A Foray Into Machine Learning

Q & A on the main concepts and terminology

Huascar Sanchez
`github.com/hsanchez`

Home Research Lab

June 9, 2020

The views expressed do not necessarily reflect the position of my employer.

Q. What is Machine Learning?

"*Machine Learning (or **ML**) is fitting a function to examples*[1] *and using that function to generalize and make predictions about new examples.*"

Derek Jedamski, GitHub

Machine Learning, by large, falls into three categories:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

And solves two types of problems: Classification and Regression.

[1] An example, or a sample, is a data point that belongs to some data

Q. What is Supervised Learning?

In **Supervised Learning** (or **SL**), you are given a bunch of examples and their labels (e.g., A or B) and the goal is to classify, when you are given a new example, to which label we should assign the new example.

You could think of these labels as the names of the **classes or clusters** to which certain portions of the data belong.

Q. Examples of supervised learning?

Classification (pattern recognition):

Speech recognition, machine translation, ...

Medical diagnosis: often, variables are missing (tests are costly).

Regression (the labels to be predicted are continuous)

Predict the price of a car from its mileage, ...

Kinematics of a robot arm: predict workspace location from angles

Q. Classification vs Regression?

Classification is the task of predicting a discrete class label.

Regression is the task of predicting a continuous quantity.

There's some overlap between classification and regression algorithms; for example

A classification algorithm may predict a continuous value, but the continuous value is in the form of a probability for a class label.

A regression algorithm may predict a discrete value, but the discrete value in the form of an integer quantity.

Q. Example of supervised learning algorithm?

Support vector machines or **SVM**, is a ML algorithm that takes some data as input and returns as output an optimal separating *hyperplane* between pieces of data; e.g., a plane separating A from B.

- In this Ex, the data sit in some high dimensional space, and the idea is to construct a plane that maximizes the margins[2] between the plane and the data. If a new datum sits closer to one area of the data, say A, then we assign this new datum to A.

(For historical reasons) This algorithm is called **support vector machines** because *the vectors that lie on the margin of the plane* are called the **support vectors**.

[2] distance between points closest to the line and the actual line

Q. What more can you say about SVM?

In summary, SVM is a method for constructing a device to discriminate. If we're having a supervised learning problem then this method gives me an optimal form of discrimination (i.e., optimal separating hyperplane[3]).

SVM works on both linearly and non-linearly separable data. In latter case, it use the kernel trick[4] to convert these data to linearly separable data in a higher dimension. **This transformation is called kernel**.

[3] A hyperplane in an n-dimensional Euclidean space is a flat, $n-1$ dimensional subset of that space that divides the space into two disconnected parts.

[4] adds one more dimension (called z-axis) – governed by the constraint $z = x^2 + y^2$ and z is the squared distance of the points from origin.

Q. Advantages and Disadvantages SVM?

Advantages

SVM Classifiers offer **good accuracy and perform faster prediction** compared to Naive Bayes algorithm. They also **use less memory** because they use a subset of training points in the decision phase. SVM works well with a clear margin of separation and with high dimensional spaces.

Disadvantages

SVM is **not suitable for large data sets** because of its high training time and it also takes more time in training compared to Naive Bayes. It **works poorly with overlapping classes** and is also **sensitive** to the type of **kernel** used.

Q. SVM hyperparameters?

SVM has a few hyperparameters [5]; however, its most popular include:

1. Kernel parameter (data transformation; kernel trick).
2. C regularization parameter (misclassification penalty).
3. γ (*Gamma*) parameter (spread of kernel/decision boundary).

[5] **Hyperparameters** are the properties that govern the entire ML training process. They are external to the model and whose values cannot be estimated from data. We use them to help estimate model parameters.

Q. SVM's kernel hyperparameter?

Kernel is a transformation that "transforms" a given data input into a desired form; e.g., from non-linearly separable data to linearly separable data in a higher dimension.

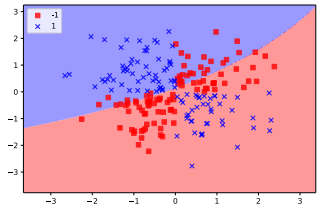
There are various types of kernels such as linear, polynomial, and *radial basis function* (RBF). Polynomial and RBF[6] are useful for non-linear hyperplanes.

[6] Unlike the polynomial kernel which looks at d extra dimensions, RBF expands into an infinite number of dimensions; enabling the inner product of data points that have an infinite number of dimensions.

Q. SVM's C (regularization) hyperparameter?

C is the penalty for misclassifying data points:

- When *C* is small, the classifier is okay with misclassifying data points; if it's too small it can lead to underfitting (high bias, low variance).

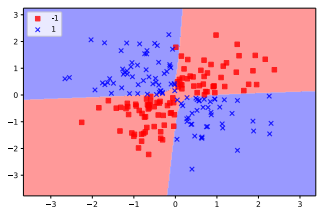


Q. SVM's C (regularization) hyperparameter?

C is the penalty for misclassifying data points:

- When *C* is small, the classifier is okay with misclassifying data points; if it's too small it can lead to underfitting (high bias, low variance).

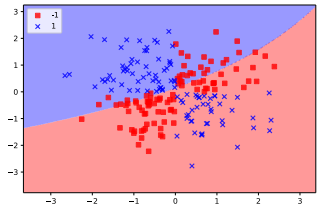
- When *C* is large, the classifier is heavily penalized for misclassifying data points and therefore it bends over backwards avoiding any misclassified data points; if it's too large it can lead to overfitting (low bias, high variance).



Q. SVM's γ hyperparameter (of Radial Basis Function (RBF) Kernel)?

γ (*Gamma*) is the "spread" of the RBF kernel & hence the decision region.

- When γ is low, the "curve" of the decision boundary is very low (like an arch) and thus the decision region is very broad.

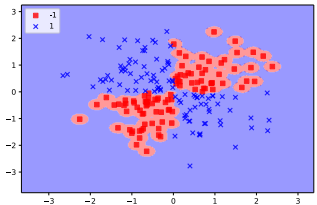


Q. SVM's γ hyperparameter (of Radial Basis Function (RBF) Kernel)?

γ (*Gamma*) is the "spread" of the RBF kernel & hence the decision region.

- When γ is low, the "curve" of the decision boundary is very low (like an arch) and thus the decision region is very broad.

