need to perform something called one-hot encoding. This entails assigning a number to a categorical variable in order to apply mathematics to the problem.

Label Encoding

Product Purchased	Categorical #	Purchase Price
Toilet Paper	1	$4.00
Shampoo	2	$6.50
Conditioner	3	$8.50

One Hot Encoding

Toilet Paper	Shampoo	Cleanser	Price
1	0	0	$4.00
0	1	0	$6.50
0	0	1	$8.50

After the data is one-hot encoded, the mathematics can be applied and the Random Forest Algorithm can come to a conclusion. If the algorithm concludes that most people in this target market are not potential customers, it may be a good idea for the company to rethink their product with these types of people in mind.

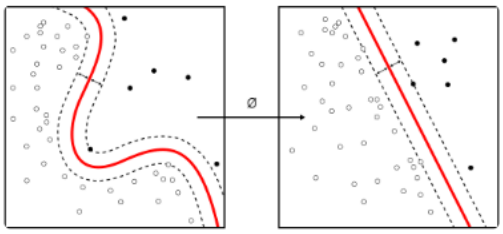
5.SVM

What is Support Vector Machine (SVM)?

A **Support Vector Machine** or **SVM** is a machine learning algorithm that looks at data and sorts it into one of two categories.

Support Vector Machine is a supervised and linear Machine Learning algorithm most commonly used for solving classification problems and is also referred to as Support Vector Classification.

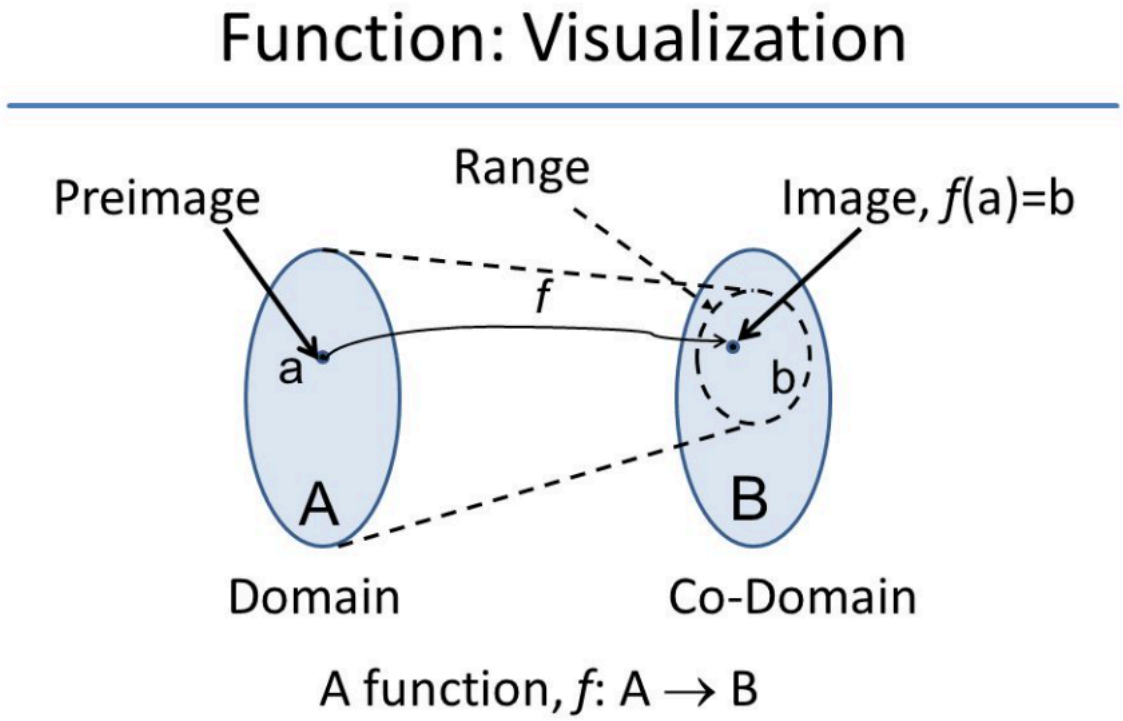
There is also a subset of SVM called SVR which stands for Support Vector Regression which uses the same principles to solve regression problems.

