and PCA

Nitin Kishore Follow
Jun 2, 2018 · 11 min read

In the current deep learning frenzy there might be less focus on some of the well known methods albeit these are very useful for minor machine learning projects that one might work on. This blog post aims at covering the important concepts and techniques related to the following methods

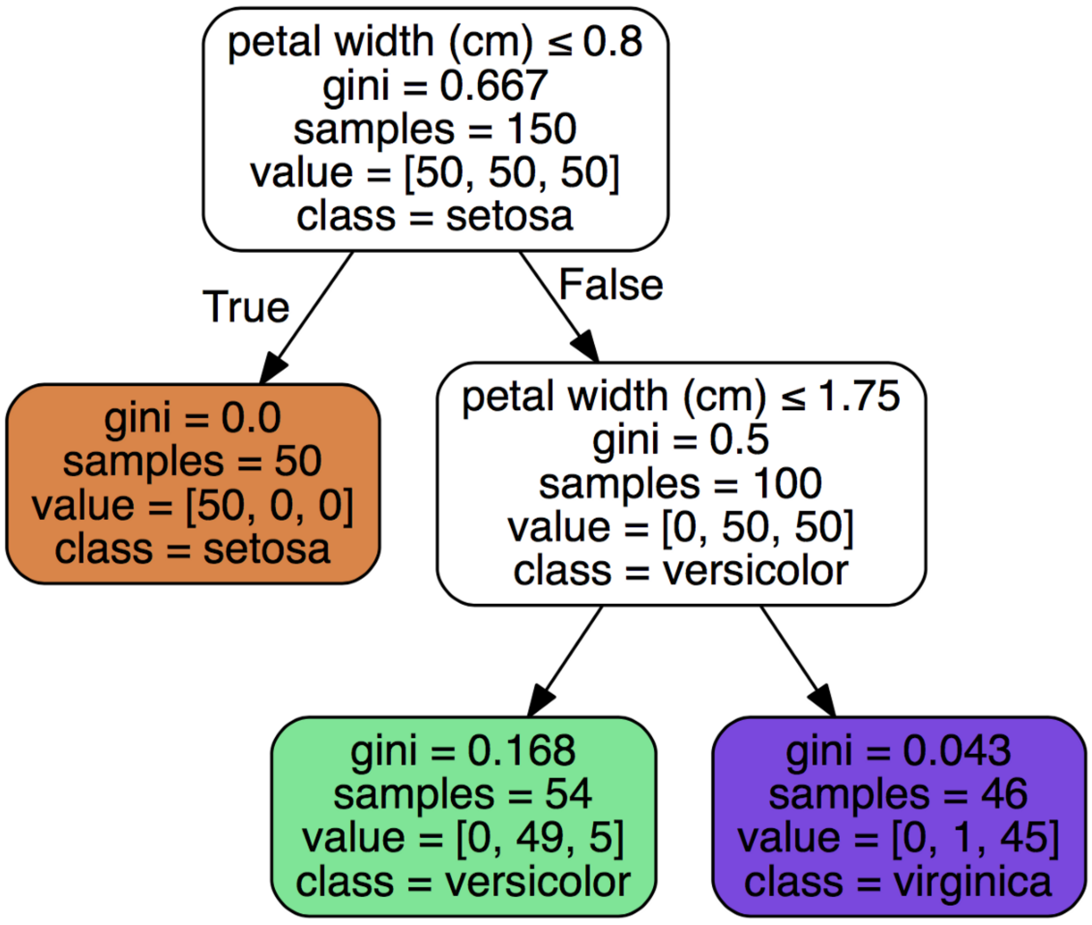
- Decision Trees
- Ensemble Learning (Random Forests)
- Curse of Dimensionality

The content of this post is a summary of the important points that I grasped from the chapters of the reference book, for the purpose of a quick review.