- When γ is high, the "curve" of the decision boundary is high; meaning the decision boundary mostly depends on individual data points, creating islands of decision-boundaries around data points (may overfit).



Q. What is Unsupervised Learning?

In **Unsupervised Learning** (or **UL**), you are given a bunch of data and you are not told they fall naturally into clusters, and also you are not told what these clusters are.

The goal is identify the clusters within data, how many clusters there are, and then be able to assign new things to these different clusters.

Q. Examples of unsupervised learning?

Learning associations

- Basket analysis. Let $\Pr(Y \mid X)$ the probability that a customer who buys X also buys Y – estimated from past purchases. If $\Pr(Y \mid X)$ is 0.7 then associate X to Y ($X \rightarrow Y$); meaning when someone buys X, recommend Y.

Clustering (group similar data points).

Density estimation (where are data points likely to lie?)

Feature selection (keep only useful features).

Outlier/novelty detection.

Q. Can you give an example of Unsupervised Learning?

Principal Component Analysis (or **PCA**). In general terms, PCA is about finding the underlying patterns of the data and also gives you a method for data compression.

If you want to *approximate*[7] the whole matrix with few vectors, the best vectors to choose are top PCA vectors (the principal components).

PCA is all about diagonalizing the covariance matrix.

PCA is an exercise in linear algebra on very high dimensional vector spaces.

[7] All your vectors can be written as the sum of a few *w*s.

Q. Unpacking PCA?

Principal Component Analysis (or **PCA**) is a classical UL algorithm.

In **PCA**, the way this works, we construct a *covariance matrix*, and my covariance matrix is just the following object: $C = \sum_j \vec{v}_j \vec{v}_j^{\dagger}$, where $\vec{v}_j^{\,T}$ is the transpose of $\vec{v}_j$.

- In other words, we construct C from the data by taking these vectors $\vec{v}_j$ and multiply them by their transpose $\vec{v}_j^{\dagger}$.

In **PCA**, you diagonalize C and say $C = \sum_k P_k \vec{\omega}_k \vec{\omega}_k^{\dagger}$[8], P_k is piece of the data with size k, and $\vec{\omega}_k$ are the set of vectors you need to find.

If and only if a small set of $P_k >> 0$, then C is effectively low-rank, and the corresponding $\vec{\omega}_k$ are the principal components. In other words, you need find the eigenvectors that have the largest eigenvalues – i.e., your principal components.

[8] C can be decomposed into a product of matrices involving eigenvalues and eigenvectors

Q. When to use PCA?

When to use it

PCA should be used if one to figure out if there are latent features driving the patterns in the data. E.g., The big shots at SRI International.

Dimensionality reduction (and feature selection). E.g., helps you visualize high dimensional data, helps you reduce noise in your data, make other ML algos work better (regression, classification) because fewer inputs.

When not to use it

PCA is not suitable in many cases: For example, if all the components of PCA have quite a high variance, there is no *good* universal stopping rule that allows you to discard some exact *k* Principal Components, meaning no good data compression.

Not good when working with fine-grained classes[9].

[9] hard-to-distinguish object classes

Q. What is Reinforcement Learning?

Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

In reinforcement learning an algorithm learns to do something by being rewarded for successful behavior and/or being punished for unsuccessful behavior. No supervised output but delayed reward. **Ex:** playing chess or a computer game, robot in a maze.

Q. Can you give an example of Reinforcement Learning?

Policy Gradient (PG)

In this method, we have the policy π that has a parameter θ. This π outputs a probability distribution of actions.

Then we must find the best parameters (θ) to maximize (optimize) a score function $J(\theta)$, given the discount factor γ and the reward r.

Main steps:

Measure the quality of a policy with the policy score function

Use policy gradient ascent to find the best param that improves the policy.

Data representation

Q. How do you represent data in ML?

In general, the given data is expressed in a form of a bunch of vectors $\vec{v}_j \in \mathbb{R}^d$ that belong to some high dimensional vector space.

For instance, in image recognition, the vector[10] of an each image is a set of pixels[11] (i.e., a pixelated version of the image).

If you have a notion of distance $\Delta(\vec{v}_i, \vec{v}_j)$, then you can compare which vectors are close to each other in this high dimensional vector space; e.g., the norm $\left\|\vec{v}_i - \vec{v}_j\right\|^2$.

[10] a.k.a., feature vector; it could be numerical (e.g., height of tree) or descriptive (e.g., eye color)

[11] each entry in the vector (e.g., pixel) represents a feature.

Q. What is it important about measuring distances?

Because the shorter the distance between two feature vectors the closer in character are the two samples they represent.

There are many ways to measure distance:

- **Manhattan distance** (L^1 norm). The sum of the absolute values of the different between entries in the vector. (Preferred dist. when dealing with high dimensional data.)
- **Euclidean distance** (L^2 norm). Square the distances between vector entries, sum these and square root.
- **Cosine similarity**. Cosine of the angle between two vectors[12]. Just take the *dot* product of two vectors and divide by the two lengths.

[12] Two vectors might be similar if they are pointing in the same direction even if they are of different lengths.

Q. Any other issues we should care to learn?

The curse of dimensionality. This phenomena involves data in high dimensions.

Ex. Suppose you are working in M dimensions, that is you data has M features. And suppose you have N data points. Having a large number of data points is good, the more the merrier. BUT what about number of features?

Think how these N data points might be distributed over M dimensions. Suppose that the numerical data for each feature is 0 or 1. Therefore, there will be 2^M possible combinations.

If N is less than 2^M then you run the risk of having every data point being on a different corner of the M-dimensional hypercube.

Building a Machine learning model

Q. What does this process looks like?

Explore and clean the data. Explore the data to really understand what those features look like, then we use some of these learnings to clean the data.

Split data into train/validation/test data sets.

Fit an initial model (baseline) and evaluate using 5-fold cross-validation.

Tune hyper-parameters using GridSearchCV to find the best models[13].

Evaluate best models on validation set and select the top model of each algorithm, and we evaluate them against each other on the validation set.

Select and evaluate the final model on the test set. We'll evaluate top model on the test set to get an unbiased view of the model performance on completely unseen data.

[13] those models that beat our baseline

Measuring success

Q. What does it mean to measure success of the model?

So the question is how do we make sure the model is learning the underlying pattern and not just memorizing the examples?

We can do this by splitting our data into three data sets: the training, validation, and testing data sets.

> This part is about making sure the model is learn the underlying pattern (or correlation) and able to make predictions about future examples.