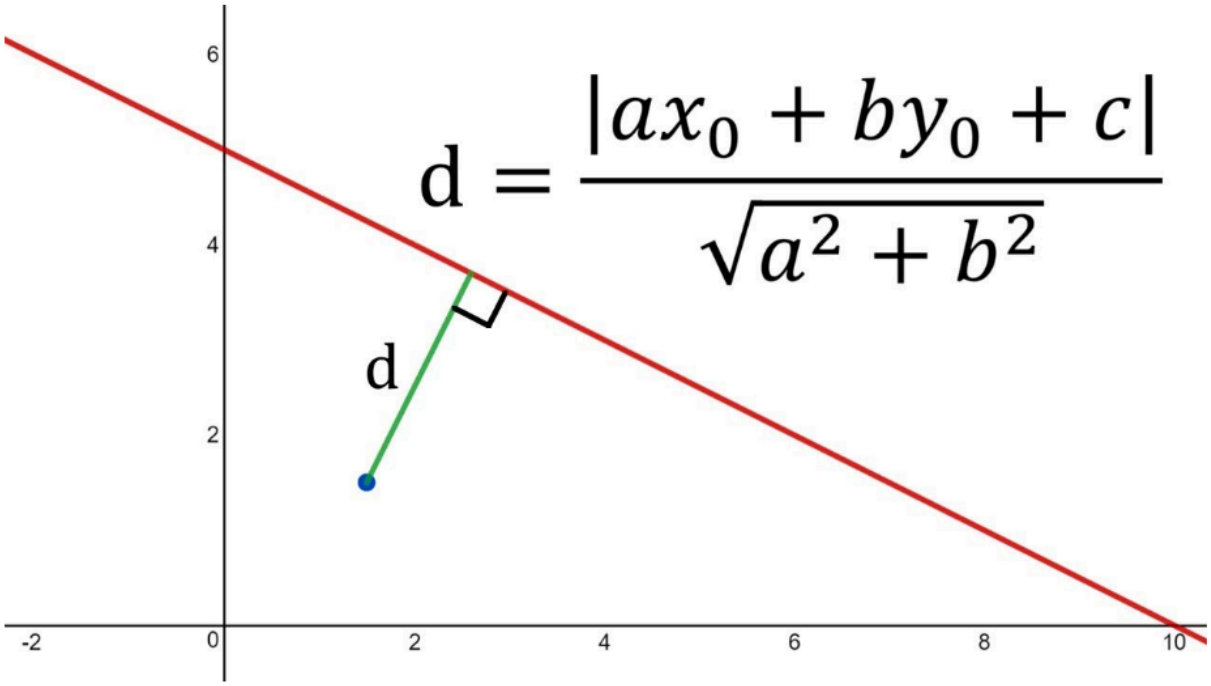


> SVM is most commonly used and effective because of the use of the Kernel method which basically helps in solving the non-linearity of the equation in a very easy manner.

Let's assume we have some data where we (algorithm of SVM) are asked to differentiate between the males and females based on first studying the characteristics of both the genders and then accurately label the unseen data if someone is a male or a female.

In this example, the characteristics which will help to differentiate the gender are basically called features in machine learning.





The distance of any line, ax + by + c = 0 from a given point say, (x_0 , y_0) is given by d.

Similarly, the distance of a hyperplane equation: $w^T\Phi(x) + b = 0$ from a given point vector $\Phi(x_0)$ can be easily written as :

$$d_H(\phi(x_0)) = \frac{|w^T(\phi(x_0)) + b|}{||w||_2}$$

> The product of a predicted and actual label would be greater than 0 (zero) on correct prediction, otherwise less than zero.

$$y_n\left[w^T\phi(x) + b\right] = \begin{cases} \geq 0 \; if \; correct \\ < 0 \; if \; incorrect \end{cases}$$

> For perfectly separable datasets, the optimal hyperplane classifies all the points correctly, further substituting the optimal values in the weight equation.

$$w^* = arg_w max \left[min_n \frac{|w^T(\phi(x_n)) + b|}{||w||_2} \right] = arg_w max \left[min_n \frac{y_n|w^T(\phi(x_n)) + b|}{||w||_2} \right] \; \because \; perfect \; seperation$$

> **Here**:
>
> arg max is an abbreviation for arguments of the maxima which are basically the points of the domain of a function at which function values are maximized.

Primal Form of SVM (Perfect Separation)

The above optimization problem is the Primal formulation since the problem statement has original variables.

