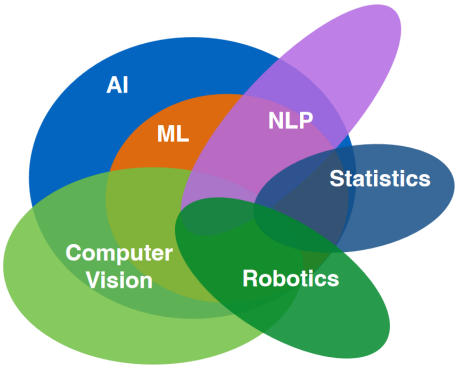
Introduction to Machine Learning Summary

Leonardo Baldo

Contents

1 Introduction

- **Artificial Intelligence:** the science to make things smart
- **Machine Learning:** building machines that can learn
- **Deep Learning:** class of ML algorithms



Artificial Intelligence

1.1 Artificial Intelligence

J.McCarthy 1956 definition: the science and engineering of making intelligent machines.

Modern definition: the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings.

1.2 Machine Learning

Arthur Lee Samuel 1959 definition: ML is the fields of study that gives computers the ability to learn without being explicitly programmed.

Tom Mitchell 1998 definition: A computer program is said to learn from experience E with respect to some class of tasks T ans performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

1.3 Learning Algorithm

The learning algorithm is an algorithm that is able to learn from data. There are 3 main ingredients:

- **Task**
  - Is described in terms of how the machine learning algorithm should process an example
  - How is an example represented? as a collection of features
- **Performance Measure**
  - How good is the machine learning system? we need to measure its performance
  - The performance measures depends on the task
- **Experience**
  - The experience is provided by the available data

1.4 Bias

Inductive Bias: all the assumptions about the nature of the target function and its selection.

Example of Inductive Bias:

- **Linear regression:** assume that the output or dependent variable is related to independent variable linearly
- **Nearest neighbors:** assume that most of the cases in a small neighborhood in feature space belong to the same class

Algorithmic Bias: It describes systematic and repeatable errors in a system that create unfair outcomes, such as privileging one arbitrary group of users over others.

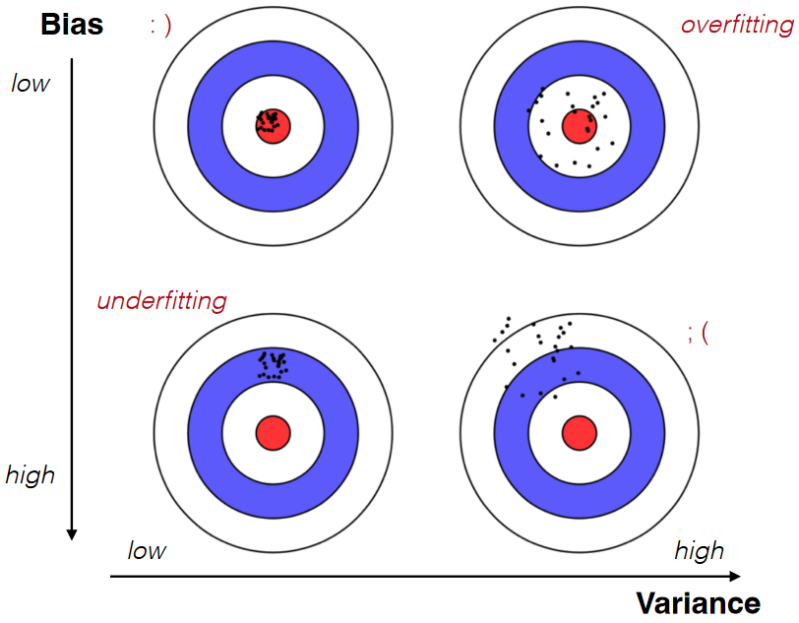
1.4.1 Bias vs Variance

The bias error is produced by weak assumptions in the learning algorithm.

The variance is an error produced by an oversensitivity to small fluctuations in the training set.

Underfitting: high bias can cause an algorithm to miss the relevant relations between features and target outputs.

Overfitting: high variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs.



Bias vs Variance

2 Linear Regression

Linear Regression is defined by the notation:

$$h_\theta x = \theta^T x \qquad \text{(Linear Regression)}$$

If two parameters are used, the function is:

$$h_\theta = \theta_0 + \theta_1^T x \qquad \text{(Linear Regression)}$$

The goal of this approach is to find parameters such that $\theta_{0,1}$, the result of the function $h_\theta(x)$, is close to the value of the target y. One way to quantify this distance is through the **Cost Function**:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \qquad \text{(Cost Function)}$$

Where:

- m: number of examples
- x: input variable
- y: output variable

The goal of the learning algorithm will then be to minimize the value of this function. To do this, the **Gradient Descent** method can be used.

3 Gradient Descent

Gradient Descent is a technique for obtaining the minimum points of a function.

The basic idea of this method is to always go in the opposite direction from the gradient, so as to arrive at the optimum value, which is at the lowest point, as quickly as possible.

$$\theta_{k+1} = \theta_k - \eta \nabla J(\theta_k) \qquad \text{(Gradient Descent)}$$

The parameter $\eta > 0$ is called the **Learning Rate** and represents the length of the "step" the algorithm takes in the direction of steepest descent. A learning rate that is too high risks that the algorithm will not converge while one that is too low increases the convergence time.

Batch Gradient Descent: To compute the gradient ∇J, the cost is computed over the entire training set, and after each update the gradient is recomputed for the new vector θ_{k+1}.