Q. But why do we split the data?

Ans. We do it to make sure[14] our model is learning the underlying pattern and not just memorizing the examples.

We do that by splitting our dataset into three separate segments; training (60%), validation (20%), and testing (20%) data sets.

Training data, or examples that the model will learn those general patterns from.

Validation data, or examples we use to select the best model (optimal algorithm and hyper-parameter settings).

Test data, or examples we use to provide an unbiased evaluation of what the model will look like in its real environment.

[14] We don't know how well the model will generalize because we don't have any additional data to test this.

Q. Why does the evaluation process look like with these datasets?

1. You start train your ML algorithm on the training data, evaluate it using the validation data, then at this point:

   If none of your models are any good based on the performance on the validation set, then you need to revisit the training phase and consider some new variables or new models. (Go back to step 1.)

   If the performance is quite good, then you can select your best model and pass it onto the testing phase. (Go to step 2.)

2. You evaluate the best model on the test set.

   If the performance is what you expect, then that model is ready to go. Otherwise go to step to the drawing board.

Q. What is cross-validation? Holdout test set?

Holdout test set: A generalization of the test set. It's just any data set that was not used in fitting a model (data set aside for evaluating the model's ability to generalize).

K-Fold Cross-Validation: Data is divided into k subsets and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other $k - 1$ subsets are combined to be used to train the model.

Q. 5-Fold Cross-Validation Example?

E.g., 5-fold CV: Given $10,000$ examples, you partition into 5 buckets, each consisting of $2,000$ examples[15].

On trial 1, we set aside the last bucket (holdout test set) and the other 4 buckets are the training set, then compute its predictive performance. On **trial** 2, we set aside the penultimate bucket, and the other 4 buckets are the training set. This will be similar for the other 3 **trials**.

At the end we *average* the performance of model over the many trails.

[15] this is partitioning is done thru sampling without replacement; no single example will appear in two different subsets and all original $10,000$ are still accounted for in these subsets

Q. Establishing our Evaluation framework?

A cohesive framework for evaluating our model, consisting of two components:

Evaluation metrics: how are gauging the accuracy of the model?

Process (how to split the data): how do we leverage a given data set to mitigate the likelihood of overfitting and underfitting.

Evaluation metrics:
There isn’t a one-size-fits-all metric.

Q. What are the evaluation metrics that we'll use?

The metric(s) chosen to evaluate a ML model depends on various factors:

- Is it a regression or a classification task? E.g., MSE vs Accuracy.
- What is the business objective? E.g., precision vs recall.
- What is the distribution of the target variable?

Examples of other metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), R-squared, Confusion Matrix and related metrics (Precision, Recall, Accuracy).

Q. Measuring classifier performance?

Binary classification:

Accuracy[16] in %: *#(classified correctly) / #(total examples)*

Precision: *#(classified correctly as A) / #(total examples predicted in A)*

Recall: *#(classified correctly as A) / #(total examples known in A)*. We can always achieve perfect recall by returning the entire dataset (which will contain many irrelevant examples)

[16] or classification error: *#(misclassified) / #(total examples)* (1 - Accuracy)

Q. Measuring classifier performance?

Binary classification:

Receiver operating curve (ROC): the pair values (false positive rate or FPR, true positive rate or TPR) as a function of a threshold $\theta \in [0, 1]$. An ideal classifier is at $(0, 1)$ (top left corner). Diagonal ($FPR = TPR$): random classifier. *This is the worst we can do*. Any classifier that is below the diagonal can be improved by flipping its decision.

Area under the curve (AUC): It reduces the ROC to a number. Ideal classifier: $AUC = 1$. ROC and AUC allow us to compare classifiers over different loss conditions, and choose a value of θ accordingly. Often, there is not a dominant classifier.

Q. Measuring classifier performance?

$K > 2$ classes:

Again, the most basic measure is the classification error.

Confusion matrix: K $\times$ K matrix where entry *(i, j)* contains the number of instances of class C_i that are classified as C_j.

It allows us to identify which types of misclassification errors tend to occur, e.g. if there are two classes that are frequently confused. **Ideal classifier:** the confusion matrix is diagonal.

Q. What is a confusion matrix?

A **Confusion Matrix** is a simple way of understanding **how well an algorithm is doing at classifying data**. It is just the idea of **false positives** and **false negatives**

	Predicted class		
True class	Positive	Negative	Total
Positive	tp: true positive	fn: false negative	p
Negative	fp: false positive	tn: true negative	n
Total	p'	n'	N

Q. Can you explain what precision and recall are?

Recall is a measure of completeness or quantity, whereas

Precision is a measure of exactness or quality:

$$\underbrace{Recall = \frac{TP}{TP + FN}}_{\text{What proportion of actual positives was identified correctly?}} \qquad \underbrace{Precision = \frac{TP}{TP + FP}}_{\text{What proportion of positive identifications was actually correct?}}$$

High precision means your algorithm has returned substantially *more relevant results than irrelevant ones*, while **high recall** means your algorithm has returned *most of the relevant results*.

Q. Can you explain what are false positives and false negatives?

False positives and **false negatives**, technically referred to as type *I* error and type *II* error respectively.

False positives are incorrect classifications of the presence of a condition when it is actually absent. A **false positive** is when you reject a true null hypothesis.

False negatives are incorrect classifications of the absence of a condition when it is actually present. A **false negative** is when you accept a false null hypothesis.

Q. Provide examples when false positives are more important than false negatives, false negatives are more important than false positives and when these two types of errors are equally important

Case 1, **Airport security**. Ensuring that truly dangerous items like weapons cannot be brought on board an aircraft. Getting false positives is better than getting false negatives: missing cases of actual weapons could lead to dangerous situations.

Case 2, **Cancer screening**. Even though false-positive results could create anxiety and lead to unnecessary and invasive follow-up tests like biopsies, missing cases of actual cancer could lead to delays in treatment that negatively affect somebody's life.

Case 3, **General forecasting** (health weather). Measuring success of testing cases of covid-19 in the US. Tracking rate of false positives and false negatives seems to make sense.