That's a nice decision boundary separating the green and yellow classes

Decision Trees

These are very versatile "non-parametric white-box models" that don't require feature scaling or centering and are capable of performing both Classification and Regressions tasks. You might have heard of neural networks as being black box models since you can't see what goes on under the hood while learning or inference. In contrast, a white box model allows you to see exactly what goes on at each step, enabling you to perform the task manually if it suits you. Decision trees make very few assumptions about the training data and if left unconstrained they would adapt themselves to the data most likely overfit.



Decision tree of depth 2 for iris dataset

The Iris data set is a very popular as an introductory example to basic machine learning. This example, wonderfully explained in the book mentioned in the references, serves to explain the process of using a decision tree. The learning algorithm for decision trees has an apropos name, C.A.R.T (*Classification and Regression Tree)* which generates only binary trees. For non binary trees, algorithms like *ID3* can be used.

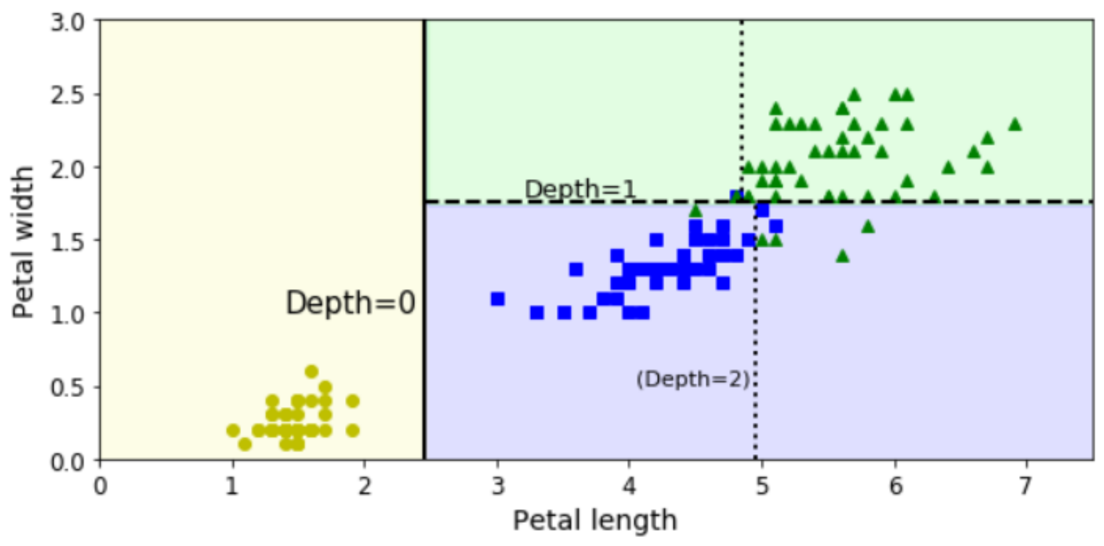
The root node, at depth 0 ,contains the training set. Based on a particular threshold for a specific feature, the samples are split into 2 subsets and this process continues recursively with the objective being to minimize the weighted average impurity of the child nodes. This is a greedy algorithm and tries to do an optimum split at the top or current level and doesn't concern itself with having minimum impurity at lower levels