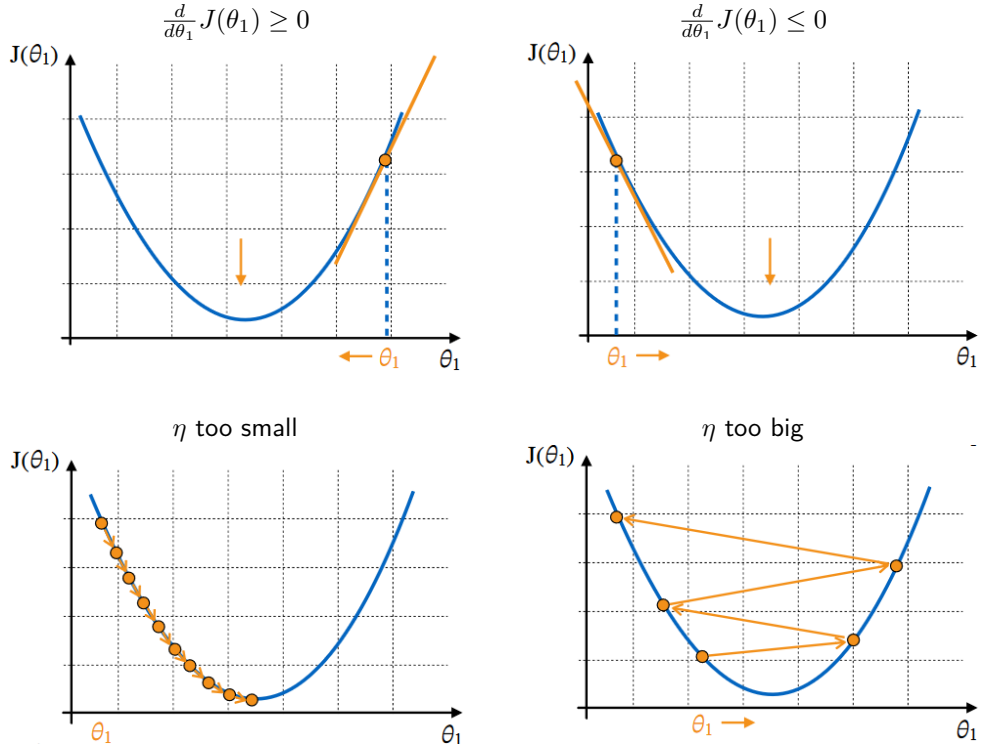
Stochastic Gradient Descent: method in which the parameters are updated for each cycle according to the gradient:

for $i = 1$ to m do{
    $\theta_0 = \theta_0 - \eta(\theta_0 + \theta_1 x_i - y_i)$
    $\theta_1 = \theta_1 - \eta(\theta_0 + \theta_1 x_i - y_i)x_i$
}

3.1 Simple Implementation

To proper implementation of gradient descent, it is necessary to update all parameters simultaneously:

repeat until convergence{
    $\text{tmp}_0 = \theta_0 - \eta \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
    $\text{tmp}_1 = \theta_1 - \eta \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
    $\theta_0 = \text{tmp}_0$
    $\theta_1 = \text{tmp}_1$
}



4 Classification

Linear Classification: used to determine to which discrete category a specific example belongs. (classification of emails into spam or ham)

Output category:

- **Binary classification:** 2 possible categories
- **Multi-class classification:** n possible categories

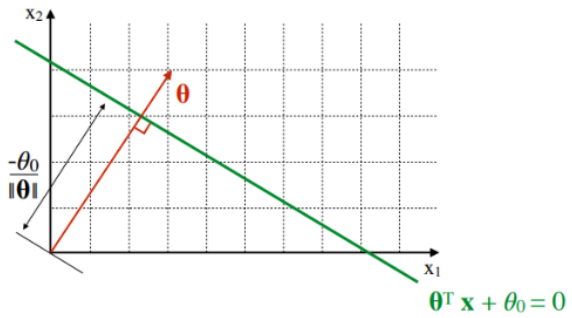
We can use binary classification by applying a threshold:

- $h_\theta(x) = \theta^T x$
- if $h_\theta(x) \geq 0$, then the output is $y = 1$
- if $h_\theta(x) < 0$, then the output is $y = -1$

$$y = sign(h_\theta(x)) \qquad \text{(Classification)}$$

Applying Linear Regression, for classification task, is not a good idea. Instead, it is preferable to use Logistic Regression, which, despite containing regression in its name, is a classification algorithm.

For larger dimensions, we assume a straight line, a shift of a certain term (bias term):



We then estimate a good decision boundary by deciding the direction θ and position θ_0 of the boundary, and we need a parameter selection criterion. For example, the Loss Functions.

4.1 Loss Functions

$$J_{01}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\{0 \text{ if } h_\theta(x_i) = y_i, \text{else } 1\} \quad \text{(Zero/One)}$$

$$J_{abs}(\theta) = \frac{1}{m}\sum_{i=1}^{m}|h_\theta(x_i) - y_i| \quad \text{(Absolute)}$$

$$J_{sqr}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 \quad \text{(Squared)}$$

The loss function for the Logistic Regression is:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\text{cost}(h_\theta(x_i, y_i)) \quad \text{(Loss function)}$$

where:

$$\text{cost}(h_\theta(x_i, y_i)) = \begin{cases} -\log(h_\theta(x_i)) & \text{if } y_i = 1 \\ -\log(1 - h_\theta(x_i)) & \text{if } y_i = 0 \end{cases}$$

Because this function is convex, we can use gradient descent. In fact, the method used to calculate the Linear Regression loss would not be correct in a convex function if used with the Logistic Regression function.