Q. Can you list other metrics derived from the Confusion Matrix?

Confusion matrix related metrics (where N is $TP + TN + FP + FN$)

Measure	Formula
error	$(\text{fp} + \text{fn})/N$
accuracy $= 1-$ error	$(\text{tp} + \text{tn})/N$
tp-rate (hit rate)	tp/p
fp-rate (false alarm rate)	fp/n
precision	tp/p'
recall $=$ tp-rate	tp/p
F-score	$\frac{\text{precision} \times \text{recall}}{(\text{precision} + \text{recall})/2}$
sensitivity $=$ tp-rate	tp/p
specificity $= 1-$ fp-rate	tn/n

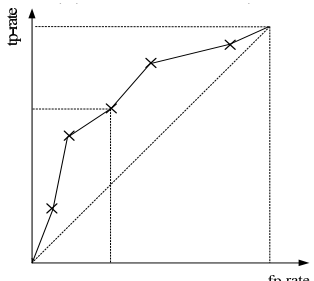
Q. How do you measure how good your classification algorithm is? You have unbalanced numbers, with our class being much larger or smaller than others

You use the Matthews correlation coefficient. The number it yields is between plus or minus one. Plus one means perfect prediction, zero means no better than random.

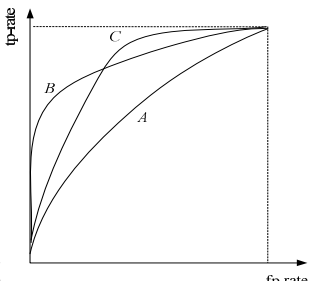
$$\frac{\text{TP x TN - FP x FN}}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

Q. ROC and AUC? Also given an example

E.g., suppose there is a threshold (or parameter) in your classification algorithm that determines whether you have an apple or a non apple object:



(a) Example ROC curve



(b) Different ROC curves for different classifiers

Process (how to split the data)

Q. Process description?

Run fivefold cross-validation and select the best models.

Re-fit models on full training set, evaluate them on the validation set, pick the best one.

Evaluate best model on the test set to gauge its ability to generalize to unseen data.

What do we mean by training, best fit, performance? Any risks?

Q. But, what is training in ML anyway?

Most ML algorithms need to be trained. That is, you give them data and they look for patterns, or best fits, etc.

They know they are doing well when perhaps a loss function has been minimized, or the rewards have been maximized.

Q. Best fits? What do you mean by best fits? Performance?

First of all, when we say “fitting” in ML we are talking about model fitting.

Model fitting is a measure of how well a machine learning model generalizes[17] to similar data on which it was *trained*.

A model that is well-fitted produces more accurate outcomes[18].

[17] Yes, we are talking about ML model performance.

[18] An overfitted model matches the data too closely. An underfitted model doesn’t match closely enough.

Q. How do you measure the performance of a ML model?

We do that by using a **cost function** or a loss function.

A cost function is used to represent how far away our model is from the real data.

A common way to do this is via the quadratic cost function[19]:

$J(\theta) = \frac{1}{2N} \sum_{n=1}^{N} (h_\theta(x^{(n)}) - y^{(n)})^2$.

We are interested in the parameters that minimize this quadratic cost function. This is called ordinary least squares (OLS).

One adjusts the mathematical model usually by varying parameters within the model, so as to *minimize the cost function*. This is interpreted as given the best model that fits the data.

[19] This is just the sum of the squares of the vertical distances between the points and the straight line

Q. What are the risks of not splitting the data?

Overfitting or underfitting to the data

Inaccurate representation of how the model will generalize

Q. Any caveats on training and testing?

Over many epochs, if the test error begins to rise (and it's much bigger than the training error) then you have overfitted.

(Please refer to the Measuring success section to discuss ways in which we can void overfitting.)

Model optimization

Q. Model optimization outline?

We'll discuss the bias-variance trade-off.

We'll cover what we mean by bias and variance from a conceptual level.

Q. Bias and Variance in Machine learning?

Bias[20], in ML, is the algorithm's tendency to consistently learn the wrong thing by not taking into account all the information in the data. (results in inaccurate predictions).

Variance[21] is an algorithm's sensitivity to small fluctuations in the training data set.

[20] High bias is the result of the algorithm missing the relevant relations between features and target outputs
[21] High variance is a result of the algorithm fitting to random noise in the training data

Q. Can you be more specific about Bias and Variance?

Bias is how far away (or error) the trained model is from the correct result *on average*. Where "on average" means over many goes at training the model, using different data:

$$Bias(\hat{f}(x')) = \underbrace{\mathbb{E}[\hat{f}(x)]}_{\text{Average error}} - f(x')$$

Variance is a measure of the magnitude of that error.

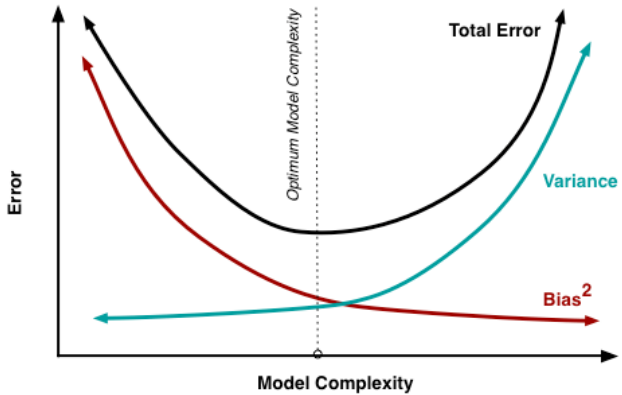
$$Var(\hat{f}(x')) = \mathbb{E}[\hat{f}(x)^2] - \mathbb{E}[\hat{f}(x')]^2$$

There' a trade-off between bias and variance: As one is reduced, the other is increased[22].

[22] This the matter of over-and-underfitting

Q. Bias and Variance tradeoff?

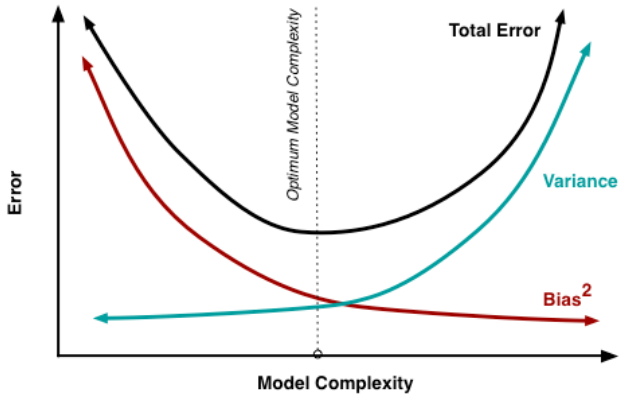
Total error = (Bias + Variance) + Irreducible Error



Model complexity is across the *x* axis and model error across the *y* axis. More complexity means higher variance. Lesser complexity means higher bias.