$$G_i = 1 - \sum_{k=1}^{n} (p_{i,k})^2$$

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}$$

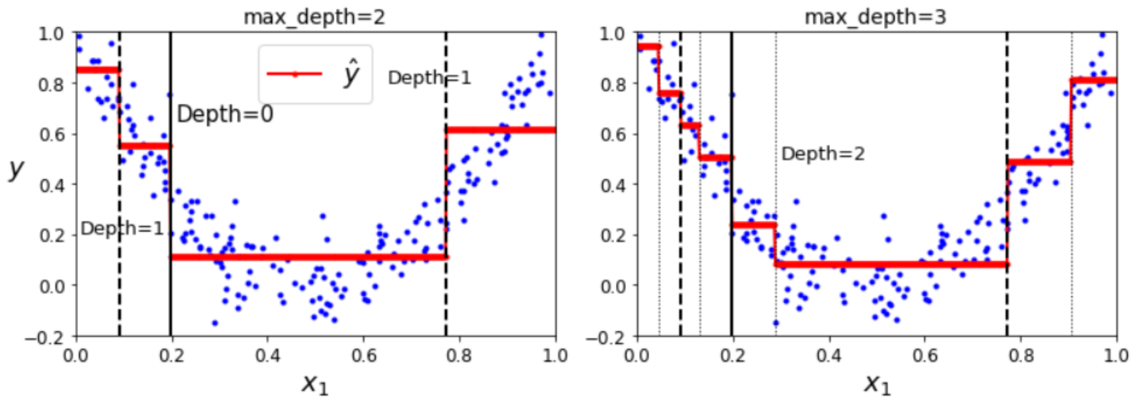
Gini impurity and Objective function for CART. K is a feature and tk is the threshold for that feature

The *gini* attribute denotes the purity of a node. If all the training samples a node applies to, belong to the same class, then that node is pure and has gini impurity of 0. We can also use *entropy* as a measure of purity. A set's entropy will be zero when it contains instances of only one class. Gini impurity is preferred due to it's appealing factor of isolating the most frequent class into a separate branch of the tree and it's ease and speed of computation where as entropy produces more balanced trees and is not the default option.



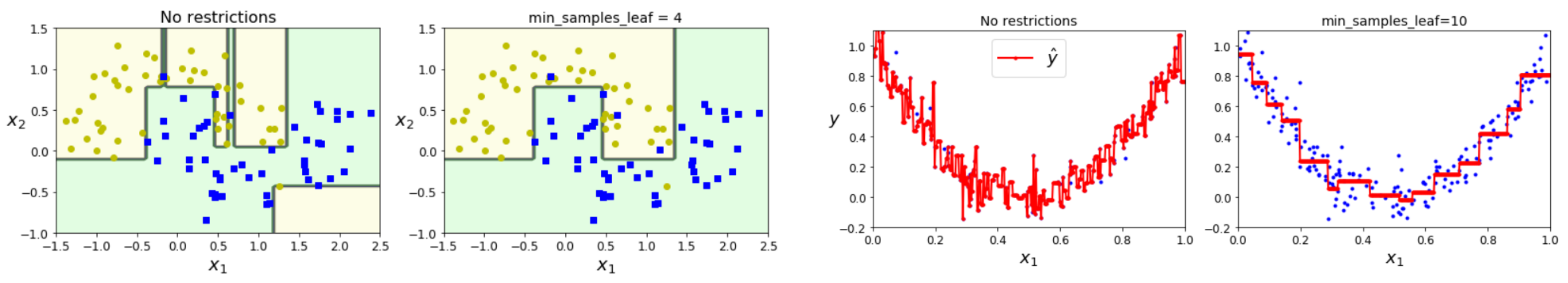
Classification : Decision boundaries at each depth

In case of regression, instead of predicting a class, the prediction is simply the average value of training instances, associated with a particular node, which is used to get the mean squared error. CART algorithm splits each region in a way that makes most training instances as close as possible to the predicted value. We minimize the MSE instead of the gini impurity in this case. Everything else remains the same.



Regression : Decision Trees

Decision trees don't have any predetermined number of parameters and is unrestricted with many degrees of freedom. One way of constraining or regularizing the tree, is to set a max depth constraint. They may have several hyper parameters like *max_depth* or *min_samples_split*. In general, reducing the max_* and increasing the min_* hyper parameters will regularize the model sufficiently.

