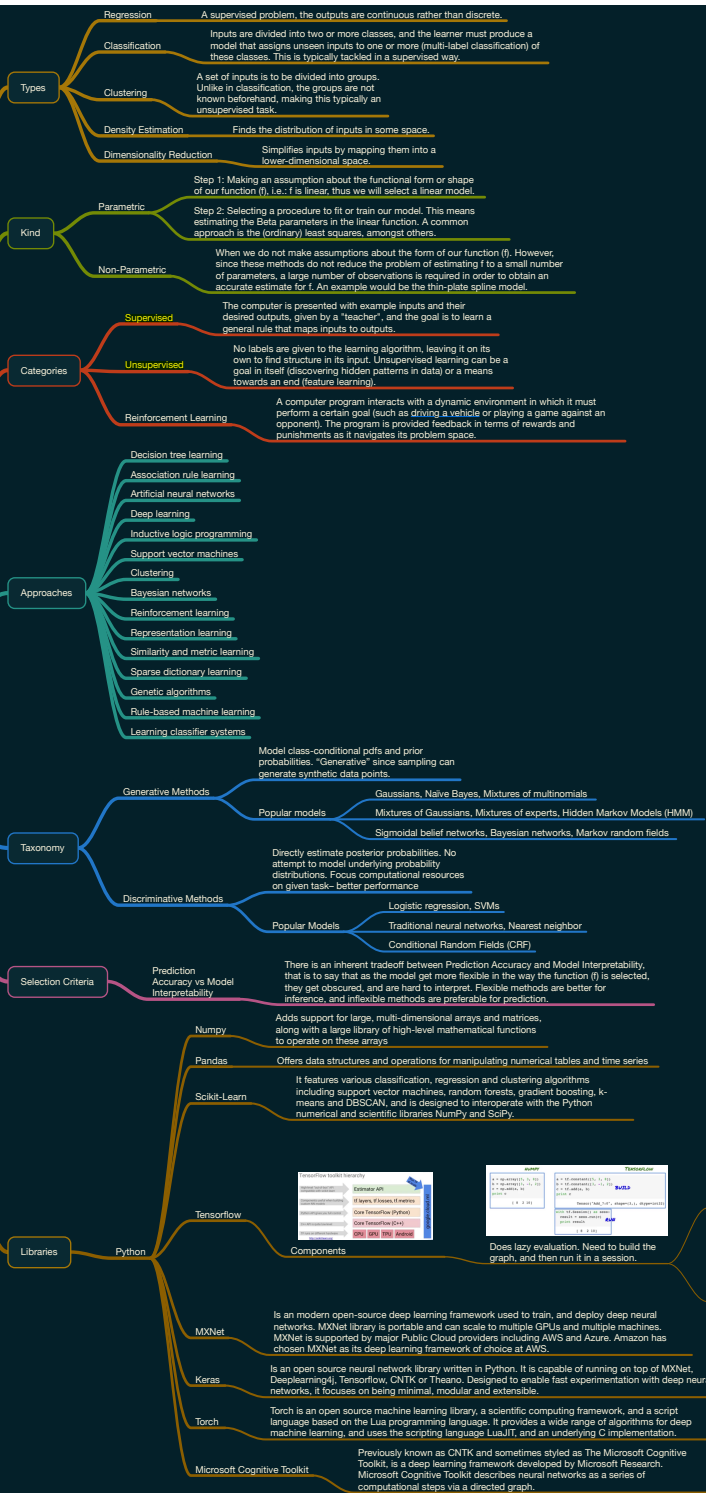


Machine Learning Concepts



Motivation

Prediction

When we are interested mainly in the predicted variable as a result of the inputs, but not on the each way of the inputs affect the prediction. In a real estate example, Prediction would answer the question of: Is my house over or under valued? Non-linear models are very good at these sort of predictions, but not great for inference because the models are much less interpretable.

Inference

When we are interested in the way each one of the inputs affect the prediction. In a real estate example, Inference would answer the question of: How much would my house cost if I had a view of the sea? Linear models are more suited for inference because the models themselves are easier to understand than their non-linear counterparts.

	Actual Positive	Actual Negative	Actual Unknown
Predicted Positive	True Positive (TP)	False Positive (FP)	Unknown Positive (UP)
Predicted Negative	False Negative (FN)	True Negative (TN)	Unknown Negative (UN)
Predicted Unknown	Unknown Positive (UP)	Unknown Negative (UN)	True Unknown (TU)

Confusion Matrix

Confusion Matrix

Fraction of correct predictions, not reliable as skewed when the data set is unbalanced (that is, when the number of samples in different classes vary greatly)

Accuracy

$(TP + TN) / (TP + FP + FN + TN)$

Precision

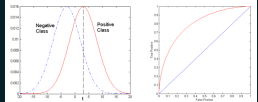
Out of all the examples the classifier labelled as positive, what fraction were correct?

