**UNIT-2 MODEL IMPROVEMENT AND REGULARIZATION**

Evaluating Machine learning models - Data Preprocessing, Feature Engineering and Feature Learning - Overfitting and Underfitting - Problems of Overfitting: Regularization, Parameter Sharing, Early Stopping, Trade Off Breadth for Depth, Ensemble Methods - Vanishing and Exploding Gradients in Convergence.

**Evaluating Machine Learning Models**

• Assess the performance and effectiveness of machine learning models.

• Evaluation is critical for model selection, tuning, and ensuring real-world utility.

• Metrics, datasets, cross-validation, and comparison with baselines.

**Model Evaluation Metrics**

**Classification Models**

• Accuracy: Proportion of correctly predicted instances.

• Precision: Fraction of true positives among all positive predictions.

• Recall (Sensitivity): Fraction of true positives among all actual positives.

• F1-Score: Harmonic mean of precision and recall.

• ROC Curve and AUC: Receiver Operating Characteristic curve and Area Under the Curve.

• Confusion Matrix: Tabulates true/false positives/negatives.

**Regression Models**

**•** Mean Absolute Error (MAE): Average absolute difference between predictions and actual values.

• Mean Squared Error (MSE): Average squared difference between predictions and actual values.

• Root Mean Squared Error (RMSE): Square root of MSE.

• R-squared (Coefficient of Determination): Measures the proportion of variance explained by the model.

**Data Splitting**

• Training Set: Used for model training.

• Validation Set: Used for hyperparameter tuning and model selection.

• Test Set: Used for final model evaluation. Never used during model development.

**Cross-Validation**

• K-Fold Cross-Validation: Divide the data into K subsets (folds) and iteratively use K-1 folds for training and the remaining fold for testing.

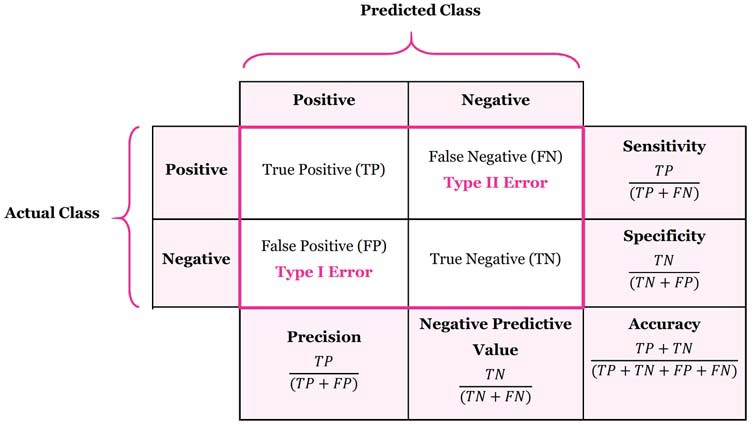
• Helps in estimating model performance more robustly, especially when the dataset is small.

**Overfitting and Underfitting**

• Overfitting: Model performs well on training data but poorly on unseen data.

• Underfitting: Model is too simple to capture the underlying patterns in the data.

**Classification Models**

****

**Cross Validation**

* This process of deciding whether the numerical results quantifying hypothesized relationships between variables, are acceptable as descriptions of the data, is known as validation.
* Generally, an error estimation for the model is made after training, better known as evaluation of residuals.
* In this process, a numerical estimate of the difference in predicted and original responses is done, also called the training error. However, this only gives us an idea about how well our model does on data used to train it.
* Now its possible that the model is underfitting or overfitting the data. So, the problem with this evaluation technique is that it does not give an indication of how well the learner will generalize to an independent/ unseen data set. Getting this idea about our model is known as Cross Validation.

**Holdout Method**

* Now a basic remedy for this involves removing a part of the training data and using it to get predictions from the model trained on rest of the data.
* The error estimation then tells how our model is doing on unseen data or the validation set.
* This is a simple kind of cross validation technique, also known as the holdout method. Although this method doesn’t take any overhead to compute and is better than traditional validation, it still suffers from issues of high variance.
* This is because it is not certain which data points will end up in the validation set and the result might be entirely different for different sets.

**K-Fold Cross Validation**

* As there is never enough data to train your model, removing a part of it for validation poses a problem of underfitting.
* By reducing the training data, we risk losing important patterns/ trends in data set, which in turn increases error induced by bias.
* So, what we require is a method that provides ample data for training the model and also leaves ample data for validation. K Fold cross validation does exactly that.
* In K Fold cross validation, the data is divided into k subsets. Now the holdout method is repeated k times, such that each time, one of the k subsets is used as the test set/ validation set and the other k-1 subsets are put together to form a training set.
* The error estimation is averaged over all k trials to get total effectiveness of our model. As can be seen, every data point gets to be in a validation set exactly once, and gets to be in a training set k-1 times.
* This significantly reduces bias as we are using most of the data for fitting, and also significantly reduces variance as most of the data is also being used in validation set.
* Interchanging the training and test sets also adds to the effectiveness of this method. As a general rule and empirical evidence, K = 5 or 10 is generally preferred, but nothing’s fixed and it can take any value.

**Stratified K-Fold Cross Validation**

* In some cases, there may be a large imbalance in the response variables. For example, in dataset concerning price of houses, there might be large number of houses having high price.
* Or in case of classification, there might be several times more negative samples than positive samples.
* For such problems, a slight variation in the K Fold cross validation technique is made, such that each fold contains approximately the same percentage of samples of each target class as the complete set, or in case of prediction problems, the mean response value is approximately equal in all the folds. This variation is also known as Stratified K Fold.

Above explained validation techniques are also referred to as Non-exhaustive cross validation methods. These do not compute all ways of splitting the original sample, i.e. you just have to decide how many subsets need to be made. Also, these are approximations of method explained below, also called Exhaustive Methods, that computes all possible ways the data can be split into training and test sets.

**Leave-P-Out Cross Validation**

* This approach leaves p data points out of training data, i.e. if there are n data points in the original sample then, n-p samples are used to train the model and p points are used as the validation set.
* This is repeated for all combinations in which original sample can be separated this way, and then the error is averaged for all trials, to give overall effectiveness.
* This method is exhaustive in the sense that it needs to train and validate the model for all possible combinations, and for moderately large p, it can become computationally infeasible.
* A particular case of this method is when p = 1. This is known as Leave one out cross validation. This method is generally preferred over the previous one because it does not suffer from the intensive computation, as number of possible combinations is equal to number of data points in original sample or n.

**Need for cross validation**

* Cross Validation is a very useful technique for assessing the effectiveness of your model, particularly in cases where you need to mitigate overfitting.
* It is also of use in determining the hyper parameters of your model, in the sense that which parameters will result in lowest test error.
* This is all the basic you need to get started with cross validation.

**Data Preprocessing**

**1. Data Cleaning**

* **Handling Missing Values**: Strategies include removal, imputation (mean, median, mode), or using advanced methods.
* **Outlier Detection and Treatment**: Identifying and dealing with outliers that can skew models.

**2. Data Transformation**

* **Scaling**: Normalizing or standardizing features to the same scale.
* **Encoding Categorical Variables**: Transforming categorical data into numerical format (one-hot encoding, label encoding).
* **Datetime Conversion**: Extracting useful information from date and time fields.

**3. Data Reduction**

* **Dimensionality Reduction**: Reducing the number of features while preserving relevant information (e.g., PCA, t-SNE).
* **Feature Selection**: Selecting the most important features based on relevance (e.g., mutual information, feature importance).

**Feature Engineering**

**1. Manual Feature Engineering**

* **Domain Knowledge**: Leveraging expertise to create meaningful features.
* **Feature Extraction**: Crafting new features by combining or transforming existing ones.

**2. Automated Feature Engineering**

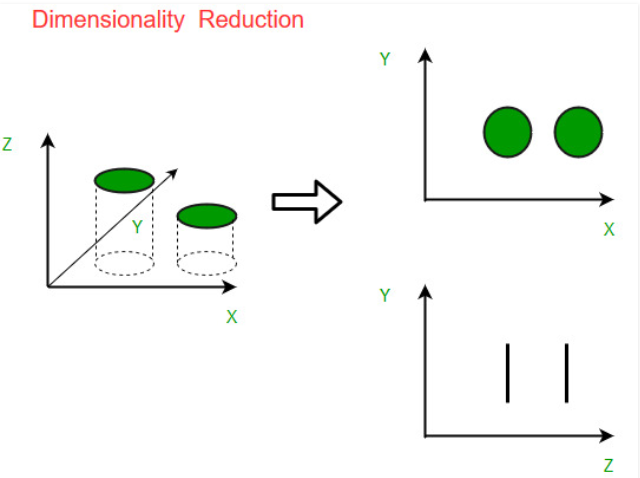
* **Feature Generation**: Using algorithms to create new features automatically (e.g., autoencoders).
* **Feature Importance**: Ranking features based on their contribution to model performance.

