Machine Learning

Davide Volpi

January 16, 2025

Contents

1 Introduction

1.1 Design of a Supervised Learning Algorithm

Learning reduces to **selecting** a function $h : X \to Y$ exploiting a given dataset, called the training set, such that its error is minimized. We have to select a function h among the functions in a pre-defined set H. Once picked H, we are going to select $h \in H$ that minimises the error on the training set.
We cannot say anything about data outside the training set.
No Free Lunch Theorem: If H is big enough there are infinite functions but their error on unseen examples will vary greatly. "Every successful ML algorithm must make assumptions".

1.2 Inductive Bias

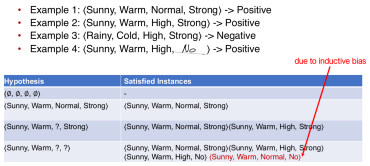
Inductive Bias are all the assumptions about the nature of the target function and it's selection. There are two type:

- **Restriction:** limit the hypothesis space.
- **Preference:** impose ordering on hypothesis space.

1.2.1 Example of Inductive Bias

Purpose: The Find-S algorithm is used in concept learning, is a type of machine learning where the goal is to identify a target concept from labelled examples.
Concept: Is a rule that classifies instances into positive/negative categories.
Goal: Identifies the most specific hypothesis that fits all positive examples while ignoring the negative examples.
The algorithm works as follow:

1. Start with the most specific hypothesis which assumes no attributes are relevant.
2. Process only positive examples. For each positive example, compare it with the current hypothesis. For any attribute that differs between the example and the current hypothesis, generalize the hypothesis by replacing that attribute value with a wildcard (?).



The algorithm skips negative examples entirely, focusing on making the hypothesis general enough to include all positive examples.

1.3 Error Stimation

1.3.1 True Error

True error of hypothesis h with respect to target concept c and distribution D is the probability that h will miss-classify an instance drawn at random according to D: $error(h) = Pr[c(x) \neq h(x)] \forall X \in D$.

1.3.2 Empirical Error

Empirical error of hypothesis h with respect to the true error is the number of examples that h miss-classifies: $error_{Tr}(h) = Pr(x, f(x))_{\in Tr}[f(x) \neq h(x)] = |\frac{\{(x,f(x)) \in Tr | f(x) \neq h(x)|\}}{Tr}|$.

1.3.3 Risk Estimation

- $RISK(h, P) = E(x, y) \sim P[L(h(x), y]$
- $RISK_{emp}(h, d) = \frac{1}{|D|} \sum_{(x,y) \in D} L(h(x), y)$

- **0/1 Loss:** $L(h(x), y) = 0 \quad h(x) = y, 1 \quad otherwise$

We want to **minimize the true error:**

1. Relate the empirical error and the true error with **generalization bounds**.
   $error_D(h) + ComplexMeasure(H)$.
2. Compute the error on unseen data (test set).

1.3.4 Bias and Variance

Bias: Refers to how well our model can represent any training set.
Variance: Sensitivity to changes in the training set.
Overfitting: The model fits too well with the training set, but it is not able to generalise well to the test set (**high variance**).
Underfitting: Inability to fit the training set, and therefore also, the test set (**high bias**).

2 Linear Regression

We have to chose Θ_0, Θ_1 so that $h_\Theta(x)$ is close to y for our training examples $\{(x^{(i)}, y^{(i)}\}$.
We have $h_\Theta(x) = \Theta_0 + \Theta_1 X$.
Least Squared (residuals): sum of squared distances $\{h_\Theta(x^{(i)}) - y^{(i)}\}$.
Cost Function: $J(\Theta_0, \Theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\Theta(x^{(i)}) - y^{(i)})^2$ (**MSE**) and we want to **minimize** $J(\Theta_0, \Theta_1)$.

2.1 Gradient Descent

$\text{tmp0} \coloneqq \theta_0 - \eta \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1);$

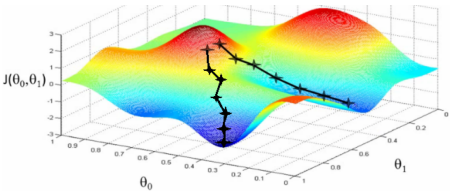
$\text{tmp1} \coloneqq \theta_1 - \eta \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1);$

$\theta_0 \coloneqq \text{tmp0}; \ \theta_1 \coloneqq \text{tmp1};$

- n is the **learning rate**, which can be small/big (0,001-0,003 ; 0,01-0,03 ; 0,1-0,3 ecc).
- When we reach any minimum we stop derivative Θ.