So,

$$Primal \; Form \; when \; data \; is \; perfectly \; speerable \; and \; no \; misclassification$$
$$min_w \frac{1}{2}||w||_2 \quad ; s.t. \; y_n\left[w^T\phi(x_n) + b\right] \geq 1, \; \forall n$$

Solution for Primal Form (Non – perfect separation):

$$min_{w,b,\{\beta_n\}} \frac{1}{2}||w||_2 + C\sum_n \beta_n$$
$$s.t.\ y_n\left[w^T\phi(x_n) + b\right] \geq 1 - \beta_n\,;\ \forall n$$
$$\beta_n \geq 0\,,\ \forall n$$

Since we have Φ, which has a complex notation. we would be rewriting the equation.

```
Concept is basically to get rid of Φ and hence rewrite Primal formulation in Dual Formulatio
```

In other words :

> **Dual Form:** rewrites the same problem using a different set of variables. So the alternate formulation will help in eliminating the dependence on Φ and reducing the effect will be done with Kernelization.

Lagrange Multiplier Method: It is a strategy to find the local minima or maxima of a function subject to the condition that one or more equations have to be satisfied exactly by the chosen values of variables.

The kernel by definition avoids the explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary For all x and x' in the input space Φ certain functions k(x,x') can be expressed as an inner product in another space Ψ. The function

$$k : \phi X \ \phi \ \rightarrow \ R$$

is referred to as a ***Kernel***. In short, for machine learning kernel is defined as written in the form of a "feature map"

$$\omega : \phi \ \rightarrow \psi$$

which satisfies

$$k(x, x') \ = \ (\omega(x) \ , \ \omega(x'))\psi$$

For a better understanding of kernels, one can refer to this link of quora

The kernel has two properties:

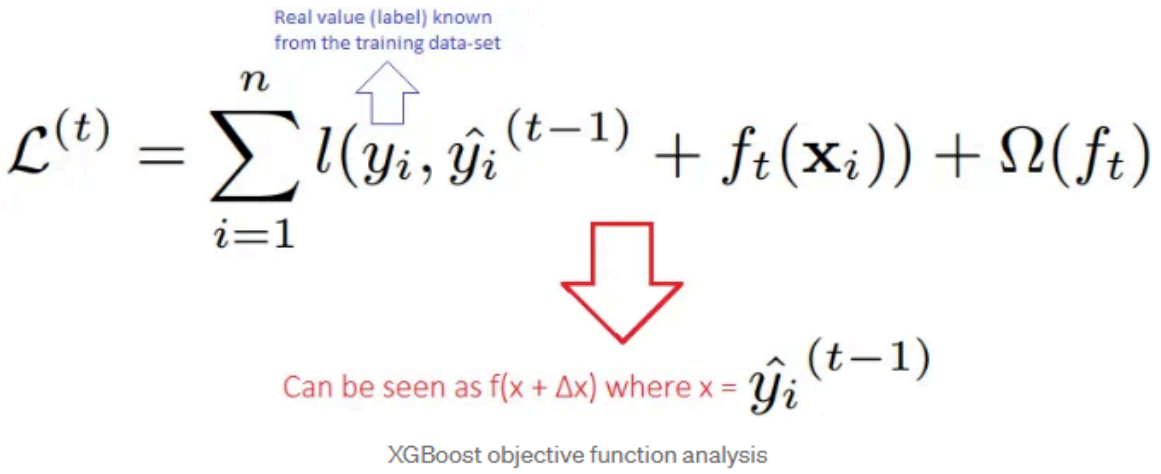
- It is symmetric in nature $k(x_n , x_m) = k(x_m , x_n)$
- It is Positive semi-definite

$$\sum \sum v_m v_n k(x_m , x_n) \ \geq \ 0$$

6.XGBoost

2. XGBoost objective function

The objective function (loss function and regularization) at iteration *t* that we need to minimize is the following:

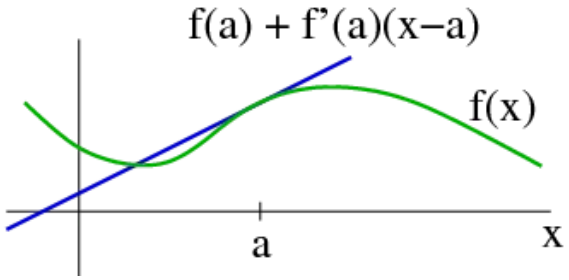


XGBoost objective function analysis

It is easy to see that the XGBoost objective is a function of functions (i.e. *l* is a function of CART learners, a sum of the current and previous additive trees), and as the authors refer in the paper [2] "cannot be optimized using traditional optimization methods in Euclidean space".