**DIMENTIONALITY REDUCTION**

* In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features.
* The higher the number of features, the harder it gets to visualize the training set and then work on it.
* Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play.
* **Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables.**
* It can be divided into feature selection and feature extraction.

**Example**

* An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not.
* This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc.
* However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree.
* Hence, we can reduce the number of features in such problems.
* A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line.
* The below figure illustrates this concept, where a 3-D feature space is split into two 1-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.



**Components of Dimensionality Reduction**

There are two components of dimensionality reduction:

**Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:

* Filter
* Wrapper
* Embedded

**Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

**Methods of Dimensionality Reduction**

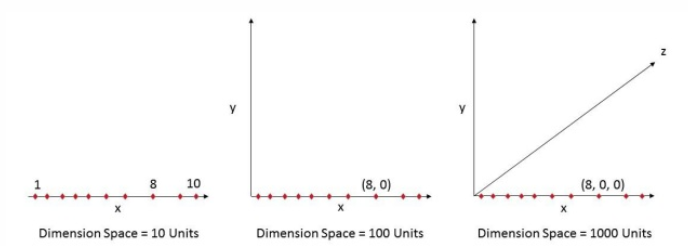
The various methods used for dimensionality reduction include:

* Principal Component Analysis (PCA)
* Linear Discriminant Analysis (LDA)
* Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear or non-linear, depending upon the method used.

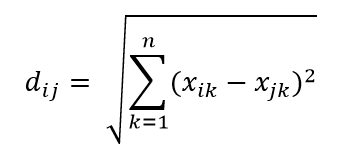
**Curse of Dimensionality**

* As the number of features or dimensions grows, the amount of data we need to generalize accurately grows exponentially
* **Curse of Dimensionality refers to non-intuitive properties of data observed when working in high-dimensional space**, specifically related to usability and interpretation of distances and volumes.
* For quick grasp, consider [this example](https://www.quora.com/What-is-the-curse-of-dimensionality/answer/Kevin-Lacker): Say, you dropped a coin on a 100 meter line. How do you find it? Simple, just walk on the line and search. But what if it’s 100 x 100 sq. m. field? It’s already getting tough, trying to search a (roughly) football ground for a single coin. But what if it’s 100 x 100 x 100 cu.m space?! You know, football ground now has thirty-story height. Good luck finding a coin there! That, in essence is “curse of dimensionality”.



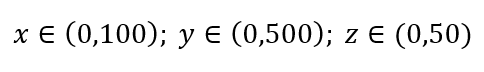
## Many ML methods use Distance Measure

* Most **segmentation** and **clustering** methods rely on computing distances between observations.
* Well known k-Means segmentation assigns points to nearest center. DBSCAN and Hierarchical clustering also required distance metrics.
* Distribution and density based **outlier detection** algorithms also make use of distance relative to other distances to mark outliers.
* Supervised classification solutions like **k-Nearest Neighbours** method also use distance between observations to assign class to unknown observation.
* **Support Vector Machine** method involves transforming observations around select Kernels based on distance between observation and the kernel.
* Common form of **recommendation systems** involve distance based similarity among user and item attribute vectors.
* Even when other forms of distances are used, number of dimensions plays a role in analytic design.
* One of the most common distance metrics is Euclidian Distance metric, which is simply linear distance between two points in multi-dimensional hyper-space. Euclidian Distance for point i and point j in n dimensional space can be computed as:

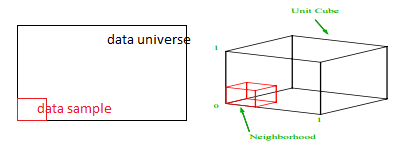


## Distance plays havoc in high-dimension

* Consider simple process of data sampling. Suppose the black outside box in Fig. 1 is data universe with uniform distribution of data points across whole volume, and that we want to sample 1% of observations as enclosed by red inside box. Black box is hyper-cube in multi-dimensional space with each side representing range of value in that dimension. For simple 3-dimensional example in Fig. 1, we may have following range:



**Figure 1 : Sampling**

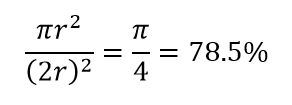


What is proportion of each range should we sample to obtain that 1% sample? For 2-dimensions, 10% of range will achieve overall 1% sampling, so we may select x∈(0,10) and y∈(0,50) and expect to capture 1% of all observations. This is because 10%2=1%. Do you expect this proportion to be higher or lower for 3-dimension?

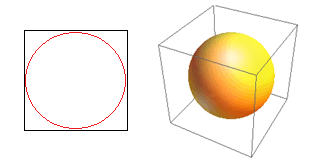
Even though our search is now in additional direction, proportional actually increases to 21.5%. And not only increases, for just one additional dimension, it doubles! And you can see that we have to cover almost one-fifth of each dimension just to get one-hundredth of overall! In 10-dimensions, this proportion is 63% and in 100-dimensions – which is not uncommon number of dimensions in any real-life machine learning – one has to sample 95% of range along each dimension to sample 1% of observations! This mind-bending result happens because in high dimensions spread of data points becomes larger even if they are uniformly spread.

This has consequence in terms of design of experiment and sampling. Process becomes very computationally expensive, even to the extent that sampling asymptotically approaches population despite sample size remaining much smaller than population.

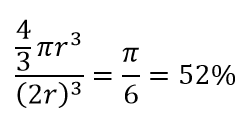
Consider another huge consequence of high dimensionality. Many algorithms measure distance between two data points to define some sort of near-ness (DBSCAN, Kernels, k-Nearest Neighbour) in reference to some pre-defined distance threshold. In 2-dimensions, we can imagine that two points are near if one falls within certain radius of another. Consider left image in Fig. 2. What’s share of uniformly spaced points within black square fall inside the red circle? That is about



**Figure 2 : Near-ness**



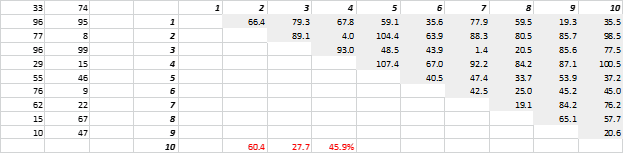
So if you fit biggest circle possible inside the square, you cover 78% of square. Yet, biggest sphere possible inside the cube covers only



of the volume. This volume reduces exponentially to 0.24% for just 10-dimension! What it essentially means that in high-dimensional world every single data point is at corners and nothing really is center of volume, or in other words, center volume reduces to nothing because there is (almost) no center! This has huge consequences of distance based clustering algorithms. All the distances start looking like same and any distance more or less than other is more random fluctuation in data rather than any measure of dissimilarity!

Fig. 3 shows randomly generated 2-D data and corresponding all-to-all distances. Coefficient of Variation in distance, computed as Standard Deviation divided by Mean, is 45.9%. Corresponding number of similarly generated 5-D data is 26.5% and for 10-D is 19.1%. Admittedly this is one sample, but trend supports the conclusion that in high-dimensions every distance is about same, and none is near or far!

**Figure 3 : Distance Clustering**



## High-dimension affects other things too

* Apart from distances and volumes, number of dimensions creates other practical problems.
* Solution run-time and system-memory requirements often non-linearly escalate with increase in number of dimensions.
* Due to exponential increase in feasible solutions, many optimization methods cannot reach global optima and have to make do with local optima.
* Further, instead of closed-form solution, optimization must use search based algorithms like gradient descent, genetic algorithm and simulated annealing.
* More dimensions introduce possibility of correlation and parameter estimation can become difficult in regression approaches.