2.2 Gradient Descent in Linear Regression

- $j = 0 \ \frac{d}{d\Theta_0} J(\Theta_0, \Theta_1) = \frac{1}{m} \sum_{i=1}^{m} (\Theta_0 + \Theta_1 x^{(i)} - y^{(i)})$
- $j = 1 \ \frac{d}{d\Theta_1} J(\Theta_0, \Theta_1) = \frac{1}{m} \sum_{i=1}^{m} (\Theta_0 + \Theta_1 x^{(i)} - y^{(i)}) x^{(i)}$



A quicker way to compute the Θ is $\Theta 0 (X^T X)^{-1} X^T y$. This formula gives a vector with the optimal values of Θ but not always is possible to compute it in this way.

3 Linear Classification Models

Decision Boundary: boundary behind a decision rule to assign a variable $y = sign(h_\Theta(x))$.
The linear boundary separates the space into two half-spaces with a **threshold**.

3.1 The Perceptron

Is a learning algorithm for **binary classification**. Given $x \in R^d$, its Hypothesis Space is composed of all linear functions in the input space $\Theta^T x$, where $\Theta \in R^d$ are parameters we want to learn from the data. This version assumes that the data is linearly separable, and in general the panel is called **hyperplane**.

- $h : X \to \{-1, 1\}$
- $\Theta^T x = \sum_i x_i \Theta_i$
- **Decision Rule:** $sign(h_\Theta(x))$

- θ = 0; m=1;
- while m>0
  - m=0 # m counts the number of errors
  - for (x,y) in D
    - if θ^Txy <= 0 #prediction error
      - $\theta = \theta +$ xy # update the classifier
      - m = m + 1

Let's try with an example dataset:
1. $h(x) < 0;\ y = -1 \to ok;\ h(x)y > 0$
2. $h(x) > 0;\ y = 1 \to ok;\ h(x)y > 0$
3. $h(x) <= 0;\ y = 1 \to err;\ h(x)y \leq 0$
4. $h(x) >= 0;\ y = -1 \to err;\ h(x)y \leq 0$

The classifier is more confident about predicting points far from the boundary that near ones.
$\Theta'^T xy$ after every step of the algorithm will be $\Theta^T xy = (\Theta + xy)^T xy = \Theta^T xy + x^T x$.
If the examples are linearly separable by a **margin** γ such that $||x^{(i)}|| \leqslant R \forall i$ and $\Theta^T x^{(i)} y_{(i)} > \gamma$, then the perceptron will find the separable hyperplane in R^2/γ^2 steps. By proving this convergence we obtain that $k^2\gamma^2 < ||w^{k+1}||^2 \leqslant kR^2$, which implies $k < R^2/\gamma^2$.

3.2 Sources of Error

Let ϵ_i be a random variable with mean 0 and variance θ^2, then $y_i = f(x_i) + \epsilon_i$ and we obtain:

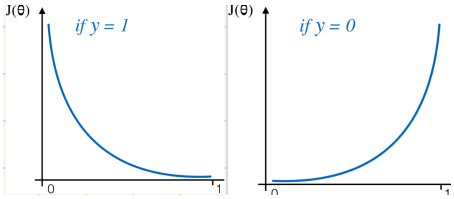
- $BIAS(h(x), P) = E_{(x,y)\sim P}[h(x) - y]$
- $VARIANCE(h(x), P) = (E_{(x,y)\sim P}[h(x) - y])^2$

We can show that $RISK(h, P) = \theta^2 + BIAS(h, P)^2 + VARIANCE(h, P)$ (θ^2 is the **irreducible error**).