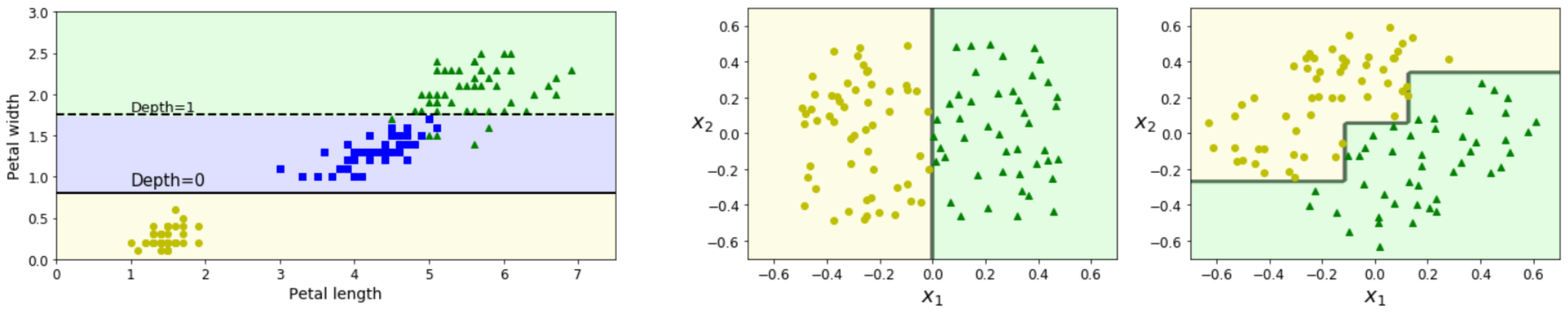


What happens when you don't regularize, in classification and regression?

Instability issues

Finding an optimal tree is a NP-Complete problem and since making predictions involves traversing the trees the overall prediction complexity is O(log(m)/log(2)) which is the number of nodes and this is independent of the number of features n. CART however trains by comparing all features and finding thresholds for all samples at each node so decision trees have a time complexity of O(n*m*log(m)/log(2))

As you can notice, the decision boundaries for splits are orthogonal (perpendicular to an axis) and the main issue with decision treees is that they arre very sensitive to variations in training data.



(left)Shows sensitivity to training set details , (right) Shows sensitivity to training set rotation

If the training algorithm is stochastic, we can get very different models even if the underlying data is the same. This instability can be addressed by averaging predictions over many trees which is the core concept of ***Random Forests.***

NOTE : A *Decision stump* is a decision tree with depth 1. It has a single decision node and two leaf nodes

Ensemble Learning

What you would call, the wisdom of the crowd

It is well known in the machine learning field that, using ensemble learning methods can increase the accuracy of your model by 2–3%.

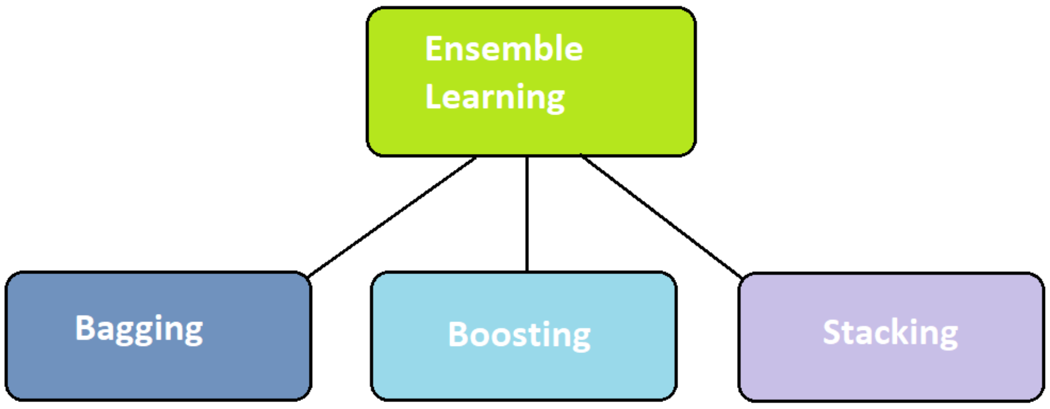
> *An ensemble of decision trees makes a random forest.*