## Dealing with High-dimension

* Correlation analysis, clustering, information value, variance inflation factor, principal component analysis are some of the ways in which number of dimensions can be reduced.
* Number of variables, observations or features a data point is made up of is called dimension of data.
* For instance, any point in space can be represented using 3 co-ordinates of length, breadth, and height, and has 3 dimensions
* The use of the term "curse of dimensionality" in machine learning is related to the fact that one can easily imagine a target function (to be learned) that is very non-smooth, for example having an exponential number of modes (ups and downs), with respect to dimensionality (the number of scalar input variables).
* Imagine that in order to produce a good prediction, our learner needs to distinguish between 10 different values of each of n variables.
* Then it may need to distinguish between 10^n different configurations of the input n-dimensional vector.
* With n easily in the hundreds, thousands or more, this is much much more than the number of examples one can hope to gather (or even the number of atoms in the universe).
* With most learning algorithms, and in particular with classical non-parametric learning algorithms (e.g. nearest-neighbor, Parzen, Gaussian kernel SVM, Gaussian kernel Gaussian Process, etc.) the learner will need to see at least one example for each of these many configurations (at least as many as necessary to cover all the variations of configurations of interest), in order to produce a correct answer around each of these configurations, one that is different from the target value required for other nearby configurations.
* However, the real difficulty is not with the dimensionality, but with the complexity (number of modes, number of ups and downs, number of different regions in input space that must be distinguished) of the target function to be learned.
* A digit image may have 32x32=1024 input grey-level pixels but it may be that the function one needs to learn (say a classifier to distinguish between 3's and 4's) can be very smooth (in fact a linear classifier will already do a reasonable job, in this case).
* If there is a so-called manifold (lower-dimensional region) near which the examples congregate, then the actual dimensionality of interest may be much smaller than the number of input variables.
* But even if the target function can be captured on a low-dimensional manifold, what really matters is how much it **varies** on that manifold, i.e., the number of different regions that need to be distinguished, because this will correspond to the number of training examples needed to achieve a good performance, when using a standard non-parametric local learner (such as the above).
* On the other hand, even if the target function is apparently very complex (e.g., a huge number of neighboring configurations need to be distinguished), whether in high or low dimension, it may still be possible to **generalize well from a reasonably small number** of training examples, so long as this apparent complexity is actually organized.
* If these ups and downs have some structure (the simplest you can think of is just repetitions of the same pattern), then some learning algorithms can potentially catch that.
* For this to happen, the learning algorithm must be able to generalize non-locally, even to regions not "covered" by training examples, far from training examples.

**OVERFITTING-UNDERFITTING**

The cause of poor performance in machine learning is either overfitting or underfitting the data.

**Approximate a Target Function in Machine Learning**

* Supervised machine learning is best understood as approximating a target function (f) that maps input variables (X) to an output variable (Y). Y = f(X).
* This characterization describes the range of classification and prediction problems and the machine algorithms that can be used to address them.
* An important consideration in learning the target function from the training data is how well the model generalizes to new data. Generalization is important because the data we collect is only a sample, it is incomplete and noisy.

**Generalization in Machine Learning**

* In machine learning we describe the learning of the target function from training data as inductive learning.
* Induction refers to learning general concepts from specific examples which is exactly the problem that supervised machine learning problems aim to solve. This is different from deduction that is the other way around and seeks to learn specific concepts from general rules.
* **Generalization** refers to how well the concepts learned by a machine learning model apply to specific examples not seen by the model when it was learning.
* The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen.
* There is a terminology used in machine learning when we talk about how well a machine learning model learns and generalizes to new data, namely overfitting and underfitting.
* Overfitting and underfitting are the two biggest causes for poor performance of machine learning algorithms.

**Statistical Fit**

* In statistics, a fit refers to how well you approximate a target function.
* This is good terminology to use in machine learning, because supervised machine learning algorithms seek to approximate the unknown underlying mapping function for the output variables given the input variables.
* Statistics often describe the goodness of fit which refers to measures used to estimate how well the approximation of the function matches the target function.
* Some of these methods are useful in machine learning (e.g. calculating the residual errors), but some of these techniques assume we know the form of the target function we are approximating, which is not the case in machine learning.
* If we knew the form of the target function, we would use it directly to make predictions, rather than trying to learn an approximation from samples of noisy training data.

**Overfitting in Machine Learning**

* Overfitting refers to a model that models the training data too well.
* Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.
* This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model.
* The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.
* Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function.
* As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.
* For example, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up.
* A statistical model is said to be overfitted, when we train it with a lot of data.
* When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set.
* Then the model does not categorize the data correctly, because of too much of details and noise.
* The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models.
* A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

**Underfitting in Machine Learning**

* Underfitting refers to a model that can neither model the training data nor generalize to new data.
* An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.
* Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.
* A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data.
* Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have less data to build an accurate model and also when we try to build a linear model with a non-linear data.
* In such cases the rules of the machine learning model are too easy and flexible to be applied on such a minimal data and therefore the model will probably make a lot of wrong predictions.
* Underfitting can be avoided by using more data and also reducing the features by feature selection.

**A Good Fit in Machine Learning**

* Ideally, you want to select a model at the sweet spot between underfitting and overfitting.This is the goal, but is very difficult to do in practice.
* To understand this goal, we can look at the performance of a machine learning algorithm over time as it is learning a training data.
* We can plot both the skill on the training data and the skill on a test dataset we have held back from the training process.
* Over time, as the algorithm learns, the error for the model on the training data goes down and so does the error on the test dataset. If we train for too long, the performance on the training dataset may continue to decrease because the model is overfitting and learning the irrelevant detail and noise in the training dataset.
* At the same time the error for the test set starts to rise again as the model’s ability to generalize decreases.
* The sweet spot is the point just before the error on the test dataset starts to increase where the model has good skill on both the training dataset and the unseen test dataset.
* You can perform this experiment with your favorite machine learning algorithms. This is often not useful technique in practice, because by choosing the stopping point for training using the skill on the test dataset it means that the testset is no longer “unseen” or a standalone objective measure.
* Some knowledge (a lot of useful knowledge) about that data has leaked into the training procedure.
* There are two additional techniques you can use to help find the sweet spot in practice: **resampling methods and a validation dataset.**
* Ideally, the case when the model makes the predictions with 0 error, is said to have a good fit on the data. This situation is achievable at a spot between overfitting and underfitting.
* In order to understand it we will have to look at the performance of our model with the passage of time, while it is learning from training dataset.
* With the passage of time, our model will keep on learning and thus the error for the model on the training and testing data will keep on decreasing.
* If it will learn for too long, the model will become more prone to overfitting due to presence of noise and less useful details. Hence the performance of our model will decrease.
* In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point the model is said to have good skills on training dataset as well our unseen testing dataset.

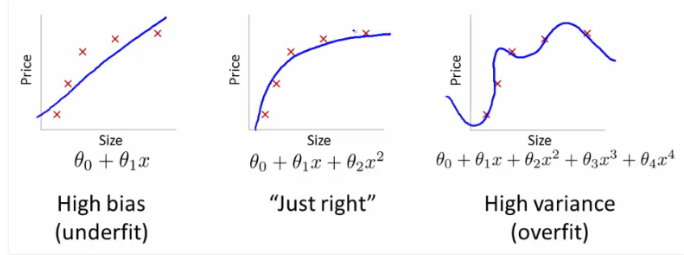
**How To Limit Overfitting**

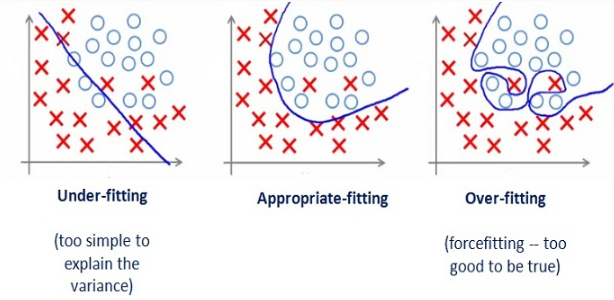
* Both overfitting and underfitting can lead to poor model performance. But by far the most common problem in applied machine learning is overfitting.
* Overfitting is such a problem because the evaluation of machine learning algorithms on training data is different from the evaluation we actually care the most about, namely how well the algorithm performs on unseen data.
* There are two important techniques that you can use when evaluating machine learning algorithms to limit overfitting:

1. Use a resampling technique to estimate model accuracy.

2. Hold back a validation dataset.

* The most popular resampling technique is k-fold cross validation. It allows you to train and test your model k-times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data.
* A validation dataset is simply a subset of your training data that you hold back from your machine learning algorithms until the very end of your project.
* After you have selected and tuned your machine learning algorithms on your training dataset you can evaluate the learned models on the validation dataset to get a final objective idea of how the models might perform on unseen data.
* Using cross validation is a gold standard in applied machine learning for estimating model accuracy on unseen data. If you have the data, using a validation dataset is also an excellent practice.