4 Logistic Regression

$h_\Theta(x) = g(\Theta^T x)$ where $g(z) = \frac{1}{1+e^{-z}}$. We have that $y = \begin{cases} 1 & if \quad h_\Theta(x) \geqslant 0.5 \\ 0 & if \quad h_\Theta(x) < 0.5 \end{cases}$.
$J(\Theta) = \frac{1}{m}\sum_{j=1}^{m} Cost(h_\Theta(x), y)$ where $Cost(h_\Theta(x), y) = -ylog(h_\Theta(x)) - (1-y)log(1-h_\Theta(x))$

So the loss becomes: $\begin{cases} -log(h_\Theta(x^i) & if \quad y^{(i)} = 1 \\ -log(1-h_\Theta(x^i) & if \quad y^{(i)} = 0 \end{cases}$



We can learn our parameters with gradient descent:

repeat until convergence{

$\theta_j := \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta) = \theta_j - \frac{\eta}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$

(simultaneously update all θ_j)

}

5 Regularization

To address overfitting we have two main options:

- **Model Selection:** Reduce the number of features.
- **Regularization:** Consists in keeping all the features, but reduce the influence of parameters Θ_j.

5.1 Regularization for Linear Regression

We can learn our parameters with gradient descent from this regularized formula:
$min_\Theta J(\Theta) = min_\Theta \frac{1}{2m}[\sum_{i=1}^{m}(h - Theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \Theta_j^2]$ where λ is the **regularization term**.

repeat until convergence{

$\theta_0 := \theta_0 - \eta \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_0^{(i)}$ (simultaneously update all θ_j)

$\theta_j := \theta_j \left(1 - \eta \frac{\lambda}{m}\right) - \eta \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)}$ $(j = 1, \ldots, n)$

}

This is going to be <1

This is exactly the same term we had before introducing regularization

- λ and n are both given constants.
- This is called **Weight Decay**.

5.2 Regularization for Logistic Regression

We can learn our parameters with gradient descent from this regularized formula:
$min_\Theta - \frac{1}{m}[\sum_{i=1}^{m} y_{(i)} log(h_\Theta(x^{(i)})) + (1 - y) log(1 - h_\Theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^{n} \Theta_j^2$

repeat until convergence{

$\theta_0 := \theta_0 - \eta \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_0^{(i)}$ (simultaneously update all θ_j)

$\theta_j := \theta_j - \eta \left[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})\, x_j^{(i)} + \frac{\lambda}{m} \theta_j\right]$ $(j = 1, \ldots, n)$

If the learning algorithm has hyperparameters (λ), we need to split the training set into **training-evaluation set** to optimize the hyperparameters.

6 True Risk Estimation

The performance on the test set is a good estimator of the true risk of h if the test set is representative of P(x,y) and no information is leaked from the training set to the test set, which is not used in the selection of h.

6.1 Hold-Out

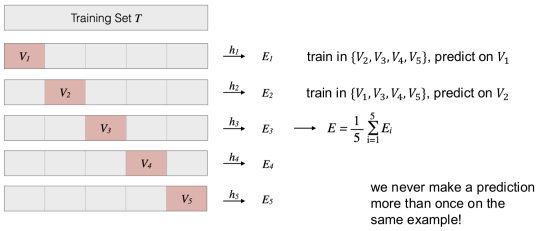
We keep a subset of v samples from the training set to evaluate our hyperparameters.
In this way parameters Θ and hyperparameters λ are optimised on the training/validation sets.
The best λ is selected on the validation set and after you can retrain the model and evaluate its performance on the test set.
The **size of the validation set** is critical because it should be large and small at the same time.

6.2 Leave One Out

Training set $= \{x_1, \ldots, x_m\}$, we train our model on $m-1$ examples (all but x_i) and then check the error on x_i.
We repeat the process for i=1 to m and then we average the predictions $E(h, \{x_1, \ldots, x_m\}) = \sum_{i=1}^{m} E_i$.
This method is **unbiased but expensive**.

6.3 K-Fold Cross Validation

Is like leave one out, but we leave out a number of examples instead of only one.
The **Rule of Thumb** states that $k = 10$ when we can afford it.



6.3.1 Hyperparameter Search

If we try too many values, E_{val} is no longer a good estimator of the true risk: $E \leqslant E_{val} + O\sqrt{ln(M/v)}$, where M is the number of models checked and v the validation size. **Telescopic Search:** coarse grained search first, then a fine-grained one on the most promising region:

- Try a number of values of different order of magnitude and select the best $\lambda : \lambda^*$.
- If λ^* is at the border of the range try bigger values until you get a λ^* which is between two values with lower performance.