5 Logistic Regression

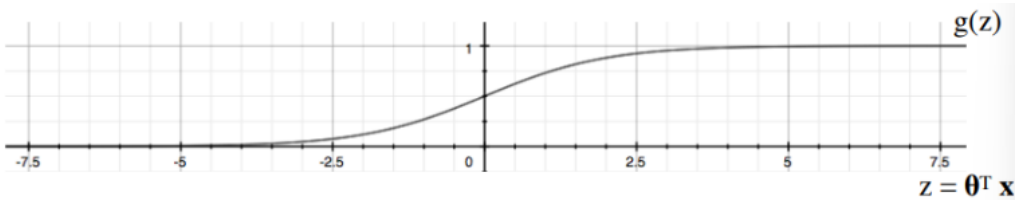
In logistic regression, the result of the prediction function $h(x)$ is between 0 and 1. This allows this value to also be considered a probability, making it ideal for classification tasks.

$$h_\theta(x) = g(\theta^T x) \qquad \text{(Prediction Function)}$$

Where:

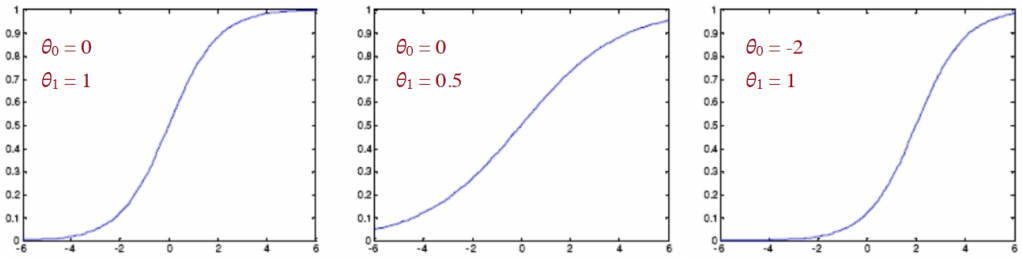
$$g(z) = \frac{1}{1 + e^{-z}}$$

The graphical representation is a sigmoid:



This function can "flatten" or "widen", depending on the values within θ, going to favor one class over another thereby succeeding in more accurately approximating a function that can divide the plane between classes.

$$h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$



5.1 Probabilistic Interpretation

Logistic regression benefits from the marginalization property, so:

$$h_\theta(x) = P(y = 1|x; \theta)$$

where $P(y = 1|x; \theta)$ denotes the probability that $y = 1$ given an input x, then:

$$P(y = 1|x; \theta) + P(y = 0|x; \theta) = 1$$

6 Regularization

6.1 Linear

Since the goal of the machine learning model is to **minimize loss**, a simple way to regularize the parameters is to multiply them by a number λ so as to force the algorithm to assign smaller and therefore less influential values, making the resulting parameters more accurately represent the totality of the examples.

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2 \qquad \text{(Loss Function)}$$

The parameter update step then becomes:

repeat until convergence{
$\theta_0 = \theta_0 - \eta\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_{0i}$
$\theta_j = \theta_j - \eta\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_{ji} + \frac{\lambda}{m}\theta_j$
}

6.2 Logistic

One can apply the same principle to logistic regression by **adding a regularization term** to the loss function, which becomes:

$$J(\theta) = \frac{1}{m}\left[\sum_{i=1}^{m} y_i \log(h_\theta(x_i)) + (1 - y_i)\log(1 - h_\theta(x_i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \qquad \text{(LF + Regularization)}$$

The parameter update step then becomes:

repeat until convergence{
$\theta_0 = \theta_0 - \eta\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_{0i}$
$\theta_j = \theta_j - \eta\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)x_{ji} + \frac{\lambda}{m}\theta_j$
}

7 Evaluation

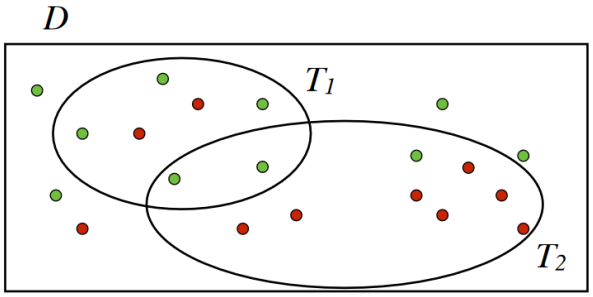
To evaluate a system, it is necessary to have access to metrics to quantify its performance.In the case of machine learning algorithms, the most important metric is Error.

7.1 True vs Empirical Error

The error that we can observe, the **Empirical Error**, is the one related to the training set available to us, and it can vary from set to set. The **Actual Error**, on the other hand, is generally unknown to us and is the classification error of all possible examples.

7.1.1 Example

- Correctly classified: $f(x) = h(x)$
- Misclassified: $f(x) \neq h(x)$



$$error_D(h) = 0.5$$

$$error_{T_1}(h) = 0.29$$

$$error_{T_2}(h) = 0.7$$

Depending on the training set the model may have too much bias or have too much variance. In the image the T_1 set of examples leads the model to have too much optimistic bias, the T_2 set on the contrary has too much variance and therefore any corrections are likely to be detrimental to the model because they will increase the bias or variance in the wrong directions. To improve our assessment of model performance it is important to make sure that the **Empirical Error** is as close as possible to the **True Error**.

7.2 Model Selection

7.2.1 Hold-out

Hold-out is the procedure that, once a training data set is defined, splits it into several subsets and one of those will be the set used to evaluate the model, also called the validation set.

- A classifier/regressor will then be trained on the training set data except those in the validation set and another, used to evaluate the performance of the model, called the test set.
- Usually the size of the training + validation sets should be larger than the test set, for example 70%, 15% and 15% respectively.
- The validation and test sets, which are not used for training, are used to be able to verify the model on data it has never seen.
- In this way, however, the validation and test data are useless for training.

7.2.2 K-fold Cross Validation

The most common method of making them useful is called **K-fold Cross Validation**, in which:

1. The dataset is divided into k equal partitions, the algorithm uses $k-1$ of them for training and the remaining partition for validation;
2. This procedure is performed k times, leaving a different partition out of the training set each time;
3. Finally, the various results obtained are averaged to calculate the overall performance of the model;
4. At this point a final evaluation of the resulting model can be made using the test set, which was not used within the cross validation, to avoid overestimating performance.