Q. Bias and Variance tradeoff?

Total error = (Bias + Variance) + Irreducible Error

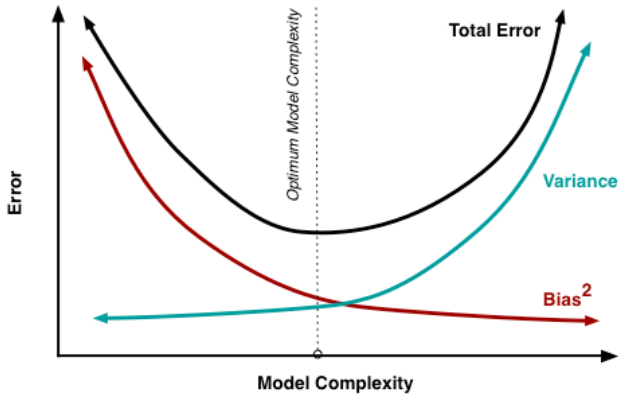


Model complexity is across the *x* axis and model error across the *y* axis. More complexity means higher variance. Lesser complexity means higher bias.

So this what bias/variance tradeoff is all about: **finding the right model complexity** that minimizes both bias and variance (I mean the total error as much as possible.

Q. Bias and Variance tradeoff?

Total error = (Bias + Variance) + Irreducible Error



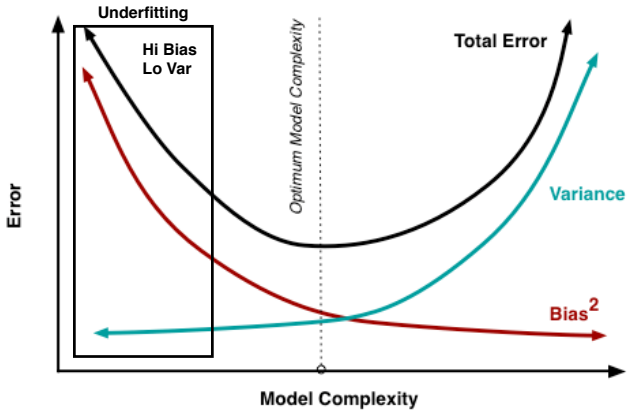
Model complexity is across the *x* axis and model error across the *y* axis. More complexity means higher variance. Lesser complexity means higher bias.

So this what bias/variance tradeoff is all about: **finding the right model complexity** that minimizes both bias and variance (I mean the total error as much as possible.

Total error is very high for very simple models and a very complex model, and then it bottoms out in the middle.

Q. What is Underfitting?

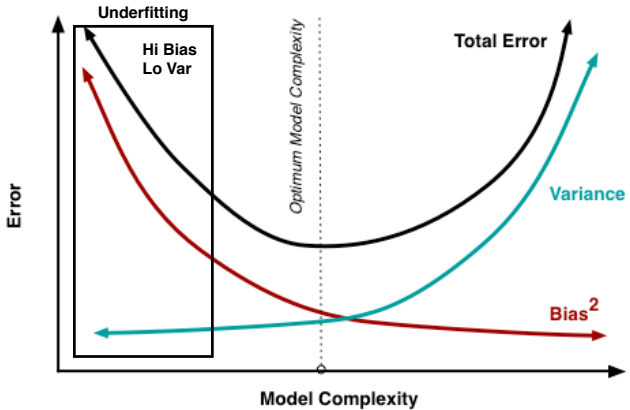
Underfitting occurs when an algorithm cannot capture the underlying trend of the data.



Underfitting happens when the model is too simple with high bias and low variance, and results in high total error.

Q. What is Underfitting?

Underfitting occurs when an algorithm cannot capture the underlying trend of the data.

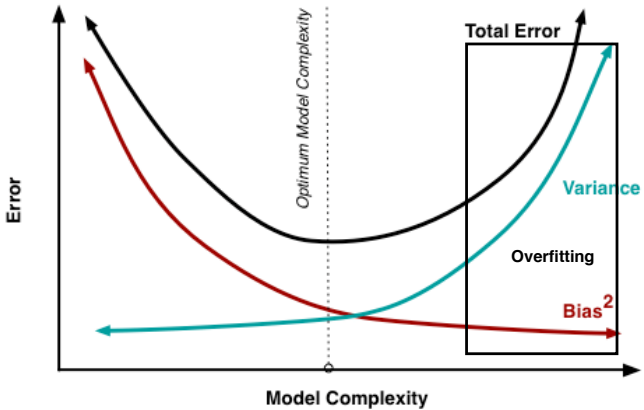


Underfitting happens when the model is too simple with high bias and low variance, and results in high total error.

Underfitting: High bias + low variance

Q. What is Overfitting?

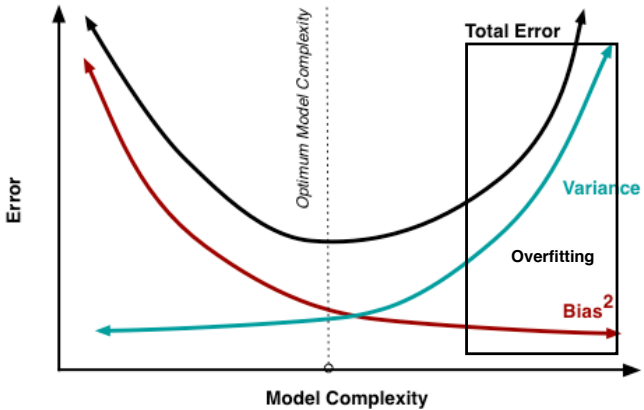
Overfitting occurs when an algorithm fits too closely to a limited set of data (i.e., training set).



In other words, the model might just memorize the examples that it has seen in the training data.

Q. What is Overfitting?

Overfitting occurs when an algorithm fits too closely to a limited set of data (i.e., training set).



In other words, the model might just memorize the examples that it has seen in the training data.