Hoeffding's Inequality: probabilistic bound on the deviation of the empirical error (observed in the validation set) from the true error: $P(|\hat{E_{val}} - E| \geqslant \epsilon) \leqslant 2e^{-2v\epsilon^2}$ where E_{val} is the observed validation error, E is the true error, ϵ is the tolerance for deviation and v is the size of the validation set.
For M Models: probability of at least one model having an error deviation greater than ϵ is no more than the sum of the probabilities for each model $P(\exists i : |\hat{E_{val}} - E| \geqslant \epsilon) \leqslant 2Me^{-2v\epsilon^2}$.
Suppose we desire a fixed large probability $1-\delta$ for which we want our bound to hold:

- Fixed the validation set size we have that: $\epsilon \leqslant \sqrt{\frac{log(2M/\delta)}{2v}}$.
- Fixed the tolerance and probability we have that: $v \geqslant \frac{1}{2\epsilon^2} log(\frac{2}{\delta}$.

7 VC Dimension

The VC dimension measures the complexity of an hypothesis space.
Dichotomy: every $h \in H$ partitions a set S into two classes (assuming a binary classification problem). There are $2^{\S|}$ dichotomies.
h is Consistent with S: iff for each $(x, y) \in S.h(x) = y$ (h makes no errors on S).
Shattering: a set S is shattered by H iff for every dichotomy of S, there exists $h \in H$ consistent with the dichotomy.
VC(H) is the size of the largest $S \subset X$ shattered by H.

1. $|S| = 1$
2. If there exists **al least one** tuple of $|S|$ points, for which H shatters it:
   - Then $|S| = |S| + 1$ (add another point and go back to one).
   - Else: $VC(H) = |S| - 1$.

In general, $VC(perceptron) = d + 1$, where d is the dimension of the input space.

7.1 Generalisation Bounds

The VC dimension is used to derive bounds on the generalisation error.
Given a $Tr = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ and a generic H, assuming a learning algorithm selects a g that minimise the empirical risk, then we can derive an upper bound on the true risk which is valid with probability $1 - \delta$: $Risk(g) = Risk_{Tr}(g) + F(\frac{VC(H)}{\delta})$, where F is a function proportional to $CV(H)/m$.
Larger the training set, highest the bound; Low VC dimension, better bounds → we need to find a tradeoff.

8 Non-Parametric Models

Non Parametric Models have a variable number of parameters, which normally depends on the size of the training set. These are simple methods for approximating any function and are instance based, so they construct a different approximation of the target function for each query instance.
These is not a standard learning phase in which we optimise an error function on the training set.
Lazy Learner: all the work is done at prediction time.

8.1 Nearest Neighbour (1-NN)

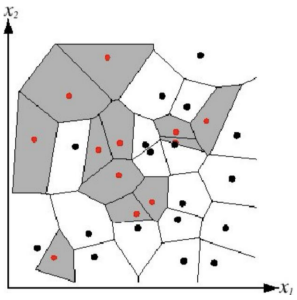
The value of the target function for a new example is estimated from the known training example. This is done by computing distances between the new example and all the training example.
Decision Rule: assign the table of the nearest example.
Euclidean Distance: $\sqrt{\sum_{j=1}^{d}(x_j^{(a)} - x_j^{(b)})^2}$.

Algorithm:

Find (x^*, y^*) (from the stored training set) closest to the test sample x

i.e. $x^* = \underset{x^{(i)} \in TrainSet}{\arg\min}\ \text{Distance}(x^{(i)}, x)$. Output: $y = y^*$

Vornoin Diagram Visualization is used for the decision boundary.
The input space is divided into classes and each line segment is equidistant between two samples of opposite classes.
Inductive Bias: It assumes that the most of the cases in a small neighbourhood of a point in feature space belong to the same class.
Sensitive to **outliers** (noise), one point might influence a large area.



8.2 K-NN

Compute the distance between the new sample and all the training samples.
Select the k closest examples (k usually odd).
Decision Rule: assign the label of the majority class among the k nearest neighbours.