Ensemble here just means a group of predictors(or classifiers). Predicting the class that gets most votes , by aggregating the predictions of a bunch of classifiers(even if they are weak ie., making random guesses), is surprisingly way better than any other individual strong classifiers and this is called a ***hard-voting classifier***. This is the same as choosing the statistical mode (most frequent prediction). If classifiers can estimate class probabilities then, predicting class with highest probability, averaged over all individual classifiers in the ensemble is called a ***soft-voting classifier***. Naturally this gives better results than hard voting, as it gives more weight or importance to highly confident votes.

To understand why this sort of method works so well, we can draw a parallel to the ***law of large numbers***. Assume you have a biased coin which turns up heads 51% of the times you toss it. After tossing it a hundred times it is reasonable to assume that we may have 51 heads and 49 tails. We have more heads than tails. In a 1000 tosses, the difference in number would only increase as we would have 20 more heads than tails(510–490). By the same logic, as we toss it more the probability of having more heads than tails keeps increasing, making sure that the ratio of heads is always close to 51%.

As common sense dictates, this will only work if all the weak classifiers are independent of each other and sufficiently diverse. It wouldn't be of much use aggregating or voting on the predictions, if all the weak classifiers make the same type of errors. To improve the accuracy of the ensemble, it is essential to have uncorrelated errors but this is very difficult , unless you train classifiers using different methods, since the common denominator is the training set. A group of decision trees is a Random Forest, but what exactly is the source of randomness when all the classifiers are the same and how does it have a better accuracy if all the decision trees are trained on the same data?

One way to make ensemble methods work is to either use very different training algorithms for the individual classifiers so that their predictions are diverse or have different errors, but since a random forest comprises of only decision trees, we need to introduce randomness in the data itself. the fact that decision trees are sensitive to changes in the training data works to our advantage point.



Broadly divided into three categories of techniques

So we use the same learning approach for all classifiers but limit their training data to random subsets. Sampling this subset of data can happen either with or without replacement. In statistics, re-sampling with replacement is called *bootstrapping*. Sampling without replacement is called *pasting*.

* Bagging

> *Bagging is short for bootstrap aggregation*

Bagging allows training instances to be sampled several times for each predictor and introduces that diversity in those subsets. The ensemble now makes predictions for new instances by aggregating the individual predictions and even though each predictor has a higher bias than it would if it was trained on the whole dataset, the aggregation reduces both the bias and the variance as the predictors are less correlated. The bagging process need not be limited to just the instances. There are two methods:

- ***Random Subspaces***—Sampling only features
- ***Random Patches***—Sampling both training instances and features
- ***Extra Trees—Extremely Random trees***. These are faster to train since we set random thresholds for the features instead of finding it but we can't be sure before hand if this does better than the alternative.

Since bagging samples several instances multiple times, it's not uncommon for few instances to never be sampled. Each predictor now has a different set of unseen *out of bag* instances that can be used as a validation set and ensemble can be evaluated by averaging the oob evaluations of each predictor.

Another source of randomness in Random forests is that , when growing trees you search for the best features among a random subset instead of optimizing to find the best for splitting a node. In this method you trade high bias for lower variance and gets a better model overall.

NOTE: An important aspect of random forests is that they can help in **feature selection**. The importance of a feature it determined by how much the tree nodes use it to reduce impurity on average. This average is weighted by the number of samples associated with each node

* Boosting

This is another form of ensemble learning that is sequential in nature. The idea is still to combine several weak learners and form a strong learner but here the predictors are trained sequentially , each trying to correct or improve the previous predictor in the pipeline. Naturally this process cannot be parallelized so it isn't used often as it doesn't scale as well as bagging or pasting. The two most popular boosting methods are

- ***Adaboost(Adaptive boosting)***—the relative weight of the misclassified instances is increased and the second predictor is trained using these updated weights and this process happens again for the next predictor. This sequential learning does resemble SGD, with the difference being that instead of tweaking a single predictors parameters we add a predictor to the ensemble and gradually make it better.