7.2.3 Leave-one-out

A special case of k-fold cross validation, called **Leave-one-out**, is when the number of partitions chosen matches the number of data within the training set, and thus at each iteration the training set will contain all but one of the available data. The purpose of cross validation is therefore to try to converge, as k increases, the average error they compute to the true error. But of course as k increases, the computational cost also increases since you have to redo k times the training so it is usually used with a relatively small k, usually 10.

7.3 Metrics

Evaluation metrics define how the goodness-of-fit of the model is evaluated. We consider several different metrics per classification task, starting with an example:

Evaluation metrics define how the goodness-of-fit of the model is evaluated. We consider several different metrics per classification task, starting with an example:

		Actual Class (groundtruth)	
		dog	no-dog
Predicted	dog	**TP:** True Positive	**FP:** False Positive *Type I error*
	no-dog	**FN:** False Negative *Type II error*	**TN:** True Negative
		P	**N**

Using this information we then see various metrics for a model:

- **Accuracy:** calculate how accurate it is

$$\frac{TP+TN}{P+N} = \frac{\text{all correct}}{\text{all instances}} = \frac{8}{11} \approx 0.73 \qquad \text{(Accuracy)}$$

- **Precision:** calculate how precise it is

$$\frac{TP}{TP+FP} = \frac{TP}{\text{all predicted}} = \frac{4}{6} \approx 0.67 \qquad \text{(Precision)}$$

- **Recall:** calculate how well it can predict

$$\text{sensitivity} = TPR = \frac{TP}{TP+FN} = \frac{TP}{\text{all actual}} = \frac{4}{5} = 0.80 \qquad \text{(Recall)}$$

- **False Positive Rate:** calculate how many of the incorrect predictions it makes

$$\frac{FP}{FP+TN} = \frac{FP}{\text{all incorrect}} = \frac{2}{6} \approx 0.33 \qquad \text{(FPR)}$$

- **F1-Score:** the harmonic mean of precision and recall

$$2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \qquad \text{(F1-Score)}$$

- **Average Precision (AP):** combines recall and precision for a ranked list

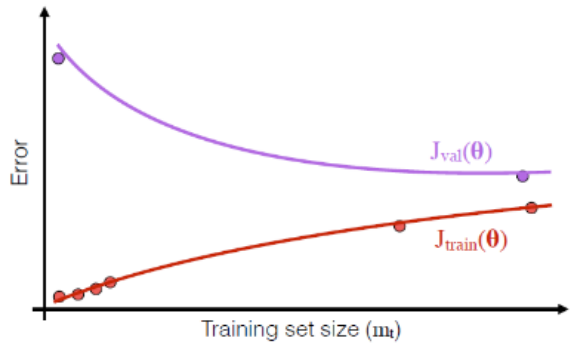
$$\frac{\sum_{k=1}^{n} P(k) \cdot R(k)}{\text{all actual}} \qquad \text{(AP)}$$

- **mean Average Precision (mAP):** the average of all AP values

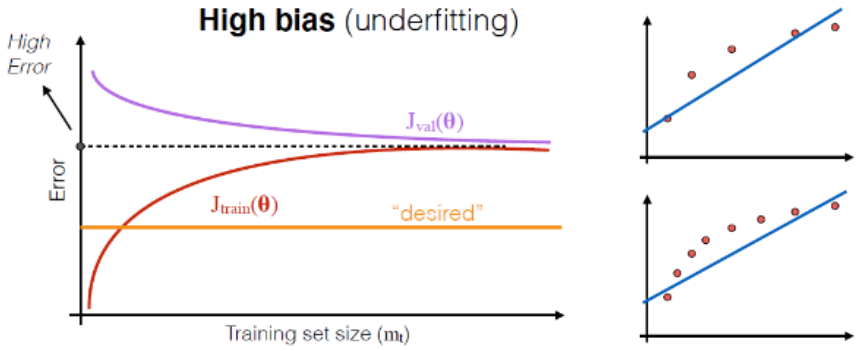
$$\frac{1}{Q} \cdot \sum_{q=1}^{Q} AP(q) \qquad \text{(mAP)}$$

8 Learning Curves

If the learning model does not produce the expected results, most often either high bias or high variance is the cause. A technique often used in practice is to compute the functions $J_{\text{train}}(\theta)$ and $J_{\text{val}}(\theta)$ by going to change the size of the training set:



To understand how to use this type of curve let us analyze examples:



Here we are in a case of high bias (underfitting) where, increasing the size of the training set, does not help and errors are higher than expected.

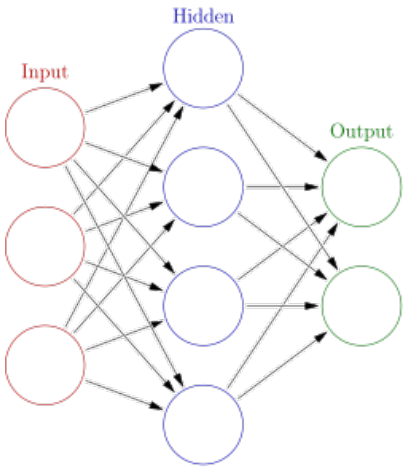
Here, on the other hand, we are in a case of high variance (overfitting) where the training error may be acceptable but the validation error is not, creating a large separation between the two curves. In cases like this increasing the size of the training set can help instead.