Recall

Out of all the positive examples there were, what fraction did the classifier pick up?

F1 score

Harmonic Mean of Precision and Recall: $2 * p * r / (p + r)$



Performance Analysis

Bias-Variance Tradeoff

Bias refers to the amount of error that is introduced by approximating a real-life problem, which may be extremely complicated, by a simple model. If Bias is high, and/or if the algorithm performs poorly even on your training data, try adding more features, or a more flexible model.

Variance is the amount our model's prediction would change when using a different training data set. High: Remove features, or obtain more data.

Goodness of Fit = R^2

$1.0 - \text{sum of squared errors} / \text{total sum of squares}$

Mean Squared Error (MSE)

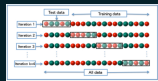
The mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors or deviations—that is, the difference between the estimator and what is estimated.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$$

Error Rate

The proportion of mistakes made if we apply out estimate model function to the training observations in a classification setting.

$$\frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$$



Cross-validation

Methods

- Leave-p-out cross-validation
- Leave-one-out cross-validation
- k-fold cross-validation
- Holdout method
- Repeated random sub-sampling validation

Grid Search

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.

Hyperparameters

Random Search

Since grid searching is an exhaustive and therefore potentially expensive method, several alternatives have been proposed. In particular, a randomized search that simply samples parameter settings a fixed number of times has been found to be more effective in high-dimensional spaces than exhaustive search.

Gradient-based optimization

For specific learning algorithms, it is possible to compute the gradient with respect to hyperparameters and then optimize hyperparameters via gradient descent. The first usage of these techniques was focused on neural networks. Since then, these methods have been extended to other models such as support vector machines or logistic regression.

Tuning

Early Stopping (Regularization)

Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit, and stop the algorithm then.

Overfitting

When a given method yields a small training MSE (or cost), but a large test MSE (or cost), we are said to be overfitting the data. This happens because our statistical learning procedure is trying too hard to find patterns in the data, that might be due to random chance, rather than a property of our function. In other words, the algorithms may be learning the training data too well. If model overfits, try removing some features, decreasing degrees of freedom, or adding more data.

Underfitting

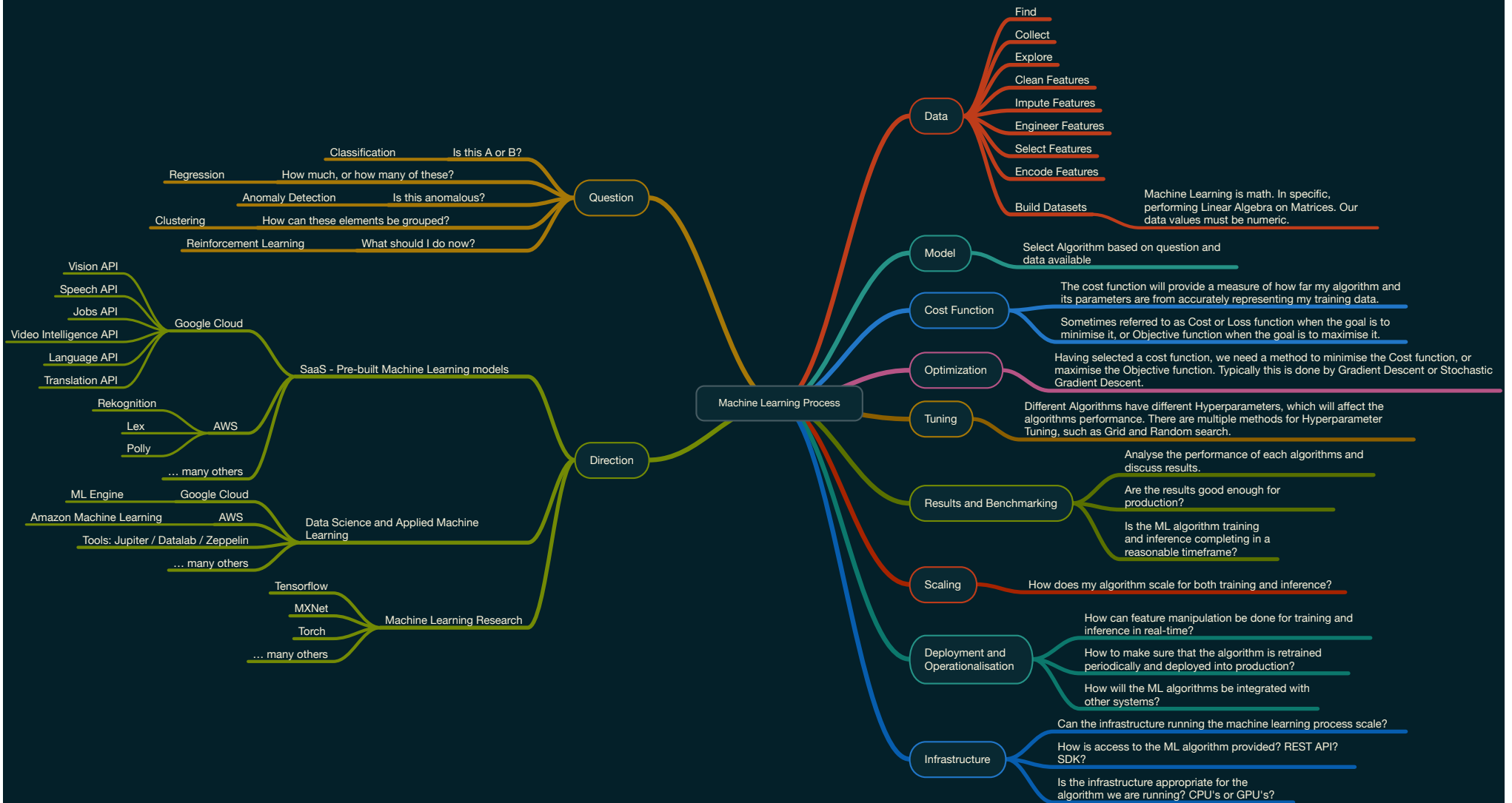
Opposite of Overfitting. Underfitting occurs when a statistical model or machine learning algorithm cannot capture the underlying trend of the data. It occurs when the model or algorithm does not fit the data enough. Underfitting occurs if the model or algorithm shows low variance but high bias (to contrast the opposite, overfitting from high variance and low bias). It is often a result of an excessively simple model.