3. Taylor's Theorem and Gradient Boosted Trees

From the reference [1] we can see as an example that the best linear approximation for a function *f(x)* at point *a* is:



As we already said, ***a*** is the prediction at step ***(t-1)*** while ***(x-a)*** is the new learner that we need to add in step ***(t)*** in order to greedily minimize the objective.

So, if we decide to take the second-order Taylor approximation, we have:

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2$$

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}[l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

XGBoost objective using second-order Taylor approximation

Where:

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \text{ and } h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

First and second order gradient statistics of the loss function

Finally, if we remove the constant parts, we have the following simplified objective to minimize at step ***t***:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

XGBoost simplified objective

5. Binary classification with log loss optimization

Let's take the case of binary classification and log loss objective function:

$$y \ln(p) + (1-y) \ln(1-p) \text{ where } p = \frac{1}{(1+e^{-x})}$$

Binary classification with Cross Entropy loss function

,where y is the real label in $\{0,1\}$ and p is the probability score.

Note that p (score or pseudo-probability) is calculated after applying the famous sigmoid function into the *output of the* ***GBT model*** x.
The output x of the model is the sum across the the CART tree learners.

So, in order to minimize the log loss objective function we need to find its 1st and 2nd derivatives (gradient and hessian) with respect to x.
In this stats.stackexchange post you can find that gradient = $(p-y)$ and hessian = $p*(1-p)$.

Summarizing the GBT model which — remember — is a sum of CART (tree) learners will try to minimize the log loss objective and the scores at leaves which are actually the weights have a meaning as a sum across all the trees of the model and are always adjusted in order to minimize the loss. This is why we need to apply the sigmoid function in the output of GBT models in case of binary classification probability scoring.

4. How to build the next learner

Being at iteration t we need to build a learner that achieves the maximum possible reduction of loss, the following questions arrive:

- Is it possible to find the optimal next learner?
- Is there any way to calculate the gain (loss reduction) after adding a specific learner?

The good news is that there is a way to "measure the quality of a tree structure q" as the authors refer, and the scoring function is the following (see the correspondence to the "simple quadratic function solution" above):

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T.$$

instances mapped to leaf$_j$

The tree learner structure q scoring function

While the bad news is that it is impossible in terms of required calculations to "enumerate all the possible tree structures q" and thus find the one with maximum loss reduction.

7.SMOTE

Synthetic Minority Over-sampling TEchnique, or SMOTE for short, is a preprocessing technique used to address a class imbalance in a dataset.

over-sample the minority class. In other words, we randomly duplicate observations of the minority class. The problem with this approach is that it leads to overfitting because the model learns from the same examples. This is where SMOTE comes in. At a high level, the SMOTE algorithm can be described as follows:

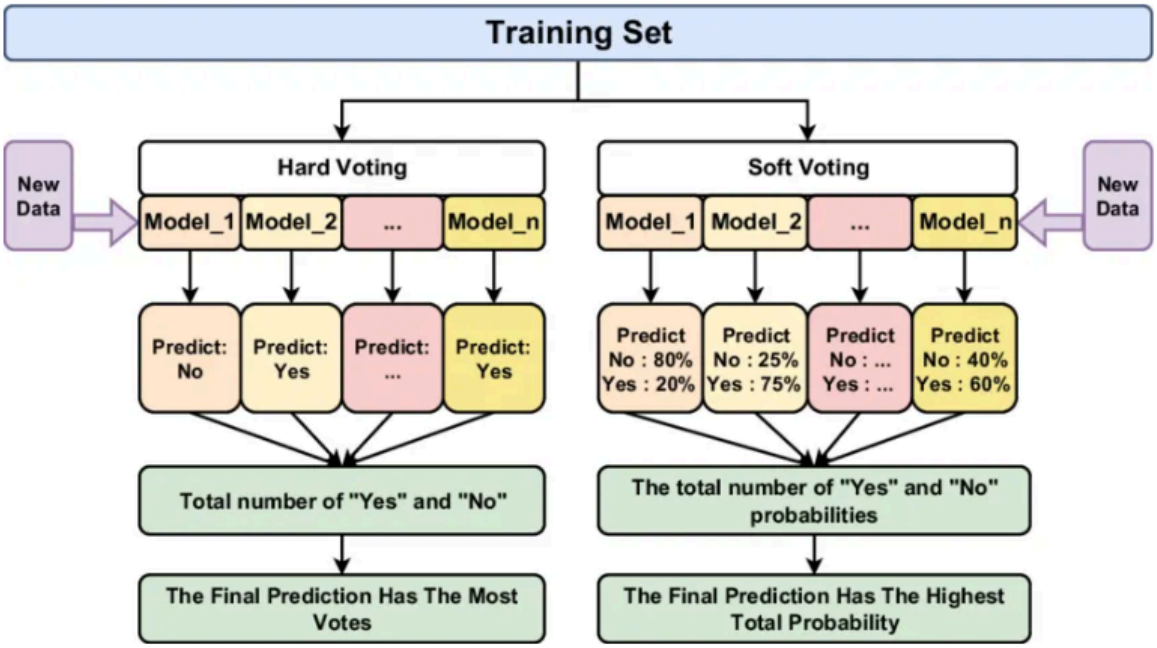
- Take difference between a sample and its nearest neighbour
- Multiply the difference by a random number between 0 and 1
- Add this difference to the sample to generate a new synthetic example in feature space
- Continue on with next nearest neighbour up to user-defined number

