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:

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

1.4 Error function

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

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

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 **test** sample is

$$E_{out}(f)$$

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

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

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

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 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

1.8 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

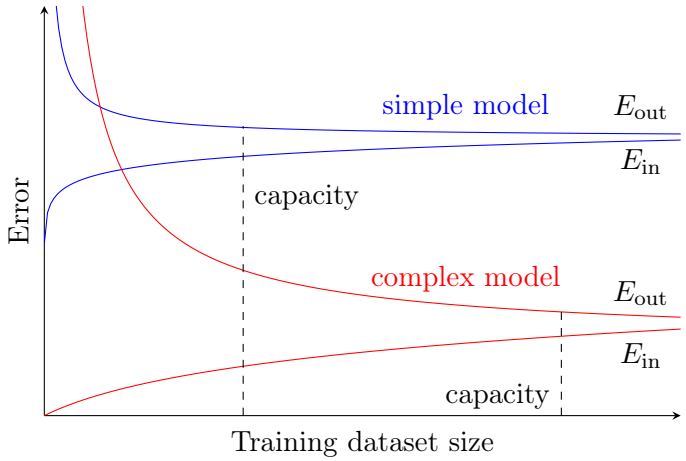
$$EH(A) = \sum_{i=1}^{k}\left[\frac{p_i + n_i}{p+n} \cdot H\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)\right]$$

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

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

1.9 Capacity

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



1.10 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.

1.11 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

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

1.11.1 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

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

1.11.2 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.

1.11.3 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.

1.11.4 Regularization

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

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

1.11.5 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.

2 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.

3 Unsupervised learning

Given only inputs as training, find a pattern: discover clusters, manifolds, embedding.