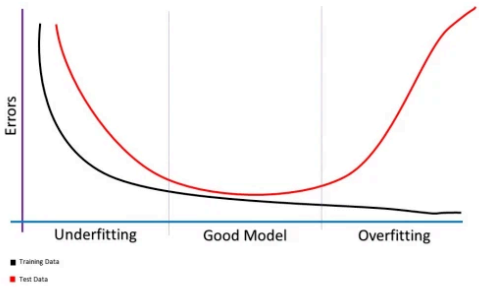
Overfitting: Low bias + high variance.

The actual process?

Q. How do you find the optimal tradeoff?

The goal is to find something in the middle[23] (a model with medium complexity); i.e., **Optimal tradeoff:** Low bias + Low variance.

OKAY, but how do you identify underfit and overfit?



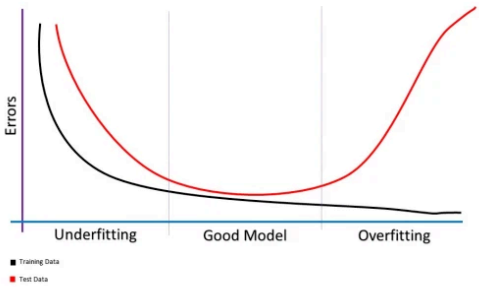
With underfit, we'll have high training error and high test error.

[23] This would learn the true pattern in the data w/o memorizing every example in the training data.

Q. How do you find the optimal tradeoff?

The goal is to find something in the middle[23] (a model with medium complexity); i.e., **Optimal tradeoff:** Low bias + Low variance.

OKAY, but how do you identify underfit and overfit?



With underfit, we'll have high training error and high test error.

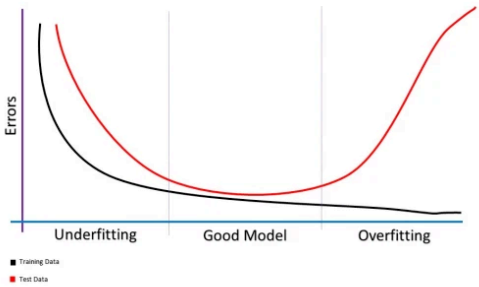
Optimal tradeoff, we'll have low training error and low test error.

[23] This would learn the true pattern in the data w/o memorizing every example in the training data.

Q. How do you find the optimal tradeoff?

The goal is to find something in the middle[23] (a model with medium complexity); i.e., **Optimal tradeoff:** Low bias + Low variance.

OKAY, but how do you identify underfit and overfit?



With underfit, we'll have high training error and high test error.

Optimal tradeoff, we'll have low training error and low test error.

With overfit, we'll have low training error and high test error.

[23] This would learn the true pattern in the data w/o memorizing every example in the training data.

Q. How do you tune a model for optimal complexity?

There are two methods to tune a model for optimal complexity:

Hyper-parameter tuning – choosing a set of optimal hyper-parameters for fitting a ML algorithm (e.g., linear regression)

Regularization – a technique used specifically to reduce overfitting by discouraging overly complex models in some way.

Q. What is a hyper-parameter?

A model **parameter** is a configuration variable that is internal to the model and *whose value can be estimated from data*.

A model **hyper-parameter** is a configuration that is external to the model and *whose value cannot estimated from data*, and *whose value guides how the algorithm learns parameter values from the data*. E.g., depth of a decision tree is a model hyper-parameter vs ticket price or ticket class are model parameters.

Q. What is Regularization?

Regularization is a form of regression, which constrains/regularizes or shrinks the (learned) coefficient estimates towards zero. In other words, this technique reduces overfitting by discouraging learning a more complex model in some way.

The goal of Regularization is to allow enough flexibility for the algorithm to learn the underlying patterns in the data but provides guardrails so it doesn't overfit. See Occam's razor – whenever possible, choose the simplest model to a problem.

Q. Can you provide Regularization examples?

Ridge regression and lasso regression: adding a penalty to the loss function (See Model fitting slide) to constrain coefficients.

Dropout: some nodes are ignored during training which forces the other nodes to take on more or less responsibility for the input-output.

Epoch vs Batch size vs Iteration

Q. What is the difference between these terms?

To find out the difference between these terms you need to know some of the machine learning terms like Gradient Descent to help you better understand.

Q. What is Gradient Descent?

Gradient descent is an *iterative* optimization algorithm used to minimize some *convex* function by *iteratively* moving in the direction of steepest descent as defined by the negative of the gradient.

In ML, we use gradient descent to update the parameters of a ML model.

> Start with an initial guess for each parameter θ_k. Then move θ_k in the direction of the slope.
>
> Update all θ_k parameters simultaneously (known as batch gradient descent), and repeat until convergence.

Q. Unpacking Gradient Descent?

Gradient means the rate of inclination or declination of a slope.

Descent means we are dealing with the inclination of a slope.

The algorithm is iterative means that we need to get the results **multiple times** to get the most optimal result (minima of a curve).

Q. What is Stochastic Gradient Descent?

Similar to batch gradient descent except that you only update using one of the data points each time.

And that data point is chosen randomly. Hence the stochastic.

In other words, stochastic gradient descent pick an n at random and then update using one of the data points. Repeat, picking another data point at random, etc.

Q. What is an epoch?

In some ML methods, one uses the same training data many times, as the algorithm gradually converges, for example, in stochastic gradient descent. Each time the whole training set of data is used in the training that is called an **epoch**[24].

One might see a decreasing error as the number of epochs increases. But that doesn't mean your algorithm is getting better, it could easily mean that you are overfitting[25]. To test for this you introduce a test data set, the data that you've held back.

[24] Typically you won't get decent results until convergence after many epochs

[25] This could happen if the learning algorithm has seen the training data so many times or epochs

Q. So, what is the right numbers of epochs?

Unfortunately, there is no right answer to this question. The answer is different for different data sets but you can say that the numbers of epochs is related to how diverse your data is. For example, do you have only black cats in your data set or is it much more diverse dataset?

Q. What is a batch?

As I said, you can't pass the entire data set into a ML algorithm at once. So, you divide data set into a number of batches or sets or parts.

Q. What is an iteration?

An iteration is the number of batches needed to complete one epoch.

Let's say we have 2000 training examples that we are going to use.

We can divide the dataset of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.

End-to-End Pipeline
(starting from fit initial model)

Run fivefold cross-validation and select the best models.

Q. Fit a basic model using cross validation?

Goal: To understand what the baseline performance looks like [26].

From sklearn, import RandomForestClassifier, and cross-val-score to compute the accuracy of the baseline model.

Next, we will select other models in order to beat our baseline.

