1 Supervised learning

Method where you train the program by feeding the learning algorithm with a mapping of inputs to correct outputs.

1.1 Regression

Regression is curve fitting: learn a continuous input $\rightarrow$ output mapping from a set of examples.

1.2 Classification

Outputs are discrete variables (category labels). Learn a decision boundary that separates one class from the other. Generally, a confidence is also desired, i.e., how sure are we that the input belongs to the chosen category.

1.3 Training set

The training set is a set of m $(\vec{X},\, y)$ pairs, where:

$$\begin{aligned} \vec{X} &\in \mathbb{R}^d && \text{models the input.} \\ y &\in \{0, 1\} && \text{models the output.} \end{aligned}$$

1.4 Error function

The loss function for a model $f : \vec{X} \mapsto y$ parameterized by $\vec{W}$ applied to a dataset $\{(\vec{X},\, y)\}$ of size m is:

$$L(\vec{W}) = \sum_{i}^{m} \left(f_{\vec{W}}(\vec{X}_i) - y_i \right)^2$$

1.5 Perceptron

Perceptron is the trivial neural network. The model for a parameter $\vec{W} = (\text{threshold},\, w_1,\, \ldots,\, w_d)$ and inputs of the form $(1,\, x_1,\, \ldots,\, x_d)$ is given by

$$f_{\vec{W}}(\vec{X}) = \text{sign}(\vec{W}\vec{X})$$

Where sign is the activation function.
If x_i is evidence for approval, then w_i should be high.
If x_i is evidence for denial, then w_i should be low.

1.5.1 Learning algorithm

The learning algorithm of the Perceptron is quite simple. The learning rate $\in (0,\, 1]$ is used to scale each step. the For a training set $S = \{\, (\vec{X}_1,\, y_1),\ \ (\vec{X}_2,\, y_2),\ \ \ldots \}$

- Starting with random weights, then show each sample in sequence repetitively.
- If the output is correct, do nothing.
- If the produced output is negative, and the correct output is positive, increase the weights.
- If the produced output is positive, and the correct output is negative, decrease the weights.
- The amount to increase/decrease is given by the current sample scaled by the learning rate.

1.6 Error

The error function for a model f in a **training** sample is

$$E_{\text{in}}(f)$$

This function is known and calculable.

The error function for a model f in a **hypothetical** sample is

$$E_{\text{out}}(f)$$

This function is **not** known, and only **approachable**.

A good approximation of E_{out}(f) is the error in a **test** (or **validation**) sample

$$E_{\text{val}}(f)$$

Given a model f in a set of M models, the bound for the probability of the error deviation surpassing a given ϵ is

$$\mathbb{P}\left(\left|E_{\text{in}}(f) - E_{\text{out}}(f)\right| > \epsilon\right) \leq 2Me^{-2N\epsilon^2}$$

Notably, $E_{\text{in}}(f)$ and $E_{\text{out}}(f)$ deviates as f becomes complex.

1.6.1 Empirical error minimization

During the learning algorithm, always conserve the weights that produce the lower error.
This has a disadvantage: It memorizes the training set.

1.7 Learning decision trees

Each layer in the tree consists of an attribute that splits the data into subsets that are ideally disjoint.
The entropy of the subsets produced is a measure of how disjoint they are.

For a set containing p positive and n negatives, the entropy is

$$H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log\left(\frac{p}{p+n}\right) - \frac{n}{p+n}\log\left(\frac{n}{p+n}\right)$$

A given attribute A, with k distinct values, divides the training set S into subsets $S_1, S_2, \ldots, S_k$.
The expected entropy remaining after applying A is