Algorithm:

Find k examples $(x^{(i)}, y^{(i)})$ (from training set) closest to the test sample x

Output: $y = \underset{y^{(z)}}{\arg\max} \sum_{j=1}^{k} \delta(y^{(z)}, y^{(j)})$

Some features have larger ranges, so are treated as more important. We can apply **feature scaling:**

- Linearly scale the range of each feature to be in $[0, 1]$ range.
- Linearly scale each dimension to have 0 mean and variance 1: $(x_j - \mu)/\sigma$.
- Divide each vector by its norm: $x/||x||$.

In alternative you can use a **custom distance function**.
Time Complexity: time to train the model and to make a prediction on a new example.
Space Complexity: how much space we need to store h.
Having $\{(x_1, y_1) \dots (x_m, y_m)\} : x \in R^n \rightarrow O(nm) = O(m)\ if\ m >> n$
We obtain $||x - x_1||_2 0 \sum_{i=1}^{n}(x[i] - x_1[i])^2$.
KNN is expensive at test time:

- **Time Complexity:** compute **approximate distance** or use **pre-sorting/fast data structures**.
- **Storage Requirements: remove redundant data points**.

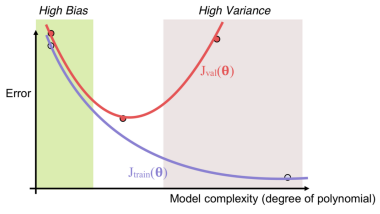
This solutions can help but are not enough. With large vectors we have to deal with the **course of dimensionality:** in high dimensional spaces all points are the same distance. We hope that data actually lies in a low-dimensional space.

9 Debugging ML Models

If your learning model doesn't work as expected, almost all the time it will be because you have either a high bias problem or a high variance problem.
Let ϵ_i be a random variable with $\mu = 0$ and σ^2, then $y_i = f(x_i) + \epsilon_i$:

- $RISK(h, P) = E_{(x,y)\sim P}[(h(x) - y)^2]$
- $RISK(h, P) = \sigma^2 + BIAS(h, P)^2 + VARIANCE(h, P)$
- $BIAS(h, P) = E_{(x,y)\sim P}[h(x) - y]$
- $VARIANCE(h, P) = E_{(x,y)\sim P}[(h(x) - E_{(x)\sim P}[h(x)])^2]$

Our learning model doesn't work as expected, what is the problem?



High bias (underfit):

$J_{train}(\theta)$ will be high

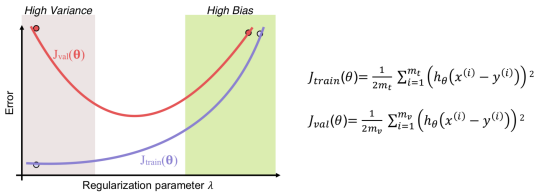
$J_{val}(\theta) \approx J_{train}(\theta)$

High variance (overfit):

$J_{train}(\theta)$ will be low

$J_{val}(\theta) \gg J_{train}(\theta)$

With the regularization we have:
$min \frac{1}{2m}[\sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \Theta_j^2]$



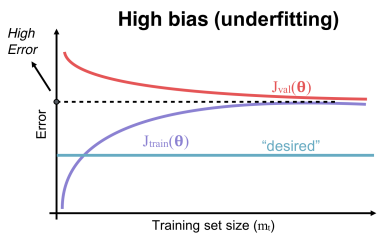
$J_{train}(\theta) = \frac{1}{2m_t}\sum_{i=1}^{m_t}\left(h_\theta(x^{(i)} - y^{(i)})\right)^2$

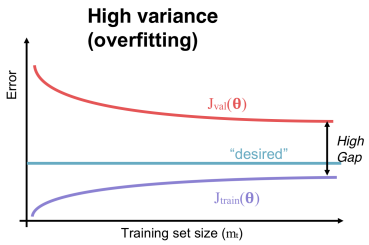
$J_{val}(\theta) = \frac{1}{2m_v}\sum_{i=1}^{m_v}\left(h_\theta(x^{(i)} - y^{(i)})\right)^2$

9.1 Learning Curves

Learning curves can be used to diagnose if a model may be suffering from bias, variance of a bit of both:

- **High bias → Underfitting:** get more data will not help much.
- **High variance → Overfitting:** getting more training data is likely to help.