[26] We'll use only the training set

Q. Tunning Hyper-parameters?

Run grid search to find the optimal hyper-parameter settings for our models.

Our goal of this step is to find the optimal model that beats that baseline performance.

We use GridSearchCV[27], which yields multiple combinations of hyper-params values, to find the combinations that improves performance[28].

[27] a wrapper around cross fail score that allows us to run grid search within cross validation.
[28] Recall that a model is a configuration of the ML algorithm on specific params values

Re-fit models on full training set, evaluate those models on the validation set and pick the best one.

Q. Evaluating results on Validation set?

Now that we've done some hyper parameter tuning, and we have a good idea of what the best hyper parameter combinations are, let's evaluate these models on the validation set[29].

The process goes like this:

1. Look at additional performance metrics beyond just accuracy (e.g., precision and recall) and also at different ML algorithms.
2. Take the 3 best performing models and **re-fit** them using the whole training data[30].
3. Evaluate these models on the validation set[31].
4. Select the best performing model.

[29] Now the performance shouldn't deviate too much from the performance we saw with the cross-validation.

[30] Why do we need to do that? Because these models were fit on only 80% of the training data; we need now the entire training data.

[31] This is the truth test. If they are overfit or underfit, then they will fail here.

Evaluate that best model on the test set to gauge its ability to generalize to unseen data.

Q. Final model selection and evaluation on test set?

Next, we'll evaluate the best model on the test set. This will give us a truly unbiased view of how it should preform moving forward.

Why do we need to evaluate it on test data, given they have already evaluated on unseen data?

> For the validation set was created by **randomly** distributing examples from the full data set. So if that validation set was slightly different perhaps we would have chosen a different model.
>
> So by nature of this fact that testing on the validation set impacted what model was chosen as our final model, it's technically part of the model training process.
>
> So this final test set is purely for evaluation purposes to see that it matches the performance that we've seen before and to give us more confidence in the performance of the model moving forward.

Machine Learning Algorithms

Regression algorithms (Supervised learning)

Q. What is Regression?

Regression is a statistical process for estimating the relationship among variables, often used to make a prediction about some outcome.

> *From a Machine Learning perspective, the modeling of this relationship is done to ensure generalization – giving the model the ability to predict outputs (dependent variable) for inputs (independent variables) it has never seen before.*

Q. What is linear regression?

Linear regression is one type of regression that is used when trying to predict a continuous target variable (output).

> The essence of LR is that given some correlated data, LR tries to explain a dependent variable in terms of independent variables.
>
> The independent variables are numerical (continuous) and we *fit* straight lines, polynomials or other functions to predict the dependent variables.

Example: Find a relationship between the number of tomatoes on a plant and how much they are watered. Using the algorithm definition for linear regression is $y = mx + b$, where y is the tomatoes number (dependent var), x is how much they are watered (independent var), and b and m (i.e., slope) are coefficients dictating the best fit line[32]. Based on the water data, you could infer the number of tomatoes on a plant for an amount of water for which we don't any data.

[32] e.g., a model that assumes a linear relationship between the input

Q. How does linear regression work?

The way Linear Regression works is by trying to find the weights (namely, m and b) that lead to the best-fitting line for the input data (i.e. X features) we have. The best-fitting line is determined in terms of **lowest cost**.

> Generally, cost refers to the loss or error that the model yields in terms of *how off it is from the actual Training data*.
>
> When it comes to Linear Regression, the **cost function** we usually use is the Squared Error Cost (i.e., Mean Square Error): $\frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$
>
> Using Gradient descent, it tries to find the values for m and b that will give you the lowest MSE score. (Training is basically finding these values.) **The best fitted line is the one with the lowest MSE value.**

Q. Linear regression on many dimensions?

We now have M independent, explanatory, variables, representing the features, and we write all x_s as vectors. For each of N data points we'll have the independent variable $x^{(n)}$ (e.g., squared footage of a house, number of rage bays) and the independent variable $y^{(n)}$ (property value).

We fit the linear function $h_\theta(x) = \theta^T x$ to the y_s, where θ is the vector of the as-yet-unknown parameters.

The cost function remains the same as in the one dimension LR: the quadratic function. And the coefficients are learned using batch or stochastic gradient descent.

Q. Linear regression Recap?

Linear Regression is the process of finding a line that best fits the data points available on the plot, so that we can use it to predict output values for inputs that are not present in the data.

Performance (and error rates) depends on various factors including the how clean and consistent the data is.

In case of bad model performance, we usually go for a **higher polynomial function**. This is basically the introduction of new variables into the Regressor function so that we allow more flexibility to it[33]

[33] However, this will cause the LR line not to be a straight line anymore.

Q. What is logistic regression?

Logistic regression or logistic classifier is a form or regression where the target variable is binary (zero or one, or true or false). It takes the form: $\frac{1}{1+e^{-(mx+b)}}$

Example: Let's say you wanted to examine the relationship between your basketball shooting accuracy and the distance that you shoot from. More specifically, you want a model that takes in "distance from the basket" in feet and spits out the probability that you'll make the shot (1 for a make, 0 for a miss).

In this example, the equation $-(mx + b)$, labeled as z, represents *log(odds of making shot)*[34], so the probability of making a shot (i.e., y) is $\frac{1}{1+e^{-z}}$. Here, m and b are the coefficients we're interested in finding thru optimization, and x is *distance from basket*.

[34] odss = P(Event) / [1-P(Event)]

Q. Logistic regression example?

Shooting Baskets. Generally, the further you get from the basket, the less accurately you shoot: when given a small distance, the model should predict a high probability and when given a large distance it should predict a low probability.

In logistic regression the output Y is in log odds. Let's say you shot 100 free throws and made 70. Based on this sample, your probability of making a free throw is 70%. Your odds of making a free throw is 2.33, which we want to bound between 0 and 1 using *log* and the *sigmoid* function as odds go from 0 to *INF*[35]:

We can write our logistic regression equation: $z = log(mx + b)$, where m and b are the coefficients to be learned.

And to get probability from z, which is in log odds, we apply the sigmoid function: $\frac{1}{1+e^{-z}}$[36] In this case, a high probability means you'll be able to shoot and a lot probability means you won't.

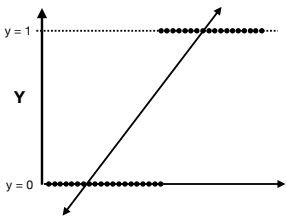
[35] log odds are just probability stated another way

[36] This shows how we can go from a linear estimate of log odds to a probability.