- ***Gradient boosting***—Instead of tweaking the instance weights at each iteration, this method fits the new predictor to the residual errors made by the previous one. The learning rate hyper parameter will scale the contribution of each tree or predictor. This is a regularization technique called ***shrinkage***. If you set it to a low value, then you need more trees in the ensemble to fit the training set which gives a better prediction accuracy. The sub-sample hyper-parameter on the other hand specifies the fraction of training instances used to train each tree and this is called ***Stochastic Gradient Boosting.***

NOTE: To find the optimal number of trees in boosting , we use **early stopping.**

* Stacking

Stacking is short for *Stacked Generalization*. The premise is that instead of trivial functions like mode or average to aggregate the individual predictions, we can train a model to perform the aggregation or learn from them. This model is called a ***blender*** or a ***meta-learner***.

The common approach to train this blender is to:

1. Split the data into 2 subsets and use the first split to train the individual predictors in the ensemble using any methods.
2. Make individual predictions on this held out data split.
3. Now we have a new dataset with individual predictions as our input and the initial target values from the held out data.
4. Train blender on this data

This process can be done by making multiple splits and using multiple blenders as well.

Curse of Dimensionality

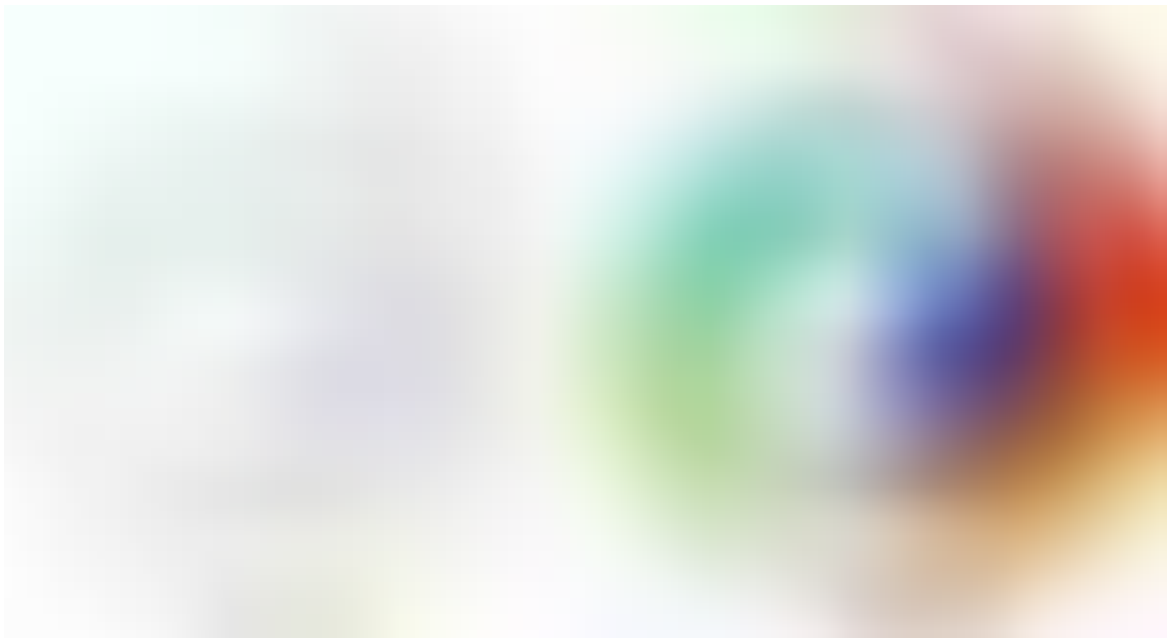
You must have undoubtedly come across this term several times by now. It becomes difficult to train algorithms when handling data with numerous dimensions and we always fear that reducing the dimensions might cause the loss of some important information. On the other hand, it enables us to train faster and also visualize the data, so it is a compromise most people are willing to make. High dimensional data often implies that the data is sprase or spread out. There are two main approaches to dimensionality reduction

- Projection
- Manifold Learning

In real world data, training instances are not uniformly spread out across all dimensions and it is very much possible for all of them to lie on a lower dimension subspace. Projection is nothing but representing the data on the lower dimension.