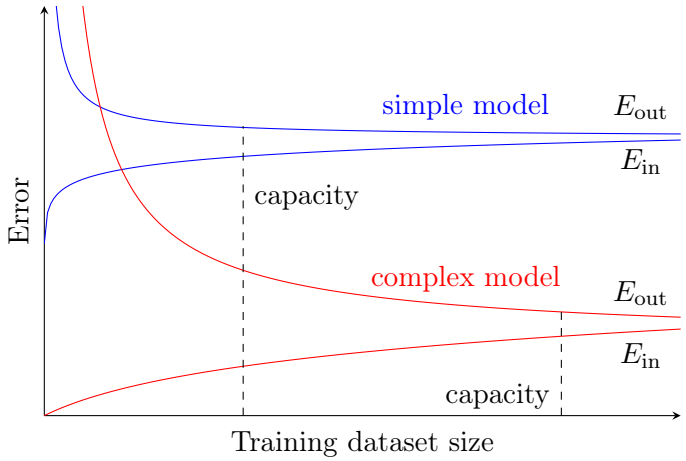
$$EH(A) = \sum_{i=1}^{k}\left[\frac{p_i + n_i}{p+n} \cdot H\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)\right]$$

The information gain, i.e. the reduction in entropy for A, is

$$I(A) = H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - EH(A)$$

1.8 Capacity

The capacity is a measure of when the training error is a good approximation for the test error.



1.9 Bias and variance

Bias is the error due to the fact that the set of functions does not contain the target function.

Variance is the error due to the fact that if we had been using another training set drawn from the same distribution, we would have obtained another function.

Regularization is a method for minimizing the training error, as long as it is still a good approximation for the test error, trading-off accuracy for simplicity.

1.10 Single layer neural networks

Using the sigmoid as the activation function, and the squared-error loss function:

$$L(\vec{W}) = \frac{1}{2}\sum_{i}^{m}\left(\sigma\left(\vec{W}\vec{X}_i\right) - y_i\right)^2$$

To find in which direction the weights minimizes L, the gradient is used:

$$\nabla L(\vec{W}) = \sum_{i}^{m} \Delta \cdot \Psi$$

Where the delta rule is

$$\Delta = \vec{X}_i \cdot \left(\sigma\left(\vec{W}\vec{X}_i\right) - y_i\right)$$

And the slope of ligistic is

$$\Psi = \sigma\left(\vec{W}\vec{X}_i\right) \cdot \left(1 - \sigma\left(\vec{W}\vec{X}_i\right)\right)$$

1.10.1 Gradient descent algorithm

The learning rate $r \in (0, 1]$ is used to scale each step.

1. Starting with random weights.
2. Compute $\nabla L(\vec{W})$.
3. $\vec{W} \leftarrow \vec{W} - r \cdot \nabla L(\vec{W}) = \vec{W} - r \cdot \sum_{i}^{m} \Delta\Psi$
4. Repeat steps 2 and 3 until $\vec{W}$ doesn't change anymore (10^{-5}).

After each iteration, $L(\vec{W})$ should be checked:

1. If $L(\vec{W})$ is converging, the learning rate is correct.
2. If $L(\vec{W})$ is diverging, the learning rate is too large.
3. If $L(\vec{W})$ is converging slowly, the learning rate too small.

Also, the algorithm needs feature scaling

$$x_i' = \frac{x_i - \min(\vec{X})}{\max(\vec{X}) - \min(\vec{X})}$$

1.10.2 Stochastic gradient descent

Instead of inspecting the whole dataset to detect the direction which minimize L, a single random sample is picked on each step.

1. Randomly shuffle the training set.
2. Starting with random weights.
3. For each sample $(\vec{X}_i, y_i)$: $\vec{W} \leftarrow \vec{W} - r \cdot \Delta\Psi$
4. Repeat step 3 until $\vec{W}$ doesn't change anymore (10^{-5}).

Convergence is not so obvious. After each bulk of iterations, e.g. 1000, check $L(\vec{W})$:

1. If $L(\vec{W})$ is converging, the learning rate is correct.
2. If $L(\vec{W})$ is diverging, the learning rate is too large.
3. If $L(\vec{W})$ is converging slowly, the learning rate too small.

1.10.3 Mini batches

While GD uses all samples in each iteration, SGD uses only one. A possible middle ground is to use a mini batch of samples in each iteration.

$$\vec{W} \leftarrow \vec{W} - r \cdot \frac{1}{b} \sum_{i}^{b} \Delta\Psi$$

Where b is the batch size, tipically 10.