#### How to avoid Overfitting:

* **Cross- Validation:** A standard way to find out-of-sample prediction error is to use 5-fold cross validation.
* **Early Stopping:** Its rules provide us the guidance as to how many iterations can be run before learner begins to over-fit.
* **Pruning:** Pruning is extensively used while building related models. It simply removes the nodes which add little predictive power for the problem in hand.
* **Regularization:** It introduces a cost term for bringing in more features with the objective function. Hence it tries to push the coefficients for many variables to zero and hence reduce cost term.

**HYPERPARAMETERS**

* Model optimization is one of the toughest challenges in the implementation of machine learning solutions.
* Entire branches of machine learning and deep learning theory have been dedicated to the optimization of models.
* Typically, we think about model optimization as a process of regularly modifying the code of the model in order to minimize the testing error.
* However, theare of machine learning optimization often entails fine tuning elements that live outside the model but that can heavily influence its behavior.
* Machine learning often refers to those hidden elements as hyperparameters as they are one of the most critical components of any machine learning application.
* Hyperparameters are settings that can be tuned to control the behavior of a machine learning algorithm.
* Conceptually, hyperparameters can be considered orthogonal to the learning model itself in the sense that, although they live outside the model, there is a direct relationship between them.
* The criteria of what defines a hyperparameter is incredibly abstract and flexible.
* Sure, there are well established hyperparameters such as the number of hidden units or the learning rate of a model but there are also an arbitrarily number of settings that can play the role of hyperparameters for specific models.
* In general, hyperparameters are very specific to the type of machine learning mode you are trying to optimize.
* Sometimes, a setting is modeled as a hyperparameter because is not appropriate to learn it from the training set.
* A classic example are settings that control the capacity of a model( the spectrum of functions that the model can represent).
* If a machine learning algorithm learns those settings directly from the training set, then it is likely to try to maximize them which will cause the model to overfit( poor generalization).
* If hyperparameters are not learned from the training set, then how does a model learn them? Remember that classic role in machine learning models to split the input dataset in an 80/20 percent ratio between the training set and the validation set respectively? Well, part of the role of that 20% validation set is to guide the selection of hyperparameters. Technically, the validation set is used to “train” the hyperparameters prior to optimization.

**Some Examples of Hyperparameters**

The number and diversity of hyperparameters in machine learning algorithms is very specific to each model. However, there some classic hyperparameters that we should always keep our eyes on and that should help you think about this aspect of machine learning solutions:

— **Learning Rate:** The mother of all hyperparameters, the learning rate quantifies the learning progress of a model in a way that can be used to optimize its capacity.

— **Number of Hidden Units:** A classic hyperparameter in deep learning algorithms, the number of hidden units is key to regulate the representational capacity of a model.

— **Convolution Kernel Width:** In convolutional Neural Networks(CNNs), the Kernel Width influences the number of parameters in a model which, in turns, influences its capacity.

**What is a Model Parameter?**

* A model parameter is a configuration variable that is internal to the model and whose value can be estimated from data.
* They are required by the model when making predictions.
* They values define the skill of the model on your problem.
* They are estimated or learned from data.
* They are often not set manually by the practitioner.
* They are often saved as part of the learned model.

Parameters are key to machine learning algorithms. They are the part of the model that is learned from historical training data. Some examples of model parameters include:

* The weights in an artificial neural network.
* The support vectors in a support vector machine.
* The coefficients in a linear regression or logistic regression.

**What is a Model Hyperparameter?**

* A model hyperparameter is a configuration that is external to the model and whose value cannot be estimated from data.
* They are often used in processes to help estimate model parameters.
* They are often specified by the practitioner.
* They can often be set using heuristics.
* They are often tuned for a given predictive modeling problem.
* We cannot know the best value for a model hyperparameter on a given problem. We may use rules of thumb, copy values used on other problems, or search for the best value by trial and error.

When a machine learning algorithm is tuned for a specific problem, such as when you are using a grid search or a random search, then you are tuning the hyperparameters of the model or order to discover the parameters of the model that result in the most skillful predictions. Model hyperparameters are often referred to as model parameters which can make things confusing. A good rule of thumb to overcome this confusion is as follows:

**If you have to specify a model parameter manually then it is probably a model hyperparameter.**

Some examples of model hyperparameters include:

* The learning rate for training a neural network.
* The C and sigma hyperparameters for support vector machines.
* The k in k-nearest neighbors.

**VALIDATION SETS**

* A validation dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning model’s hyperparameters.
* The validation dataset is different from the test dataset that is also held back from the training of the model, but is instead used to give an unbiased estimate of the skill of the final tuned model when comparing or selecting between final models.
* Generally, the term “validation set” is used interchangeably with the term “test set” and refers to a sample of the dataset held back from training the model.
* The evaluation of a model skill on the training dataset would result in a biased score. Therefore the model is evaluated on the held-out sample to give an unbiased estimate of model skill. This is typically called a train-test split approach to algorithm evaluation.
* Suppose that we would like to estimate the test error associated with fitting a particular statistical learning method on a set of observations.
* The validation set approach is a very simple strategy for this task. It involves randomly dividing the available set of observations into two parts, a training set and a validation set or hold-out set.
* The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set.
* The resulting validation set error rate — typically assessed using MSE in the case of a quantitative response—provides an estimate of the test error rate.the final model evaluation must be performed on a held out dataset that has not been used prior, either for training the model or tuning the model parameters.
* Ideally, the model should be evaluated on samples that were not used to build or fine-tune the model, so that they provide an unbiased sense of model effectiveness.
* When a large amount of data is at hand, a set of samples can be set aside to evaluate the final model. The “training” data set is the general term for the samples used to create the model, while the “test” or “validation” data set is used to qualify performance.

**Validation vs Test set**

* A validation dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning model’s hyperparameters.
* The validation dataset is different from the test dataset that is also held back from the training of the model, but is instead used to give an unbiased estimate of the skill of the final tuned model when comparing or selecting between final models.
* Generally, the term “validation set” is used interchangeably with the term “test set” and refers to a sample of the dataset held back from training the model.
* The evaluation of a model skill on the training dataset would result in a biased score.
* Therefore the model is evaluated on the held-out sample to give an unbiased estimate of model skill. This is typically called a train-test split approach to algorithm evaluation.
* The final model evaluation must be performed on a held out dataset that has not been used prior, either for training the model or tuning the model parameters.
* Ideally, the model should be evaluated on samples that were not used to build or fine-tune the model, so that they provide an unbiased sense of model effectiveness.
* When a large amount of data is at hand, a set of samples can be set aside to evaluate the final model. The “training” data set is the general term for the samples used to create the model, while the “test” or “validation” data set is used to qualify performance.
* Perhaps traditionally the dataset used to evaluate the final model performance is called the “test set”.
* The importance of keeping the test set completely separate is peeking. They suggest locking the test set away completely until all model tuning is complete.
* Peeking is a consequence of using test-set performance to both choose a hypothesis and evaluate it. T
* he way to avoid this is to really hold the test set out—lock it away until you are completely done with learning and simply wish to obtain an independent evaluation of the final hypothesis.
* The training dataset used to fit the model can be further split into a training set and a validation set, and that it is this subset of the training dataset, called the validation set, that can be used to get an early estimate of the skill of the model.
* If the test set is locked away, but you still want to measure performance on unseen data as a way of selecting a good hypothesis, then divide the available data (without the test set) into a training set and a validation set.
* Specifically, training, validation, and test sets are defined as follows:
* Training set: A set of examples used for learning, that is to fit the parameters of the classifier.
* Validation set: A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network.
* Test set: A set of examples used only to assess the performance of a fully-specified classifier.

**ESTIMATORS**

Estimator:

* This isn't a word with a rigorous definition but it usually associated with finding a current value in data.
* If we didn't explicitly count the change in our pocket we might use an estimate.
* That said, in machine learning it is most frequently used in conjunction with parameter estimation or density estimation.
* In both cases there is an assumption that data we currently have comes in a form that can be described with a function.
* With parameter estimation, we believe that the function is a known function that has additional parameters such as rate or mean and we may estimate the value of those parameters.
* In density estimation we may not even have an assumption about the function but we will attempt to estimate the function regardless.
* Once we have an estimation we may have at our disposal a model. The estimator then would be the method of generating estimations, for example the method of maximum likelihood.