> *A **manifold** is a topological space that locally resembles Euclidean space near each point.*

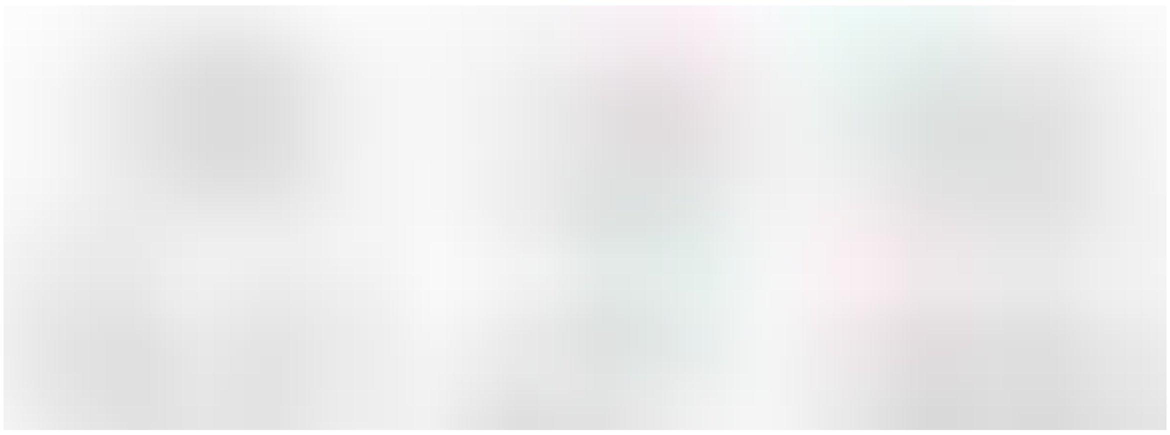
A 2D manifold is a 2D shape that can be twisted and bent in a higher dimensional space. Most real world high dimensional datasets lie close to a much lower-dimensional manifold. This is called the manifold hypothesis.



Swiss roll is an example of a 2D manifold. It is rolled in 3D

In general, reducing the dimensions reduces training time but it depends entirely on the dataset, for this to lead to better solutions.

The most popular dimensionality reduction algorithm is the **Principal Component Analysis** which identifies the hyper-plane closest to the data and projects the data onto it. The constraint here is that , to choose the right hyperplane, that provides minimum loss of information, we need to preserve the maximum amount of variance we can. The principal components are unit vectors defining axes that are othogonal to each other(as many as the dimensions in the data), with the first one accounting for the maximum variance. Their directions however are unstable, because slight perturbations in the data can cause the a pair of PCs to point in the opposite direction and even rotate or swap. The plane they define will mostly be the same .However PCA assumes the data is centered around the origin, so a prerequisite for this method is to center the data. We can obtain these components from a matrix factorizing technique called **Singular Value Decomposition** and then choose to project the data onto the hyper-plane defined by the first d components, where d is a smaller number of dimensions than the actual data had. Choosing this value d, depends on how much variance needs to be preserved. For visualization however, d is usually 2 or 3.


Different methods of dimensionality reduction

Please share this with your friends and hit that claps (👏) button below to spread it around even more. Also add any other useful applications for these algorithms or how you used these to improve your model's performance, in the comments below!

. . .

References

1. **Hands-on machine learning with Scikit-Learn and TensorFlow** by **Aurélien Géron**
2. https://amunategui.github.io/simple-heuristics/index.html