Q. Why won't Linear Regression work for binary target variables?

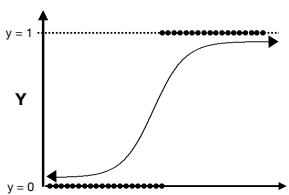
In other words, why do we need two regression algorithms?

Linear regression



$y = mx + b$

Logistic regression



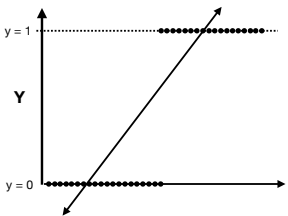
$$y = \frac{1}{1 + e^{-(mx+b)}}$$

Linear regression for binary target variable will have a hard time try to come up with a best fit line that makes any sense.

Q. Why won't Linear Regression work for binary target variables?

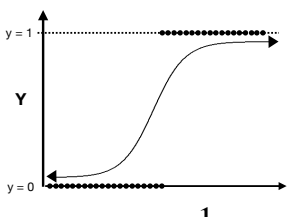
In other words, why do we need two regression algorithms?

Linear regression



$$y = mx + b$$

Logistic regression



$$y = \frac{1}{1 + e^{-(mx+b)}}$$

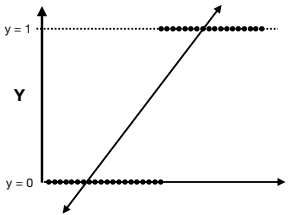
Linear regression for binary target variable will have a hard time try to come up with a best fit line that makes any sense.

It'll try to fit a line that fits all of the data and it will end up predicting negative values and values over one, which is impossible.

Q. Why won't Linear Regression work for binary target variables?

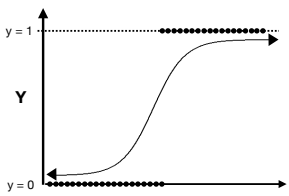
In other words, why do we need two regression algorithms?

Linear regression



$$y = mx + b$$

Logistic regression



$$y = \frac{1}{1 + e^{-(mx+b)}}$$

Linear regression for binary target variable will have a hard time try to come up with a best fit line that makes any sense.

It'll try to fit a line that fits all of the data and it will end up predicting negative values and values over one, which is impossible.

Logistic regression is built off of a logistic or sigmoid curve (S shape) and will always be between zero and one, which makes it a better fit for a binary classification problem.

Q. When should you consider using Logistic Regression?

When to use? Consider using it any time you

- You have a binary target variable.
- You're interested in feature importance or having a better understanding of what's going on within the algorithm.
- You have well-behaved[37] data, need a **quick** initial benchmark.

When not to use? Consider not using it any time you

- You have a continuous target variable.
- You have a massive amount of data[38].
- You have unwieldy[39] data, performance[40] is the only thing that matters.

[37] e.g., no many outliers, no many missing values, no complex relationships
[38] e.g., lots of features and very few rows, or lots of rows and very few features
[39] e.g., many outliers, many missing values, complex relationships
[40] will usually do pretty well on any given problem, but will rarely be the best.

KMeans Clustering Algorithm

Q. What is KMeans?

KMeans is a unsupervised ML algorithm. It takes some data points as input and groups[41] them into k clusters[42] (the output).

This k is called a hyper-parameter; a variable whose value we set before training.

The idea is simple: You have a bunch of vectors $\in \mathbb{R}^d$. Then you init a bunch seed-guesses for the k centers (centroids) of the clusters. You assign each vector to the nearest cluster. And then, when everything is done, you calculate the mean values of the clusters (updated k centers). And then you just do it again: you re-assign the nearest mean. Then you keep on going until it converges[43].

[41] grouping is the training phase of KMeans, and uses the square of the L2 norm as its cost function

[42] It uses the distance between points as a measure of similarity, based on k averages (i.e. means).

[43] Once those centroids stop moving(no further change in cost), the algorithm stop

Q. When to use KMeans?

When to use it

You have unlabeled data and don't know the number of clusters within it.

You a have a decently large data set (less than 10K) with a smaller number of dimensions, these data are numeric, or continuous.

When not to use it

High dimensional data, or data of varying sizes and density.

Messy data with lots of outliers (as it can centroids can be dragged by outliers).

Naive Bayes Classifier

Q. What is Naive Bayes?

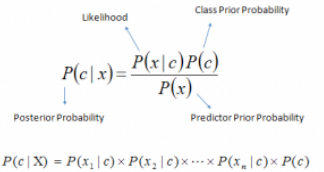
Naive Bayes is a classification technique (Generative model) based on **Bayes Theorem** with an assumption of independence among predictors (features). In other words, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Some **benefits** include:

It's *easy to build* and particularly *useful for very large data sets*.

Along with *simplicity*, it's is known to outperform even highly sophisticated classification methods (NB has better *resilience to missing* data than SVM).

Q. Bayes Theorem (Quick Overview)?



Bayes theorem

Offers a way of calculating posterior probability $\Pr(c \mid x)$ from $\Pr(c)$, $\Pr(x)$, and $\Pr(x \mid c)$.

$\Pr(c \mid x)$ simply means, "given some feature vector $x_i \in X$, what is the probability of sample i belonging to class $c_j \in C$?"

The **objective function** of **Naive Bayes**: Maximize $\Pr(C \mid X)$ given the training data to formulate a decision rule for new data.

Q. How does Naive Bayes work?

Step 1: Convert the data set into a frequency table (co-occurance matrix)

Step 2: Create Likelihood table by finding the probabilities like “Spam” probability (0.29) and probability of “No Spam” (0.64).

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Q. Example: Naive Bayes?

Problem: Players will play if weather is sunny. Is this statement is correct?

We can solve it using above discussed method of posterior probability.

$\Pr(Y \mid \text{Sunny}) = \Pr(\text{Sunny} \mid Y) * \Pr(Y) / \Pr(\text{Sunny})$

Here we have $\Pr(\text{Sunny} \mid Y) = 3/9 = 0.33$, $\Pr(Y) = 9/14 = 0.64$,
$\Pr(\text{Sunny} = 5/14 = 0.36$

Now, $\Pr(Y \mid \text{Sunny}) = 0.33 * 0.64 / 0.36 = 0.60$, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.