9 Neural Networks

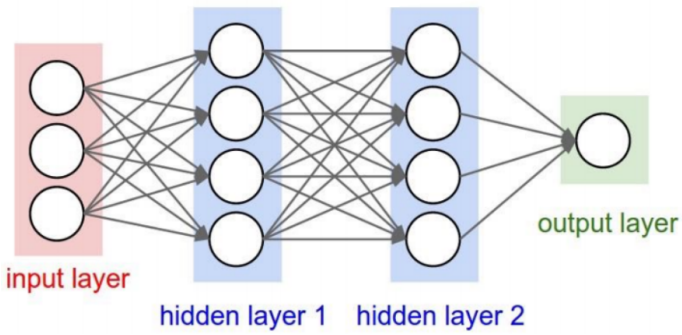
Neural Networks: model inspired by the structure and function of biological neural networks in animal brains.

9.1 Architecture

Architecture in the context of neural networks refers to the pattern developed to connect different neurons together. Typically, neurons are organized by levels (layers). In this course, only fully connected architectures are addressed, that is, where all neurons in one layer are connected to those in the next layer.



In this example a 2 layer neural network is schematized, one of which is hidden, for the nomenclature we do not consider the input layer since it is not affected by the parameters. When going to establish the architecture of a neural network one goes to establish the number of parameters to be learned.

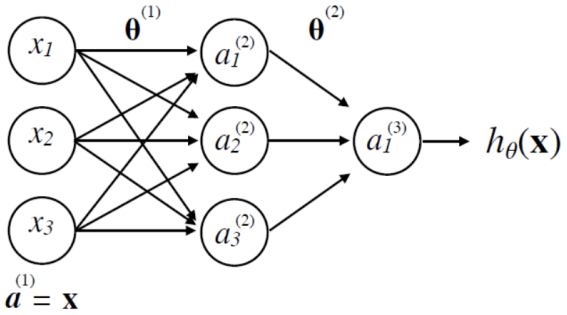


The total number of parameters is given by the sum of:

- 3 (# neurons input layer) $\times 4$ (# neurons hidden layer 1)
- 4 (# neurons hidden layer 1) $\times 4$ (# neurons hidden layer 2)
- 4 (# neurons hidden layer 2) $\times 1$ (# neurons output layer)
- 4 (bias hidden layer 1)
- 4 (bias hidden layer 2)
- 1 (bias output layer)

9.2 Feed-Forward Computation

Feed-Forward: neural networks in which the outputs of one layer become the inputs of the next layer. Each output will be the weighted sum of the inputs of the previous layer processed by the activation function, and this will be repeated until the final output is reached.



Node $a_1^{(2)}$ takes as input the three outputs of the first layer and through the nonlinear function f calculates the linear combination of the three inputs plus the bias:

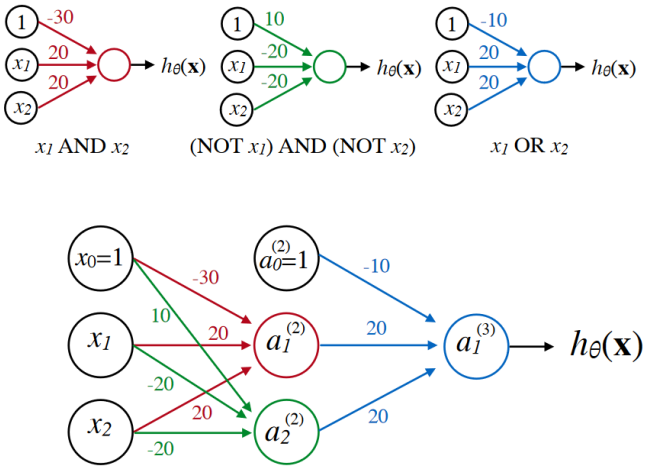
$$a_1^{(2)} = f(\theta_{10}^{(}1)x_0 + \theta_{11}^{(}1)x_1 + \theta_{12}^{(}1)x_2 + \theta_{13}^{(}1)x_3)$$

This is done for all nodes in the layer. $\theta^{(j)}$ denotes the matrix of weights between layer j and $j+1$. The summation of all linear combinations of a layer is denoted by the letter $z_i^{(j)}$ where j denotes the number of the layer, in this case $a_1^{(2)} = f(z_1^{(2)})$. If we focus only on the last hidden layer we can see that what we do is the same as logistic regression in which the function $h_\theta(x) = f(\theta^{(2)}a^{(2)})$ and the input is no longer the features but is the $a^{(2)}$ learned from the network. This makes us realize that all the previous hidden layers do is learn latent features from the data.

9.3 Boolean Functions

Neural networks can model various logic functions; a single perceptron can model a wide range of logic functions. But the XOR function is not learnable by a perceptron. The solution to this problem was the creation of deep neural networks multilayer.

To represent an XOR function one must be able to solve a nonlinear classification problem, and to do this one must combine several perceptrons together in the following way:



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	y	$h_\theta(\mathbf{x})$
0	0	≈ 0	≈ 1	1	≈ 1
0	1	≈ 0	≈ 0	0	≈ 0
1	0	≈ 0	≈ 0	0	≈ 0
1	1	≈ 1	≈ 0	1	≈ 1

$y = x_1$ XNOR x_2

9.4 Multiple Classes

We can use two strategies to tackle a K-way multiclass classification problems:

- **One-vs-one:**
  - we train $K(K-1)/2$ binary classifiers
  - each binary classifier is trained to discriminate between two classes
  - at prediction time we apply a voting scheme
- **One-vs-all:**
  - we train a classifier per class
  - each binary classifier is trained using its positive samples and samples belonging to all other classes as negative
  - this strategy requires each classifier to produce a real-valued confidence score for its decision
  - at prediction time we select the classifier with the highest confidence score

9.4.1 Loss Function

The Loss Function for the logistic regression is:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i \cdot \log(h_\theta(x_i)) + (1-y_i)) \cdot \log(1-h_\theta(x_i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \qquad \text{(Loss Function)}$$

To adapt it to multiclass classification, a summation is added to calculate the cost for the right class. Since the class vector uses one hot encoding the cost is calculated only for the predicted class.