1.10.4 Regularization

To prevent large weights, the norm of the weights is added to the loss function:

$$L(\vec{W}) = |\vec{W}| + \frac{1}{2}\sum_{i}^{m}\left(\sigma\left(\vec{W}\,\vec{X}_i\right) - y_i\right)^2$$

1.10.5 Early stopping (cross validation)

Other way to improve is to prevent overfitting:

1. Separate the data into training and validation sets.
2. Minimize $L(\vec{W})$ on the training set, stopping when $L(\vec{W})$ on the validation set stops improving.

1.11 Multi layered neural networks

This approach introduces one or more hidden layers in the network, each with one or more neurons. The model for a hidden layer h is the aggregation of the models of each neuron i in the layer.

$$y_{h,i} = \sigma\left(\vec{W}_i\,\vec{X}_h\right)$$

The aggregation of the outputs of the layer defines the input for the neurons in the next layer

$$X_{h^+} = (1,\, y_{h,1},\, \ldots,\, y_{h,i})$$

In practice, the layer's weights are aggregated in a matrix, performing the calculation in a single take. One implication is that the number of neurons in the hidden layers is directly proportional to the model's complexity.

1.11.1 Backpropagation

1. Starting with random weights.
2. For each sample, calculate the model, and if the result is incorrect:
   (a) Calculate *local gradients* for each neuron.
   For the neuron l in the last layer k:

   $$\delta_{k,l} = \sigma'\left(\vec{W}_l\,\vec{X}_k\right)\cdot(y - y_l)$$

   For the hidden neurons, let i^+ be the attached neuron in the next layer:

   $$\delta_{h,i} = \sigma'\left(\vec{W}_i\,\vec{X}_h\right)\cdot\left(\delta_{h^+,i^+}\;\cdot\; w_{h^+,i^+}\right)$$

   (b) Update the weights with the delta rule.
   Let $w^+_{h,i,j}$ be the updated weight, $w_{h,i,j}$ the current weight, and $w^-_{h,i,j}$ the previous weight:

   $$w^+_{h,i,j} = w_{h,i,j} + \gamma w^-_{h,i,j} + r\cdot\delta\cdot x_{h,i,j}$$

   Where γ is the momentum, a constant defined to prevent local optima.

1.12 Support Vector Machines

The VC dimension of a model is the higher number of samples for which it can solve **any** learning problem. Therefore, the VC dimension is an estimate of the capacity of a model.

The VC dimension for a model f and a training set of size n is also a bound on the test error

$$L_{\text{test}}(f) \leq L_{\text{train}}(f) + O\left(\sqrt{\frac{\text{VC}(f)}{n}}\right)$$

To reduce the test error:

1. Keep the training error low.
2. Minimize $\text{VC}(f)$.

By limiting the data to a sphere, we can place a bound on the VC dimension.
Let d be the dimensionality of the data, D the diameter of the sphere, and ρ the margin of the model

$$\text{VC}(f) \leq \min\left(d, \left\lceil \frac{D^2}{\rho^2} \right\rceil\right)$$

Therefore, by maximizing ρ, $\text{VC}(f)$ becomes **independent of the dimensionality of the data**.

1.12.1 Kernels

A kernel allows one to map the entries to a higher dimensional feature space, possibly allowing simpler ways to delimit such entries.
One example is the polynomial kernel:

$$(\vec{x} \cdot \vec{y})^n$$

1.13 Neural networks versus SVMs

1. Linear SVMs are similar to a Perceptron, but with an optimal cost function.
2. If a Kernel is used, then SVMs are comparable to 2-layer neural networks.
3. A 3-layer neural network might correspond to an ensemble of multiple Kernel SVMs.