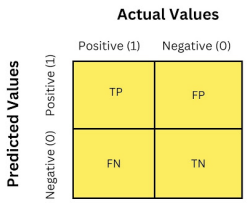
What to do next	BIAS	VARIANCE
More training data		X
Try smaller set of features		X
Try getting more features	X	
Try adding complexity to the model	X	
Try decreasing λ	X	
Try increasing λ		X

9.2 Diagnosis our Datasets

Learning curves can be also used to diagnose the quality of our training/validation sets.
Unrepresentative Training Set: the ts does not provide sufficient information to learn the problem.
Unrepresentative Validation Set: the vs might be small or too east.

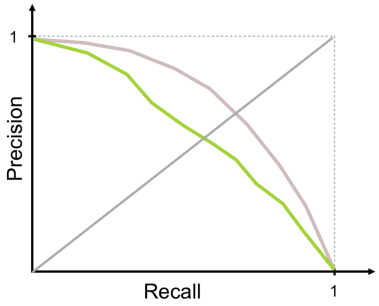
10 Metrics

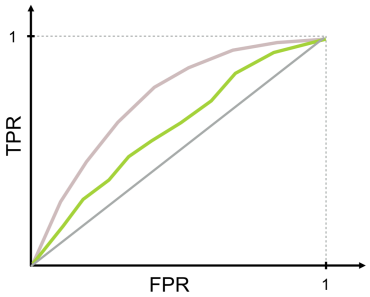
- **Accuracy**= $\frac{TP+TN}{TP+FP+TN+FN}$ = all correct / all instances.
- **Precision**= $\frac{TP}{TP+FP}$ = TP / all predicted positive.
- **Recall**= $\frac{TP}{TP+FN}$ = TP / all the ground truth instances.
- **F1-Score**= $2 * \frac{Precision*Recall}{Precision+Recall}$ = harmonic mean of Pr and Rc.



10.1 Precision-Recall Curves

PR curves are obtained by computing precision and recall figures as a function of the hyperparameters. **ROC curves** are obtained by computing precision and recall figures as a function of hyperparameters, but using FPR and TPR and than calculating the area under the curve.





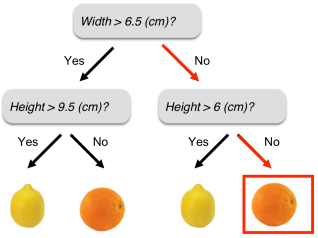
10.2 Unbalanced Data

- Use accuracy only if the classes are balanced.
- Precision-recall and mAP are usually more relevant.
- In case of multiple classes you can compute micro/macro/weighted metrics.

11 Decision Trees

A decision tree is a structure in which **internal nodes** represent **attributes**, **leaf nodes** represent **class labels** and **branching** is determined by the attribute value.
They provide interpretable results and represent the hypothesis function $h(x)$. The general idea is:

1. Pick an attribute and do a simple test.
2. Conditioned on a choice, pick another attribute.
3. In the leaves, assign a class with majority vote.
4. Do all branches as well



A Tree is built by splitting the training set into subsets which constitute the successor children.

11.1 Classification and Regression

Each path from the root to a leaf defines a **region** R_n of the input space.
Let $\{(x^{n_1}, t^{n_1}), \ldots, (x^{n_k}, t^{n_k})\}$ be the training examples falling into the region R_n.

The **classification tree** will have:

- **Discrete output** (class labels).
- Leaf y^k is set to the most common value in $\{(t^{n_1}, \ldots, t^{n_k})\}$.

The **regression tree** will have:

- **Continuous output**.
- Leaf y^k is set to the mean value in $\{(t^{n_1}, \ldots, t^{n_k})\}$.

11.2 Expressiveness

Discrete input-output: a tree can express any function of the input attributes.

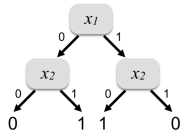
Continuous input-output: a tree can approximate any function. A decision tree represents a boolean function where each path from root to leaf represents a conjunction of test on attributes; different paths leading to the same classification represent a disjunction con conjunctions.

$x_1, x_2 \in \{0, 1\}$
$y = x_1$ XOR x_2

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

DNF for $y = 1$

(x_1=0 AND x_2=1) OR (x_1=1 AND x_2=0)



This two rules define a series of **Disjunctive Normal Form (DNF)**, one for each class.