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \cdot \log(h_\theta(x_i))_k + (1-y_k^{(i)})) \cdot \log(1-h_\theta(x_i))_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{u_l}\sum_{j=1}^{u_{l+1}}(\theta_{ji}^l)^2 \qquad \text{(Softmax)}$$

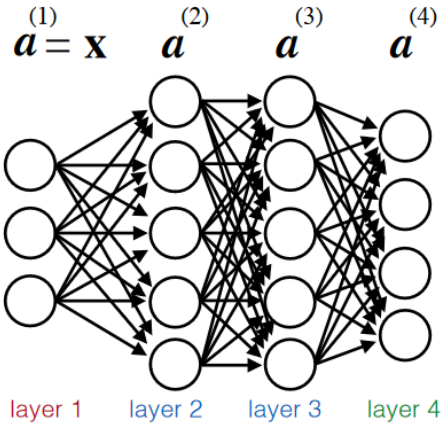
where:

- L is the number of layers in the network
- u_l is the number of units in the layer l

9.5 Parameter Learning

9.5.1 Feed Forward

The prediction value $h_\theta(x)$ is calculated by feed forward, by multiplying the output z of each layer with the weights θ of the next one and calculating the output z of the linear function f on the result.



9.5.2 Backpropagation

In the final layer, the cost will be the difference between the ground truth y_j and the output of the last layer a_j^l, and the error will then be propagated backward by multiplying the error by the vector of weights $\theta l - 1$ of the previous layer and multiplied element wise by $f'(z^{l-1})$, where z is the output of the linear function:

$$\delta^{l-1} = (\theta^{l-1})^T \delta^l \cdot f'(z^{l-1})$$

Backpropagation Algorithm:

set $\Delta_{ij}^l = 0 \quad (\forall i, j, l)$
for $k = 1$ to m{
    set $a^l = xa^k$
    for $l = 2$ to L{
        compute a^l
    }
    compute $\delta^L = a^L - y^k$
    compute $\delta^{L-1}, \delta^{L-2}, \ldots, \delta^2 = a^L - y^k$
    compute gradient $\delta_{ij}^l = \Delta_{ij}^l + a_j^l \delta_i^{l+1}$
}
if j!=0{
    compute $D_{ij}^l = \frac{1}{m}(\Delta_{ij}^l + \lambda \theta_{ij}^l)$
}
if j!=0{
    compute $D_{ij}^l = \frac{1}{m}\Delta_{ij}^l$
}

Consequences:

- Each unit j is responsible for a fraction of the error δ_j
- The error is calculated according to the strength of the connections between the layers
- The error is propagated backward depending on the weights of the units

10 Support Vector Machines

10.1 Characteristics

- Supervised learning algorithm usually used for classification
- Binary classifier, so it is not probabilistic
- Linear classifier, but can perform nonlinear classification using particular kernels;
- Use regularized logistic regression

10.2 Relation between Logistic Regression and SVM

Logistic classification has the following formula:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{\theta^T x}}$$

Where:

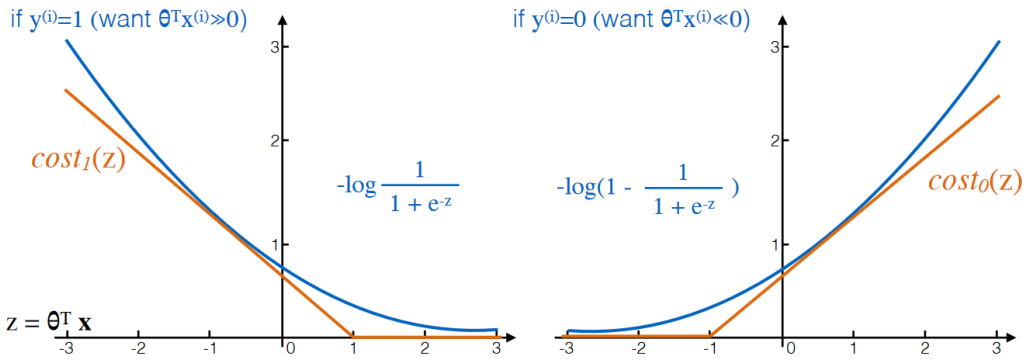
$$g(z) = \frac{1}{1 + e^{-z}}$$

The cost function for logistic regression given an example i and a vector of weights θ is as follows:

$$\text{cost} = -y^i \cdot \log(\frac{1}{1 + e^{\theta^T x^i}}) - (1 - y^i) \cdot \log(1 - \frac{1}{1 + e^{\theta^T x^i}})$$

What we'd like logistic regression to do:

- if $y = 1$, we want $h_\theta(x) \approx 1, \theta^T x \gg 0$
- if $y = 0$, we want $h_\theta(x) \approx 0, \theta^T x \ll 0$



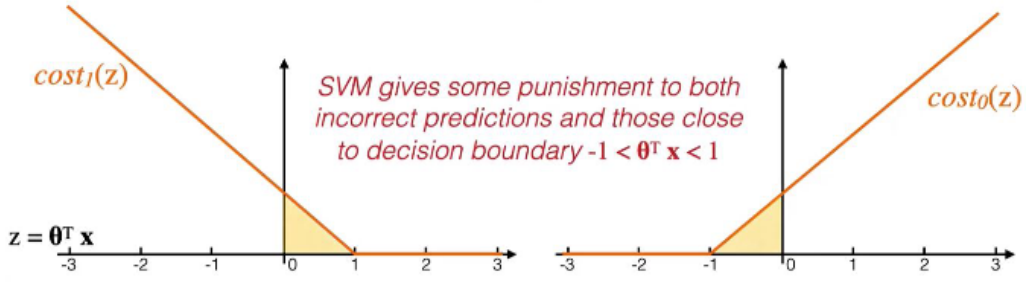
The objective function of an SVM can be written as follows:

$$\min \frac{1}{\lambda} \sum_{i=1}^{m} [y^i \cdot \text{cost}_1(\theta^T x^i) + (1 - y^i) \cdot \text{cost}_0(\theta^T x^i)] + \frac{1}{2} \sum_{j=i}^{n} \theta_j^2$$

The functions cost_1 and cost_0 allow us to create a decision boundary.