**Model:**

* The model is the function (or pooled set of functions) that you may accept or reject as being representative of your phenomenon.
* The word stems from the idea that you may apply domain knowledge to explaining/predicting the phenomenon though this isn't required.
* A non-parametric model might be derived entirely from the data at hand but the result is often still called a model.
* This terminology highlights the fact that what has been constructed when a model has been constructed is not reality but only a 'model' of reality.
* As George Box has said "All models are wrong but some are useful". Having a model allows you to predict but that may not be its purpose; it could also be used to simulate or to explain.

**BIAS AND VARIANCE**

* Supervised machine learning algorithms can best be understood through the lens of the bias-variance trade-off.
* In supervised machine learning an algorithm learns a model from training data.
* The goal of any supervised machine learning algorithm is to best estimate the mapping function (f) for the output variable (Y) given the input data (X).
* The mapping function is often called the target function because it is the function that a given supervised machine learning algorithm aims to approximate.
* The prediction error for any machine learning algorithm can be broken down into three parts:Bias Error, Variance Error, Irreducible Error
* The irreducible error cannot be reduced regardless of what algorithm is used. It is the error introduced from the chosen framing of the problem and may be caused by factors like unknown variables that influence the mapping of the input variables to the output variable.

## Bias Error

* Bias are the simplifying assumptions made by a model to make the target function easier to learn.
* Generally, parametric algorithms have a high bias making them fast to learn and easier to understand but generally less flexible. In turn, they have lower predictive performance on complex problems that fail to meet the simplifying assumptions of the algorithms bias.

**Low Bias**: Suggests less assumptions about the form of the target function. EG: Decision Trees, k-Nearest Neighbors and Support Vector Machines.

**High-Bias**: Suggests more assumptions about the form of the target function. EG:  Linear Regression, Linear Discriminant Analysis and Logistic Regression.

## Variance Error

* Variance is the amount that the estimate of the target function will change if different training data was used.
* The target function is estimated from the training data by a machine learning algorithm, so we should expect the algorithm to have some variance.
* Ideally, it should not change too much from one training dataset to the next, meaning that the algorithm is good at picking out the hidden underlying mapping between the inputs and the output variables.
* Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data. This means that the specifics of the training have influences the number and types of parameters used to characterize the mapping function**.**
* **Low Variance**: Suggests small changes to the estimate of the target function with changes to the training dataset. EG: Linear Regression, Linear Discriminant Analysis and Logistic Regression.
* **High Variance**: Suggests large changes to the estimate of the target function with changes to the training dataset. EG: Decision Trees, k-Nearest Neighbors and Support Vector Machines.
* Generally, nonparametric machine learning algorithms that have a lot of flexibility have a high variance.

**Bias-Variance Trade-Off**

* The goal of any supervised machine learning algorithm is to achieve low bias and low variance. In turn the algorithm should achieve good prediction performance.
* You can see a general trend in the examples above:
* Parametric or linear machine learning algorithms often have a high bias but a low variance.
* Non-parametric or non-linear machine learning algorithms often have a low bias but a high variance.
* The parameterization of machine learning algorithms is often a battle to balance out bias and variance.

**Examples of configuring the bias-variance trade-off for specific algorithms:**

* The k-nearest neighbors algorithm has low bias and high variance, but the trade-off can be changed by increasing the value of k which increases the number of neighbors that contribute t the prediction and in turn increases the bias of the model.
* The support vector machine algorithm has low bias and high variance, but the trade-off can be changed by increasing the C parameter that influences the number of violations of the margin allowed in the training data which increases the bias but decreases the variance.

There is no escaping the relationship between bias and variance in machine learning.

* Increasing the bias will decrease the variance.
* Increasing the variance will decrease the bias.
* There is a trade-off at play between these two concerns and the algorithms you choose and the way you choose to configure them are finding different balances in this trade-off for your problem
* In reality, we cannot calculate the real bias and variance error terms because we do not know the actual underlying target function. Nevertheless, as a framework, bias and variance provide the tools to understand the behavior of machine learning algorithms in the pursuit of predictive performance.

**REGULARIZATION**

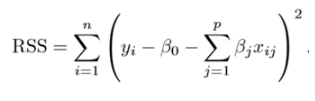
* One of the major aspects of training your machine learning model is avoiding overfitting.

The model will have a low accuracy if it is overfitting.

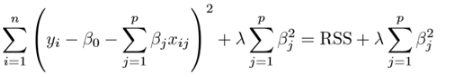
* This happens because your model is trying too hard to capture the noise in your training dataset.
* By noise we mean the data points that don’t really represent the true properties of your data, but random chance.
* Learning such data points, makes your model more flexible, at the risk of overfitting.
* The concept of balancing bias and variance, is helpful in understanding the phenomenon of overfitting.



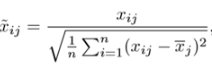
* The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.
* Now, this will adjust the coefficients based on your training data.
* If there is noise in the training data, then the estimated coefficients won’t generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.



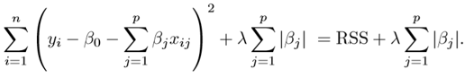
**Ridge Regression**



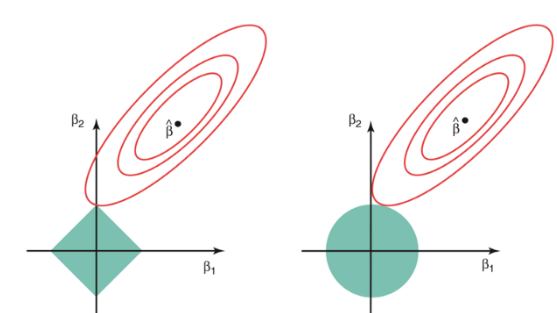
* Above image shows ridge regression, where the RSS is modified by adding the shrinkage quantity. Now, the coefficients are estimated by minimizing this function.
* Here, λ is the tuning parameter that decides how much we want to penalize the flexibility of our model.
* The increase in flexibility of a model is represented by increase in its coefficients, and if we want to minimize the above function, then these coefficients need to be small.
* This is how the Ridge regression technique prevents coefficients from rising too high. Also, notice that we shrink the estimated association of each variable with the response, except the intercept β0, This intercept is a measure of the mean value of the response when xi1 = xi2 = …= xip = 0.
* When λ = 0, the penalty term has no eﬀect, and the estimates produced by ridge regression will be equal to least squares.
* However, as λ→∞, the impact of the shrinkage penalty grows, and the ridge regression coeﬃcient estimates will approach zero.
* As can be seen, selecting a good value of λ is critical. Cross validation comes in handy for this purpose. The coefficient estimates produced by this method are also known as the L2 norm.
* The coefficients that are produced by the standard least squares method are scale equivariant, i.e. if we multiply each input by c then the corresponding coefficients are scaled by a factor of 1/c.
* Therefore, regardless of how the predictor is scaled, the multiplication of predictor and coefficient(Xjβj) remains the same. However, this is not the case with ridge regression, and therefore, we need to standardize the predictors or bring the predictors to the same scale before performing ridge regression. The formula used to do this is given below.



**Lasso**



* Lasso is another variation, in which the above function is minimized. Its clear that this variation differs from ridge regression only in penalizing the high coefficients.
* It uses |βj|(modulus)instead of squares of β, as its penalty. In statistics, this is known as the L1 norm.
* Lets take a look at above methods with a different perspective. The ridge regression can be thought of as solving an equation, where summation of squares of coefficients is less than or equal to s.
* And the Lasso can be thought of as an equation where summation of modulus of coefficients is less than or equal to s. Here, s is a constant that exists for each value of shrinkage factor λ.
* These equations are also referred to as constraint functions.
* Consider there are 2 parameters in a given problem. Then according to above formulation, the ridge regression is expressed by β1² + β2² ≤ s.
* This implies that ridge regression coefficients have the smallest RSS(loss function) for all points that lie within the circle given by β1² + β2² ≤ s.
* Similarly, for lasso, the equation becomes,|β1|+|β2|≤ s. This implies that lasso coefficients have the smallest RSS(loss function) for all points that lie within the diamond given by |β1|+|β2|≤ s.
* The image below describes these equations.