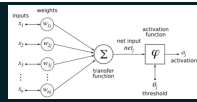
Bootstrap

Test that applies Random Sampling with Replacement of the available data, and assigns measures of accuracy (bias, variance, etc.) to sample estimates.

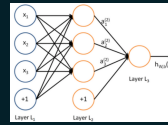
Bagging

An approach to ensemble learning that is based on bootstrapping. Shortly, given a training set, we produce multiple different training sets (called bootstrap samples), by sampling with replacement from the original dataset. Then, for each bootstrap sample, we build a model. The results in an ensemble of models, where each model votes with the equal weight. Typically, the goal of this procedure is to reduce the variance of the model of interest (e.g. decision trees).





A unit often refers to the activation function in a layer by which the inputs are transformed via a nonlinear activation function (for example by the logistic sigmoid function). Usually, a unit has several incoming connections and several outgoing connections.



Comprised of multiple Real-Valued inputs. Each input must be linearly independent from each other.

Layers other than the input and output layers. A layer is the highest-level building block in deep learning. A layer is a container that usually receives weighted input, transforms it with a set of mostly non-linear functions and then passes these values as output to the next layer.

With SGD, the training proceeds in steps, and at each step we consider a mini-batch $x_{i:m}$ of size m . The mini-batch is used to approximate the gradient of the loss function with respect to the parameters.

Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than in computations for individual examples, due to the parallelism afforded by the modern computing platforms.

However, if you actually try that, the weights will change far too much each iteration, which will make them "overcorrect" and the loss will actually increase/diverge. So in practice, people usually multiply each derivative by a small value called the "learning rate" before they subtract it from its corresponding weight.

Neural networks are often trained by gradient descent on the weights. This means at each iteration we use backpropagation to calculate the derivative of the loss function with respect to each weight and subtract it from that weight.

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J_t(\theta)$$

Simplest recipe: keep it fixed and use the same for all parameters.

$$\alpha = \frac{\alpha_0 \tau}{\max(t, \tau)}$$

Reduce by 0.5 when validation error stops improving

Reduction by $O(1/\tau)$ because of theoretical convergence guarantees, with hyperparameters ϵ_0 and τ and t is iteration numbers

Better yet: No hand-set learning of rates by using AdaGrad

Better results by allowing learning rates to decrease Options:

The implementation for weights might simply drawing values from a normal distribution with zero mean, and unit standard deviation. It is also possible to use small numbers drawn from a uniform distribution, but this seems to have relatively little impact on the final performance in practice.

Thus, you still want the weights to be very close to zero, but not identically zero. In this way, you can random these neurons to small numbers which are very close to zero, and it is treated as symmetry breaking. The idea is that the neurons are all random and unique in the beginning, so they will compute distinct updates and integrate themselves as diverse parts of the neural network.

This ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence. The detailed derivations can be found from Pages 18 to 23 of the slides. Please note that, in the derivations, it does not consider the influence of ReLU neurons.