10.3 Decision Boundary

The decision boundary allows too ambiguous results to be penalized, for example with a decision boundary between 1 and -1 values of $\theta^T x$ will be penalized.



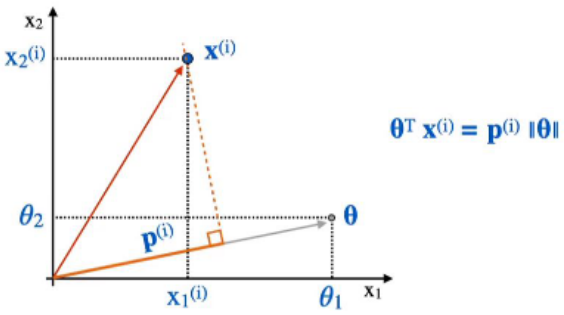
The Loss Function of SVM is called **Hinge Loss** and is defined as follows:

$$\text{cost}(h_\theta(x_i), y_i) = \begin{cases} max(0, 1 - \theta^T x_i) & \text{if } y_i = 1 \\ max(0, 1 + \theta^T x_i) & \text{if } y_i = 0 \end{cases} \qquad \text{(Hinge Loss)}$$

With this loss function an equidistant margin will be selected from the various examples.
We can rewrite the optimization objective as follows:

$$\min \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 = \frac{1}{2} \|\theta\|^2 \quad \Rightarrow \quad \begin{cases} \theta^T x_i \geq 1 & \text{if } y_i = 1 \\ \theta^T x_i \leq -1 & \text{if } y_i = 0 \end{cases}$$

In case we were in two dimensions:



p_i is the projection of x_i onto θ, so $\theta^T x_i = p_i \|\theta\|$.
We can rewrite the two constraints in this way:

- $p_i \|\theta\| \geq 1$, if $y_i = 1$
- $p_i \|\theta\| \leq -1$, if $y_i = 0$

The SVM will then go to choose a decision boundary that maximizes p_i since the only way to minimize the objective function is by decreasing θ.

10.4 Margin Extension

In case there are outliers among the examples it is necessary to introduce a way to ignore this error noise. We can introduce into the model a variable ξ_i called the **slack variable**:

$$\min \frac{1}{2}\|\theta\|^2 + \alpha \sum_{i=1}^{m} \xi_i$$
$$\xi_i \geq 0, \forall i : y_i(p_i\|\theta\|) \geq 1 - \xi_i$$

Extending the soft margin leads to the definition of hinge loss:

$$l(h_\theta(x_i), y_i) = \max(0, i - y_i(\theta^T x_i))$$

The Hinge Loss is a convex function and can be optimized by gradient descent.

10.5 Kernel Trick

For classification of nonlinear data with SVM, a practice called **Kernel Trick** is used. The Kernel Trick projects the data into a new space via a function called kernel. Starting from a particular space $\mathbb{R}_n$, impose the same constraints in a higher dimensional space $\mathbb{R}_m$. The kernel function is a function $\phi : \mathbb{R}_n \to \mathbb{R}_m$ that maps vectors in $\mathbb{R}_n$ to a space $\mathbb{R}_m$.

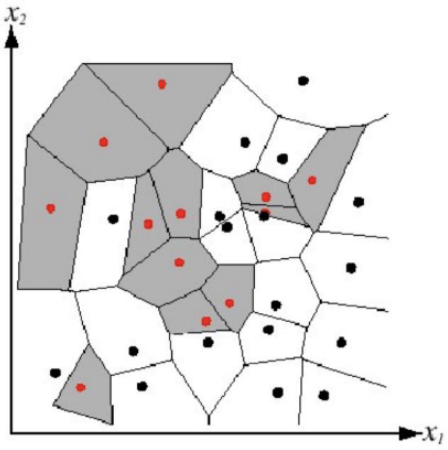
$$k(x, y) = \phi(x)^T \phi(y) \tag{1}$$

11 kNN Models

By nonparametric models we generally refer to **Nearest Neighbor Models**, this type of approach is called "lazy learning" in that it does not build any model, all the work is done during the prediction phase.

11.1 Nierest Neighbors

A simple implementation of a Nearest Neighbor algorithm simply corresponds to calculating the minimum distance between the input and the examples in the dataset and assigning the class of the nearest example to the input. This method does not go to explicitly create decision boundaries but they can be calculated and shown using a Voronoi diagram.

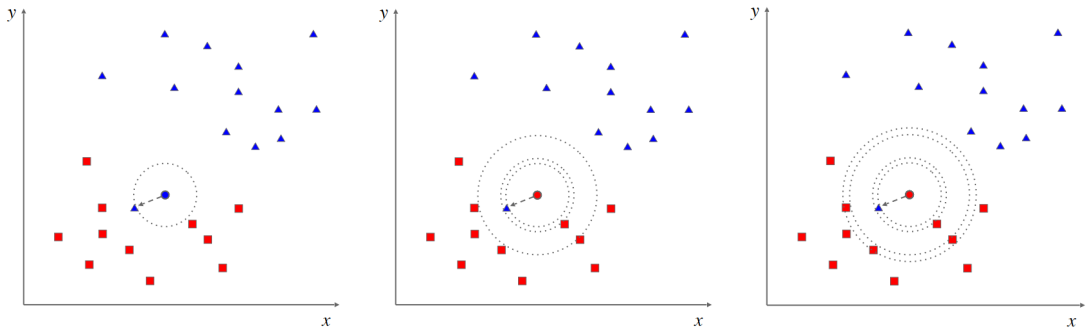


11.2 k-Nearest Neighbors

The k closest examples are considered to classify the input. The input is ranked according to the class of the majority of the k closest examples. However, too large a k can be a problem since there is a risk that considering too many examples always wins the most frequent class in the dataset. A good rule for choosing the k is as follows:

$$k = \sqrt{m}$$

where m is the number of examples in the dataset.