Algorithm *SMOTE*(T, N, k)
Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples

(* *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* *)
if $N < 100$
  then Randomize the T minority class samples
    $T = (N/100) * T$
    $N = 100$
endif
$N = (int)(N/100)$ (* *The amount of SMOTE is assumed to be in integral multiples of 100.* *)
k = Number of nearest neighbors
$numattrs$ = Number of attributes
$Sample[\,][\,]$: array for original minority class samples
$newindex$: keeps a count of number of synthetic samples generated, initialized to 0
$Synthetic[\,][\,]$: array for synthetic samples
(* *Compute k nearest neighbors for each minority class sample only.* *)
for $i \leftarrow 1$ **to** T
    Compute k nearest neighbors for i, and save the indices in the $nnarray$
    Populate(N, i, $nnarray$)
endfor

Populate(N, i, $nnarray$) (* *Function to generate the synthetic samples.* *)
while $N \neq 0$
    Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
    for $attr \leftarrow 1$ **to** $numattrs$
      Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
      Compute: gap = random number between 0 and 1
      $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
    endfor
    $newindex{+}{+}$
    $N = N - 1$
endwhile
return (* *End of Populate.* *)
End of Pseudo-Code.

Misc: <u>Voting Classifier :</u>



A Voting Classifier is an ensemble learning technique that combines multiple classifiers and predicts the class based on a voting mechanism. This approach enhances model accuracy, reduces overfitting, and makes predictions more robust.

There are two types of voting classifiers:

1. **Hard Voting**: The final prediction is based on the majority vote of individual classifiers. For instance, if three classifiers predict [Class A, Class B, Class A], the final prediction will be Class A.
2. **Soft Voting**: Instead of counting votes, it averages the predicted probabilities of each classifier and selects the class with the highest probability.

Why Use Voting Classifiers?

Voting classifiers improve performance when individual models have different decision boundaries. Here are some key benefits:

- **Increased Accuracy**: Aggregating diverse models reduces error rates.
- **Reduced Overfitting**: If individual models overfit, combining them helps in generalization.
- **Diversity in Predictions**: Different models capture different aspects of the data, leading to better decision-making.

However, to ensure an effective ensemble:

- The base models should be diverse.
- Each model should perform reasonably well (accuracy above 50%).

Mathematical Intuition Behind Voting Classifiers

Hard Voting

Consider three models M1,M2,M3M_1, M_2, M_3 predicting a binary classification problem (Class 0 or Class 1). The votes are:

M1=0,M2=1,M3=0M_1 = 0, \quad M_2 = 1, \quad M_3 = 0

Since Class 0 has the majority vote, the final prediction is **Class 0**.

Soft Voting

Instead of discrete votes, we take the average probability of each class:

$$P(0) = \frac{P_{M_1}(0) + P_{M_2}(0) + P_{M_3}(0)}{3}$$

$$P(1) = \frac{P_{M_1}(1) + P_{M_2}(1) + P_{M_3}(1)}{3}$$

If P(0)>P(1)P(0) > P(1), predict Class 0, otherwise predict Class 1.

When to Use Voting Classifiers?

Voting Classifiers work best when:

1. **You have diverse models**: Combining different algorithms improves learning.
2. **Models have similar performance**: If one model is much stronger, it may dominate voting.
3. **Reducing Overfitting is important**: Combining models makes predictions more robust.

However, if:

- A single model consistently outperforms others, using an ensemble may not help.
- The dataset is small, the computational cost may not justify ensemble methods.