1.14 Naive Bayes

Assuming conditional independence between the input dimensions, the probability of the target can be approximated using the Bayes theorem:

$$P\big(y \mid x_1, \ldots, x_d\big) \approx P(y) \cdot \prod_i^d P\big(x_i \mid y\big)$$

1.15 Ensemble learning

Ensemble learning consists in combining several simple models to form a more complex model.

Bagging: Each model training with a different dataset

Boosting: Same dataset, but instrumented for each model to mitigate the weakness of others

1.16 Boosting

Boosting is the technique of combining simple models iteratively to create a complex model.
Each model is intentionally **biased** to avoid the errors of the previous model.

One simple method of boosting is the **additive boosting**:
Considering binary classifiers

$$h : \vec{X} \mapsto y$$
$$y \in \{-1, 1\}$$

The model is defined as

$$h(\vec{X}) = \text{sign}\left(h_1(\vec{X}) + \ldots + h_n(\vec{X})\right)$$

1.16.1 Adaboost

The adaptive boosting algorithm is an additive algorithm, with associated importances:

$$h(X) = \text{sign}\left(\alpha_1 \cdot h_1(X) + \ldots + \alpha_n \cdot h_n(X)\right)$$

The adaboost algorithm is **always based on very simple models**, usually decision stumps.
As a consequence, it **does not overfit**.

1.17 Bagging

Boostrap aggregation is the technique of combining models trained in subsets of the training dataset.
The subsets are constructed by uniformly sampling the dataset, and may contain intersections.
Small subsets **prevent** the base models from **overfitting**, and therefore bagging circumvents **variance** in the data.

The models may be combined using many techniques:

- Majority voting.
- Averaging probabilites.
- Averaging estimates.
- Etc.

In practice, the base models are usually decision trees.

1.17.1 Random forests

Random forests exploits randomness in instances and features.
Each decision tree is trained with a random subset of **features** and instances.
As a consequence, random forests circumvent overfitting in decision trees.

2 Unsupervised learning

Unsupervised learning consists to, given only inputs as training, find a pattern:

- Clusters
- Manifolds
- Embeddings
- Etc.

2.1 Distance function

Some common distance functions are:

Nearest neighbor: $\min(|x-y|^2)$

Furthest neighbor: $\max(|x-y|^2)$

Centroid: $|\mu_i - \mu_j|^2$

2.2 Hierarchical agglomerative clustering

The hierarchical agglomerative clustering technique constructs a dendogram based on a distance function. Starting with individual clusters, it iteratively merges the closest ones until the dendogram is complete. Finally, a cut across the dendogram corresponds to a similarity threshold.

2.3 K-Means

The K-Means algorithm constructs clusters by placing centroids and agglomerating by the closest centroid. Considering k clusters, place k centroids in the space. Update the centroids iteratively using an expectation-maximization algorithm:

- Each cluster is defined by the points that are closest to the correspondent centroid.
- The centroids are updated with the mean of the points in it's cluster.

This technique can be interpreted as optimizing a loss function

$$L = \sum_i |x_i - \mu_j|^2$$

Different initial centroids may lead to different final losses, motivating different initialization methods:

Random: May choose nearby points

Distance based: Limits the search space, not enough randomness.

Random and distance based: Choose far points randomly.

The choice of k also impacts the disposition of results:

Small k: loose clustering.

Large k: any point is a group itself.

To prevent an overly large k, we must penalize complexity (regularization):

$$L = \log\left(\frac{1}{n \cdot d} \cdot \sum_i |x_i - \mu_j|^2\right) + k \cdot \frac{\log n}{n}$$

Where d is the dimensionality of the data, which also plays a fundamental role. With more dimensions, points tend to get sparse. Therefore, the more dimensions there are, the more samples one will need.

2.4 Mixture model

The mixture model constructs clusters by assigning probabily distributions and agglomerating by the most probable distribution. Therefore, it differs from K-Means by allowing overlaping clusters.
As in the K-Means algorithm, the distributions are randomly initialized, and updated using an expectation-maximization algorithm.

3 Reinforcement learning

Method where you train the program by rewarding the learning algorithm positively or negatively according to the produced results. This method is similar to how we teach animals.