* The above image shows the constraint functions(green areas), for lasso(left) and ridge regression(right), along with contours for RSS(red ellipse).
* Points on the ellipse share the value of RSS. For a very large value of s, the green regions will contain the center of the ellipse, making coefficient estimates of both regression techniques, equal to the least squares estimates. But, this is not the case in the above image.
* In this case, the lasso and ridge regression coefficient estimates are given by the ﬁrst point at which an ellipse contacts the constraint region.
* Since ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coeﬃcient estimates will be exclusively non-zero.
* However, the lasso constraint has corners at each of the axes, and so the ellipse will often intersect the constraint region at an axis.
* When this occurs, one of the coeﬃcients will equal zero. In higher dimensions(where parameters are much more than 2), many of the coeﬃcient estimates may equal zero simultaneously.
* **This sheds light on the obvious disadvantage of ridge regression, which is model interpretability.**
* It will shrink the coefficients for least important predictors, very close to zero. But it will never make them exactly zero. In other words, the final model will include all predictors.
* However, in the case of the lasso, the L1 penalty has the eﬀect of forcing some of the coeﬃcient estimates to be exactly equal to zero when the tuning parameter λ is suﬃciently large. **Therefore, the lasso method also performs variable selection and is said to yield sparse models.**

#### What does Regularization achieve?

* A standard least squares model tends to have some variance in it, i.e. this model won’t generalize well for a data set different than its training data.
* Regularization, significantly reduces the variance of the model, without substantial increase in its bias.
* So the tuning parameter λ, used in the regularization techniques described above, controls the impact on bias and variance.
* As the value of λ rises, it reduces the value of coefficients and thus reducing the variance. Till a point, this increase in λ is beneficial as it is only reducing the variance(hence avoiding overfitting), without loosing any important properties in the data.
* But after certain value, the model starts loosing important properties, giving rise to bias in the model and thus underfitting.
* Therefore, the value of λ should be carefully selected.
* This is all the basic you will need, to get started with Regularization. It is a useful technique that can help in improving the accuracy of your regression models.

**Ensemble learning and Random Forests**

**Voting Classifier:**

A Voting Classifier is a machine learning model that trains on an ensemble of numerous models and predicts an output (class) based on their highest probability of chosen class as the output.  
It simply aggregates the findings of each classifier passed into Voting Classifier and predicts the output class based on the highest majority of voting. The idea is instead of creating separate dedicated models and finding the accuracy for each them, we create a single model which trains by these models and predicts output based on their combined majority of voting for each output class.

Voting Classifier supports two types of votings.

Hard Voting: In hard voting, the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers. Suppose three classifiers predicted the output class(A, A, B), so here the majority predicted A as output. Hence A will be the final prediction.

Soft Voting: In soft voting, the output class is the prediction based on the average of probability given to that class. Suppose given some input to three models, the prediction probability for class A = (0.30, 0.47, 0.53) and B = (0.20, 0.32, 0.40). So the average for class A is 0.4333 and B is 0.3067, the winner is clearly class A because it had the highest probability averaged by each classifier.

**Bagging and Pasting**

Bagging and pasting are techniques that are used in order to create varied subsets of the training data. The subsets produced by these techniques are then used to train the predictors of an ensemble.

Bagging, short for bootstrap aggregating, creates a dataset by sampling the training set with replacement. Pasting creates a dataset by sampling the training set without replacement.

Ensemble models perform best when the predictors they are made up of are very different from one another. When predictors differ greatly from one another it follows that the instances in which they make errors will also be different. This leads to higher accuracy for the ensemble, as it is less likely that the majority of the predictors in the ensemble will make the same error.

In order to attain a varied set of predictors one method is to use different training algorithms for each of the predictors. Bagging provides an alternate method for achieving diversity among the predictors in an ensemble. Each of the predictors is created using the same training algorithm, but is trained on a random subset of the training data that has been sampled with replacement.

Bagging is most powerful when used to form an ensemble of high variance predictors. High variance predictors are able to almost perfectly model the datasets that they are trained on, but generalize very poorly. When these predictors are used to form an ensemble, a much lower variance model is created that has comparable bias.

When bagging is used in order to create an ensemble of high bias models the ensemble will generally have comparable performance to the individual predictors.

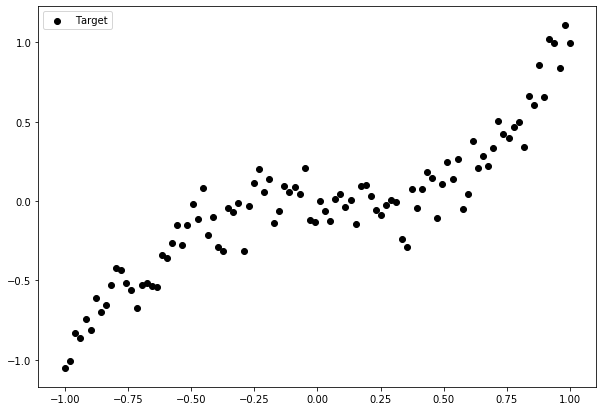
This is due to the fact that bagging does not decrease bias. The high bias models do not possess the complexity required at adequately model the data and thus an ensemble of these predictors will also be unable to encapsulate the complexity within the data. In most cases the resulting ensemble will not have the desired performance.

###### How Does Bagging and Pasting Work To Create A Model?

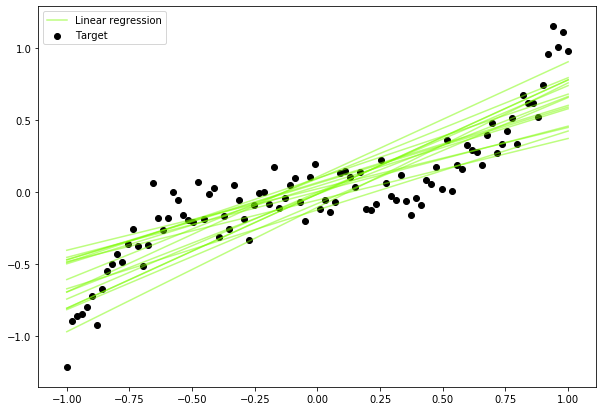
Both bagging and pasting are very simple to implement. For pasting each dataset is created by randomly sampling the chosen number of samples from the training set without replacement. For bagging each dataset is created by randomly sampling the chosen number of samples from the dataset with replacement. A separate predictor is then trained on each of the created datasets. These predictors are then used to form an ensemble.

###### **The Effect of Bagging on Variance.**

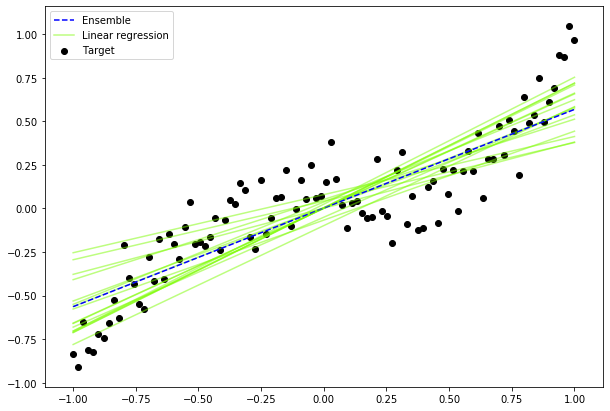
A fantastic article on this topic is available [here](https://towardsdatascience.com/understanding-the-effect-of-bagging-on-variance-and-bias-visually-6131e6ff1385), I highly recommend it. The plots displayed below have been inspired by those found in the aforementioned article.



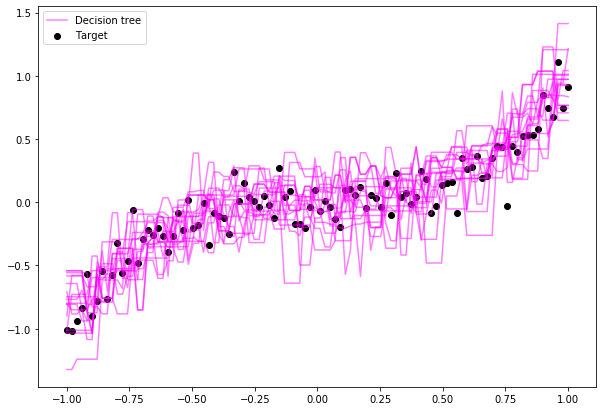
Let us consider the dataset displayed in figure one. We will begin by training fifteen linear regression models using bagging and plotting each of their predictions.



It is clear that the simple models are unable to adequately model the dataset as they do not possess the required complexity. Thus, when we aggregate the predictions of all of these models (as ensemble methods do) the result would still not capture the complexity of the dataset.