11.3 Python Implementation

```
import numpy as np

class NearestNeighbor:

    def __init__(self):
        pass

    def train(self, X, y):
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        num_test = X.shape[0]
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)
        for i in range(num_test):
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances)
            Ypred[i] = self.ytr[min_index]
        return Ypred
```

11.4 Pros and Cons

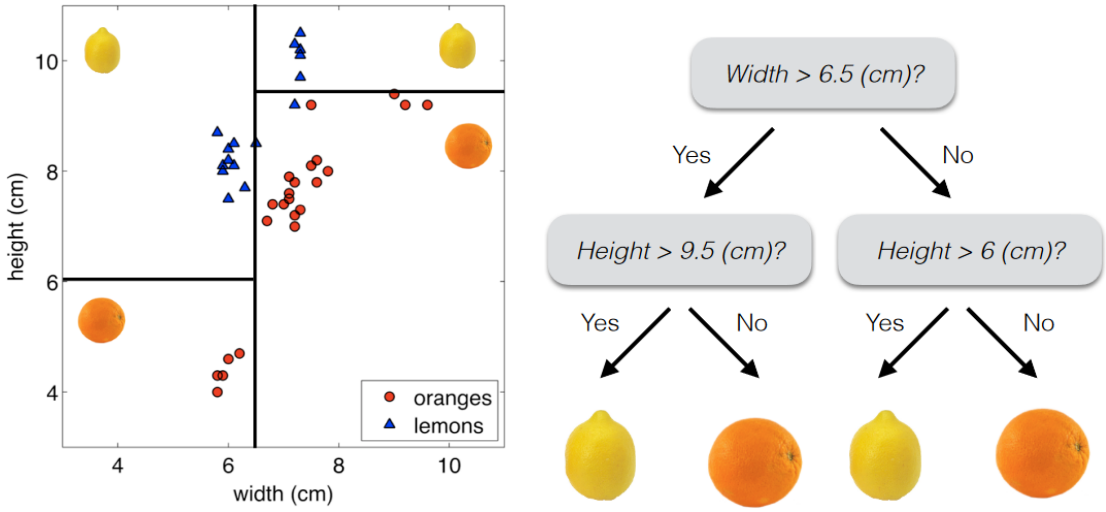
- Creates complex decision boundaries and adapts to data density
- With many samples it works well
- Sensitive to noise between classes and different feature scales
- In high dimensionality datasets is less effective
- Complexity scales linearly with the number of training data, very expensive at test time
- Inductive bias (assumes neighboring samples in feature space have the same class)

12 Decision Trees

The space is recursively partitioned, each node in the tree is responsible for establishing a subdivision of the space, the final leaf corresponds to the output.

- Each node corresponds to only one attribute
- Branching is determined by the value of the attribute
- Each leaf is associated with a class

For each training set it is possible to general a decision tree with a path to the correct leaf for each sample, the generated tree however does not generalize over other test examples (overfitting). It is therefore necessary to regularize to create more compact trees that can generalize.



12.1 Learning Decision Trees

Creating the simplest decision tree is a complete NP problem, so heuristic solutions are used to simplify the problem:

1. start with an empty decision tree
2. divide the tree on the attribute that subdivides the examples among them in the best way
3. recursively repeat the process

12.1.1 ID3 Algorithm

ID3 is a popular for learning decision trees and works like this:

1. creates a root node r
2. if all examples S belong to the same class y returns the root node with label y
3. if A (attributes) is empty it will return r with the same value as the most frequent class in S
4. else select the optimal node:
    a. partitions the set S into subsets using the attribute for which entropy is minimized
    b. creates a decision node containing the selected attribute
    c. applies recursion in the underlying branch

12.1.2 Entropy

To choose the optimal attribute, one can use the **Entropy** H, which is the measure of the amount of randomness of a certain dataset S, which is defined by the formula:

$$H(S) = -\sum_{c \in C} p_c \log_2 p_c \qquad \text{(Entropy)}$$

where:

- C is the set of all possible classes
- p_c is the proportion of elements in c to all elements contained in the dataset S

In case of **high entropy** we will have:

- variable with a uniform distribution
- flat histogram
- less predictable sampled values

In case of **low entropy** we will have:

- distribution of the variable with many peaks and valleys
- histogram with many highs and lows
- more predictable sampled values

Information gain: entropy-based measure that measures the change in entropy depending on the attribute on which the partitioning is based.

The formula for information gain is as follows:

$$\text{IG}(S, a) = H(S) - \sum_{t \in T} p_t H_t = H(S) - H(S|a) \qquad \text{(Information gain)}$$

where:

- $H(S)$ is the entropy of the dataset S
- T are the subsets created by splitting S according to the attribute a
- p_t is the proportion of elements in t to all elements in S
- H_t is the entropy of the subset $t \in T$

12.2 Characteristics

What makes a good tree?

- Not too small: need to handle important but subtle distinctions in data
- Not too large: computational efficiency, avoid overfitting training examples
- Occam's Razor: find the simplest hypothesis
- Inductive Bias: small trees that place high information gain attributes close to the root are preferred

12.3 Issues

- You have exponentially less data at lower levels
- Too big of a tree can overfit the data
- Greedy algorithms usually don't yield the global optimum
- Information gain privileges the attributes that assume a broad range of values