One problem with the above suggestion is that the distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs. It turns out that you can normalize the variance of each neuron's output to 1, by scaling its weight vector by the square root of its fan-in (i.e., its number of inputs)

$$J = \max(0, 1 - s + \kappa_s)$$

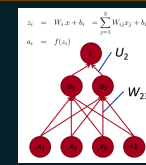
$s = U^T f(Wx + b)$
 $s_0 = U^T f(Wx_0 + b)$

$$s = U^T f(Wx + b) \quad x \in \mathbb{R}^{20 \times 1}, W \in \mathbb{R}^{8 \times 20}, U \in \mathbb{R}^{8 \times 1}$$

$$s = U^T a$$
$$a = f(s)$$
$$z = Wx + b$$

Neural Network taking 4 dimension vector representation of words

$$\frac{\partial s}{\partial x_j} = \sum_{i=1}^K \frac{\partial s}{\partial a_i} \frac{\partial a_i}{\partial x_j} = \sum_{i=1}^K \frac{\partial (U^T a)}{\partial a_i} \frac{\partial a_i}{\partial x_j} = \sum_{i=1}^K U_i \frac{\partial f(Wx + b)}{\partial x_j} = \sum_{i=1}^K U_i f'(Wx + b) \frac{\partial W_{ij} x}{\partial x_j} = \sum_{i=1}^K U_i f'(Wx + b) W_{ij}$$
$$w_{ij} = \frac{\partial s}{\partial x_j} W_{ij}$$



In this method, we reuse partial derivatives computed for higher layers in lower layers, for efficiency.

Defines the output of that node given an input or set of inputs.

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

ReLU

$$f(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid / Logistic

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

Binary

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Tanh

$$f(x) = \ln(1 + e^x)$$

Softplus

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^J e^{x_j}} \quad \text{for } i = 1, \dots, J$$

Softmax

$$f(\vec{x}) = \max_i x_i$$

Maxout

Leaky ReLU, PReLU, RReLU, ELU, SELU, and others.

Neural Networks

Machine Learning Models

Regression

$$Y = \beta_0 + \beta_1 X + \epsilon$$
$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad (\text{data: } i = 1 \dots n)$$

Linear Regression

is a flexible generalization of ordinary linear regression that allows for response variables that have error distribution models other than a normal distribution. The GLM generalizes linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

Generalised Linear Models (GLMs)

Identity $\mu = \mathbf{X}\beta$

Inverse $\mu = (\mathbf{X}\beta)^{-1}$

$$\mu = \frac{\exp(\mathbf{X}\beta)}{1 + \exp(\mathbf{X}\beta)} = \frac{1}{1 + \exp(-\mathbf{X}\beta)}$$

Logit

Cost Function is found via Maximum Likelihood Estimation

Locally Estimated Scatterplot Smoothing (LOESS)

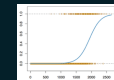
Ridge Regression

Least Absolute Shrinkage and Selection Operator (LASSO)

Logistic Regression

$$p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

Logistic Function



$$p(C_k | \mathbf{x}) = \frac{p(C_k) p(\mathbf{x} | C_k)}{p(\mathbf{x})}$$

Naive Bayes

Multinomial Naive Bayes

Bayesian Belief Network (BBN)

$\hat{y} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} p(C_k) \prod_{i=1}^n p(x_i | C_k)$

Naive Bayes Classifier. We neglect the denominator as we calculate for every class and pick the max of the numerator

Dimensionality Reduction

Principal Component Analysis (PCA)

Partial Least Squares Regression (PLSR)

Principal Component Regression (PCR)

Partial Least Squares Discriminant Analysis

Quadratic Discriminant Analysis (QDA)

Linear Discriminant Analysis (LDA)

Instance Based

k-nearest Neighbour (kNN)

Learning Vector Quantization (LVQ)

Self-Organising Map (SOM)

Locally Weighted Learning (LWL)

Random Forest

Classification and Regression Tree (CART)

Gradient Boosting Machines (GBM)

Conditional Decision Trees

Gradient Boosted Regression Trees (GBRT)

Decision Tree

Clustering

Hierarchical Clustering

Linkage

complete

single

average

centroid

Disimilarity Measure

Euclidean

Manhattan

Euclidean distance or Euclidean metric is the "ordinary" straight-line distance between two points in Euclidean space.

The distance between two points measured along axes at right angles.

How many clusters do we select?

k-Means

k-Medians

Fuzzy C-Means

Self-Organising Maps (SOM)

Expectation Maximization

DBSCAN

Data Structures Metrics

Dunn Index

Connectivity

Silhouette Width

Non-overlap APN

Average Distance AD

Average Distance Between Means ADM

Figure of Merit FOM

Stability Metrics