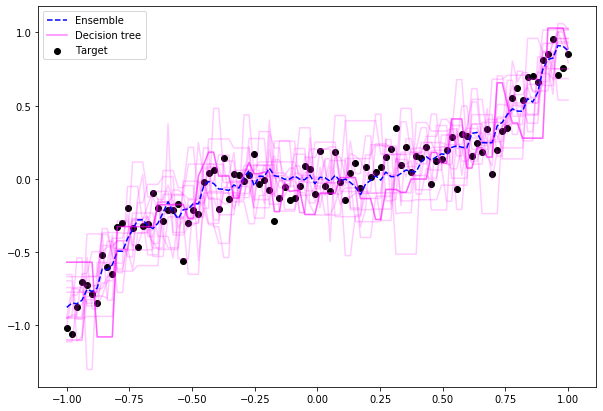


This illustrates why when bagging is used to create an ensemble of high bias models, the ensemble will not generally outperform the individual predictors. Bagging creates an ensemble with lower variance than the predictors it is made up of, thus it is not able to overcome the inability of the high bias models to capture the complexity of the dataset.To illustrate when bagging is useful, we will train fifteen decision trees and plot their predictions.

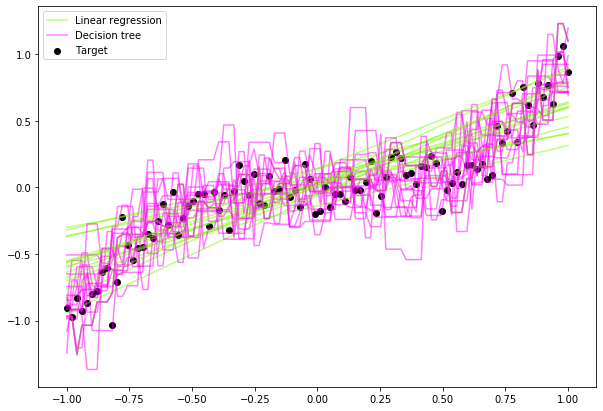


Notice how closely each of the decision trees fits the dataset that it was trained on. This is characteristic of a high variance model. The decision trees are able to capture the complexity of the dataset, but generalize poorly. In order to overcome this the decision trees are used to form an ensemble. This results in a model with much lower variance than the individual decision trees but with comparable bias.

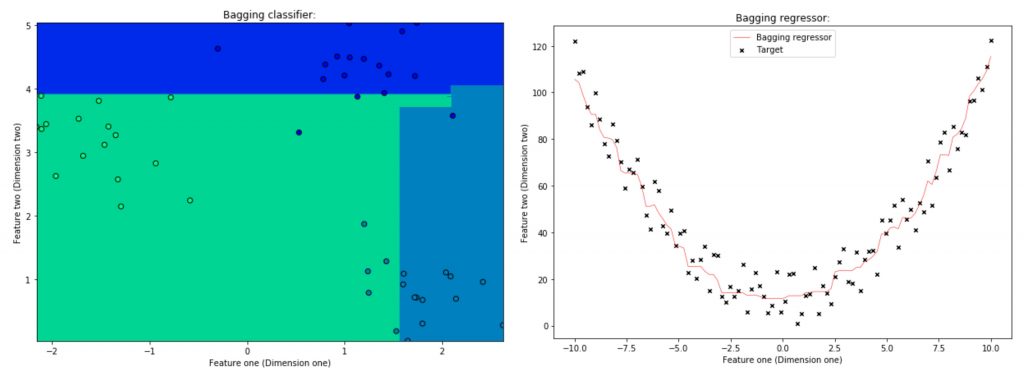
This may lead to the question, “Why not simply train multiple decision trees on the full training set?” If this was done the predictors in the ensemble would all be almost or completely identical. Thus, all of the predictors would make the same errors and the resulting ensemble would have high variance. This is why bagging is used, as it introduces diversity among the predictors. It is this diversity which leads to different predictors making different errors. In figure 3 this is illustrated in the predictions of the models; many predictors have relatively accurate predictions with few making large errors. When these predictors are used to form an ensemble, the resulting model will fit the dataset well while not modelling the noise.



The plot below shows the difference between the predictions of high variance and high bias models.



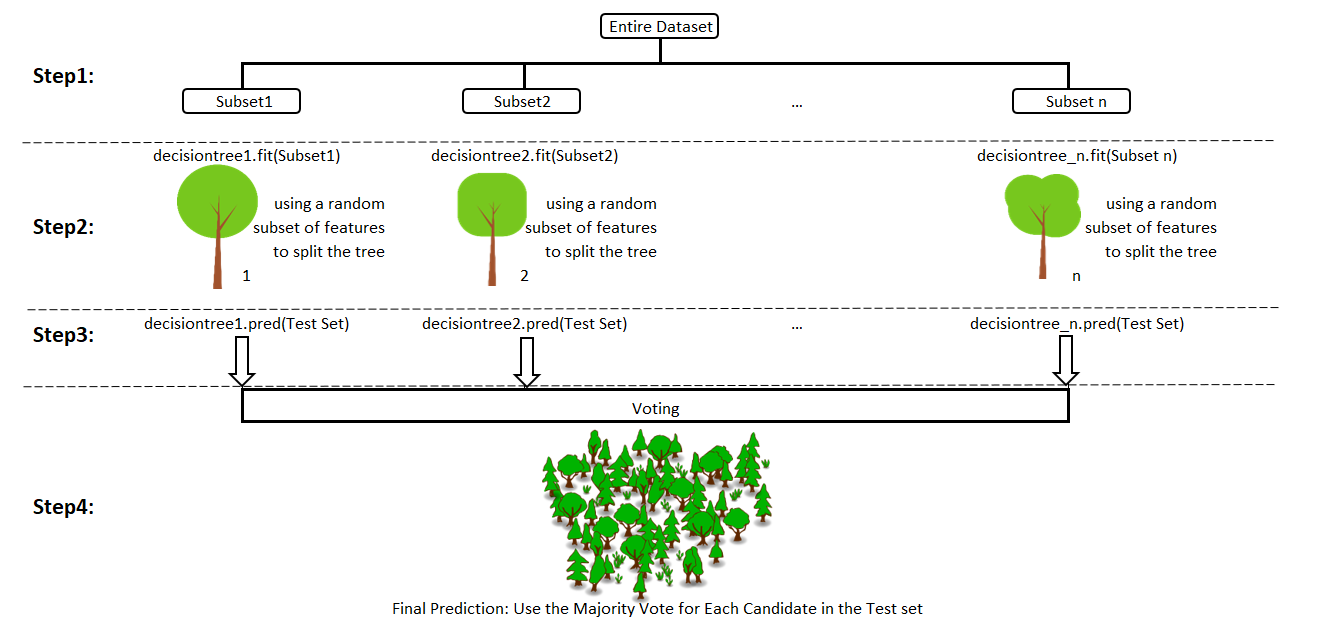
As each of the predictors in the ensemble is trained separately it is possible to train the predictors in parallel. This greatly decreases the time required to train an ensemble model as it can be trained using multiple processing cores or even servers if available.



**Random Forests**

Random forest is an ensemble model using bagging as the ensemble method and decision tree as the individual model.

Let’s take a closer look at **the magic🔮 of the randomness**:



Step 1: **Select n (e.g. 1000) random subsets** from the training set

Step 2: **Train n (e.g. 1000) decision trees**

* one random subset is used to train one decision tree
* the optimal splits for each decision tree are based on a random subset of features (e.g. 10 features in total, randomly select 5 out of 10 features to split)

Step 3: **Each individual tree predicts** the records/candidates in the test set, independently.

Step 4: **Make the final prediction**

For each candidate in the test set, Random Forest uses the class (e.g. cat or dog) with the**majority vote**as this candidate’s final prediction.

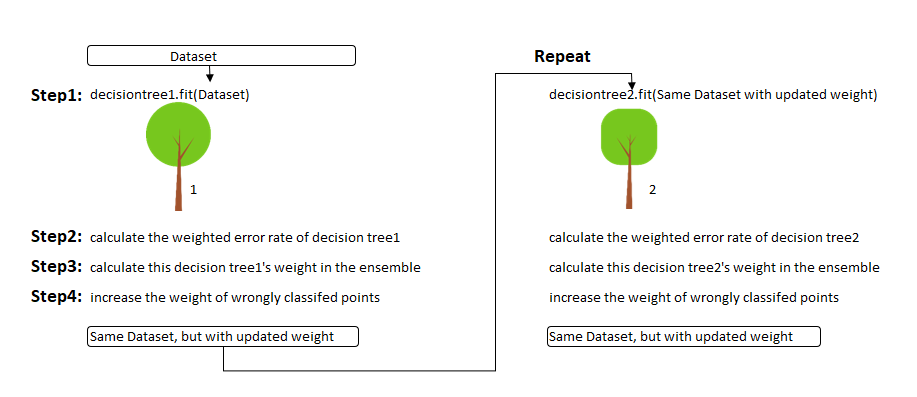
Of course, our 1000 trees are the parliament here.

# AdaBoost (Adaptive Boosting)

AdaBoost is a boosting ensemble model and works especially well with the decision tree. Boosting model’s key is learning from the previous mistakes, e.g. misclassification data points.

AdaBoost learns from the mistakes by increasing the weight of misclassified data points.

**AdaBoost adapts**.



Step 0: **Initialize the weights** of data points. if the training set has 100 data points, then each point’s initial weight should be 1/100 = 0.01.

Step 1: **Train**a decision tree

Step 2: **Calculate the weighted error rate (e)**of the decision tree. **The weighted error rate (e)** is just how many wrong predictions out of total and you treat the wrong predictions differently based on its data point’s weight. **The higher the weight**,**the more the corresponding error will be weighted**during the calculation of the (e).

Step 3: **Calculate this decision tree’s weight**in the ensemble

the weight of this tree = learning rate \* log( (1 — e) / e)

* the higher weighted error rate of a tree, 😫, the less decision power the tree will be given during the later voting
* the lower weighted error rate of a tree, 😃, the higher decision power the tree will be given during the later voting

Step 4: **Update weights** of wrongly classified points

the weight of each data point =

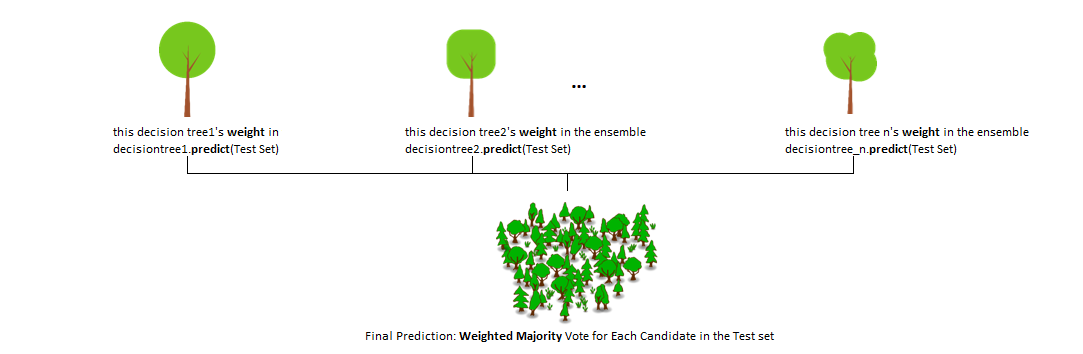
* if the model got this data point correct, the weight stays the same
* if the model got this data point wrong, the new weight of this point = old weight \* np.exp(weight of this tree)

Note: The higher the weight of the tree (more accurate this tree performs), the more boost (importance) the misclassified data point by this tree will get. The weights of the data points are normalized after all the misclassified points are updated.

Step 5: **Repeat**Step 1(until the number of trees we set to train is reached)

Step 6: **Make the final prediction**

The AdaBoost makes a new prediction by adding up the weight (of each tree) multiply the prediction (of each tree). Obviously, the tree with higher weight will have more power of influence the final decision.

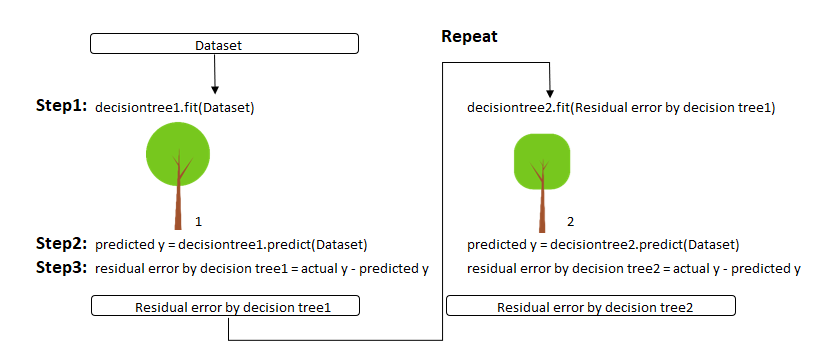


**Gradient Boosting**

Gradient boosting is another boosting model. Remember, boosting model’s key is learning from the previous mistakes.

Gradient Boosting learns from the mistake — residual error directly, rather than update the weights of data points.

Let’s illustrate **how Gradient Boost learns.**



Step 1: T**rain**a decision tree

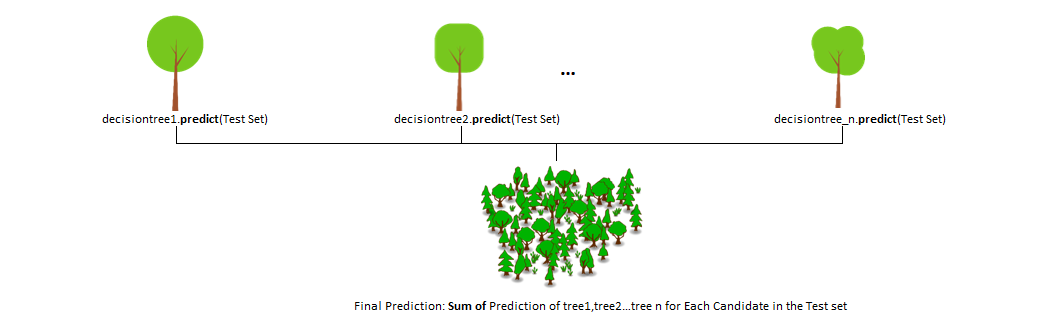
Step 2: **Apply**the decision tree just trained to predict

Step 3: **Calculate**the residual of this decision tree, Save residual errors as the new y

Step 4: **Repeat**Step 1 (until the number of trees we set to train is reached)

Step 5: **Make the final prediction**

The Gradient Boosting makes a new prediction by simply adding up the predictions (of all trees).



**Vanishing Gradients**

Vanishing gradients occur during the training of deep neural networks when the gradients of the loss function with respect to the network's parameters become extremely small as they are propagated backward through the network layers.

* **Activation Functions**

Some activation functions, like sigmoid and hyperbolic tangent (tanh), squash their inputs into a small range (0 to 1 for sigmoid, -1 to 1 for tanh). When gradients are backpropagated through many layers, they are repeatedly multiplied by values in this small range, causing them to diminish significantly.

* **Deep Networks**:

As the depth of a neural network increases, the number of multiplicative operations in the backpropagation process grows exponentially. If gradients are small to begin with, they can quickly approach zero.

Causes:

* **Slow Convergence**

Models with vanishing gradients tend to converge very slowly during training. This means it takes a long time for the network to learn and adapt to the data.

* **Difficulty in Training Deep Networks**

Vanishing gradients can make it extremely challenging to train very deep networks effectively. Without proper handling, deep networks may not learn meaningful representations.

**Exploding Gradients**

Exploding gradients are the opposite problem of vanishing gradients. They occur when gradients during backpropagation become excessively large as they are propagated backward through the network.

* **Initialization**

Poor weight initialization, especially if weights are initialized too large, can lead to exploding gradients. Large weights cause large gradient updates.

* **Activation Functions**

In some cases, activation functions like the Rectified Linear Unit (ReLU) can cause gradients to explode. ReLU has an unbounded positive range, which means it doesn't inherently limit the growth of activations and gradients.

**Consequences:**

* **Divergence**

Models with exploding gradients diverge during training. This means the optimization process becomes unstable, and the model's parameters may become extreme, leading to NaN (not-a-number) values.

* **Training Instability**

Exploding gradients can lead to training instability, making it challenging to find a suitable set of model parameters.

**Handling Vanishing and Exploding Gradients**

1. **Weight Initialization**

Proper weight initialization techniques like Xavier/Glorot initialization help control the variance of activations, reducing the likelihood of vanishing and exploding gradients.

1. **Activation Functions**

Using activation functions like ReLU can mitigate vanishing gradients compared to sigmoid and tanh.

1. **Batch Normalization**

Batch normalization normalizes activations within each layer, which can stabilize gradient flow and mitigate vanishing gradients.

1. **Gradient Clipping**

In the case of exploding gradients, gradient clipping can be employed to set a threshold on gradient values during training.

1. **Skip Connections**

Architectures like Residual Networks (ResNets) use skip connections to allow gradients to flow directly, addressing vanishing gradient issues in deep networks.