11.3 Best Splitting

Goal: select the splitting such that the resulting areas have as many examples of the same class as possible.
Entropy (H)**:** is a measure of the amount of uncertainty (or randomness) in the dataset S. Is defined as $H(S) = -\sum_{c \in C} p(c) log_2(p(c))$, where C is the set of classes in S and $p(c)$ is the proportion of elements in C to the elements in S.
For binary classification problems we have $c \in \{c_1, c_2\}$. Therefore, it will help to define $B(q)$ as the entropy of a **Boolean r.v.** that is true with probability q, where the entropy becomes $B(q) = -(q log_2(q) + (1 - q) log_2(1 - q))$.

High entropy:

- Variable has a uniform like distribution.
- Flat histogram.
- Values sampled from it are less predictable.

Low entropy:

- Distribution of variable has peaks and valleys.
- Histogram, has lows and highs.
- Values sampled from it are more predictable.

Gini Index (Gini): measures the probability of incorrectly classifying a random chosen element if it is labelled according to the distribution of labels in the dataset. $Gini(S) = 1 - \sum_{c \in C} p(c)^2$.
Information Gain: measures the difference in entropy from before to and after the set S is split on an attribute a:

- $IG(S, a)$ measures how much uncertainty is S was reduced after splitting set S on attribute a.
- $IG(S, a) = H(s) - \sum_{t \in T} p(t)H(t) = H(S) - H(S|a)$.
- We select a that maximises $|G(S, a)|$ for the next split of the tree.

11.4 ID3-Algorithm

ID3(*Examples*, *Target_attribute*, *Attributes*)
Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
  - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
  - The decision attribute for *Root* $\leftarrow A$
  - For each possible value, v_i, of A,
    - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
    - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
    - If $Examples_{v_i}$ is empty
      - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
      - Else below this new branch add the subtree ID3($Examples_{v_i}$, *Target_attribute*, *Attributes* − {A}))
- End
- Return *Root*

11.5 Pruning

Pre-pruning (early stopping): stops the tree's growth before it comes fully developed.

- Max depth, min samples required to split, min improvement in the impurity measure.

Post-pruning (after training): starts with a fully grown tree and removes branches that provide little predictive power.

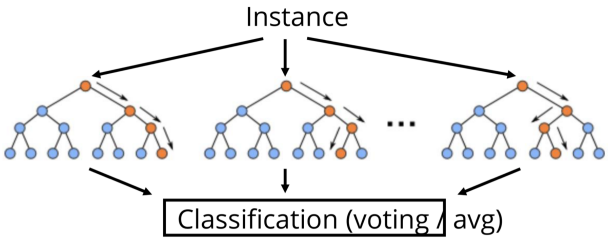
- **Cost complexity:** removes branches based on a trade-off between tree complexity and its performance on validation data.
- **Reduce error:** eliminates nodes if their removal does not increase the tree's error on validation set.

If we have **real valued attributes**, we can use IG to select threshold values. We pick the values for which adjacent labels are different, then compute the IG values and perform the split with the condition that maximizes the IG value.
It can happen that some real-world datasets **miss some features**. We can assign it the value that is most common among training examples at node n, among training examples of the same class of node n or assign a probability to each of the possible values of A.

12 Random Forest

Is a classification algorithm consisting of many decision trees:

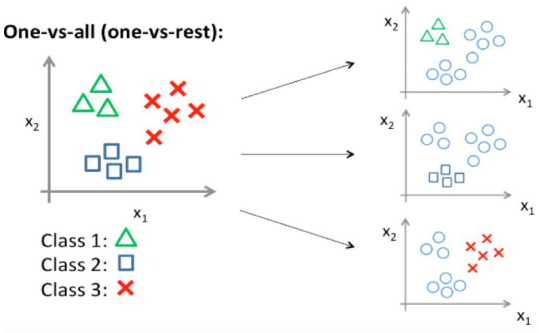
- For $b = 1, \ldots, B$:
  - Sample with replacement, n training examples from X,Y; call these X_b, Y_b.
  - Train a decision tree on X_b, Y_b.
  - After training, predictions for unseen samples x' can be made by majority voting on the predictions from all the individual trees.
- It uses bagging and feature randomness to combine trees.
- In this way it reduces the variance of the original classifier and then making an ensemble out of it.



13 Multiclass Classification

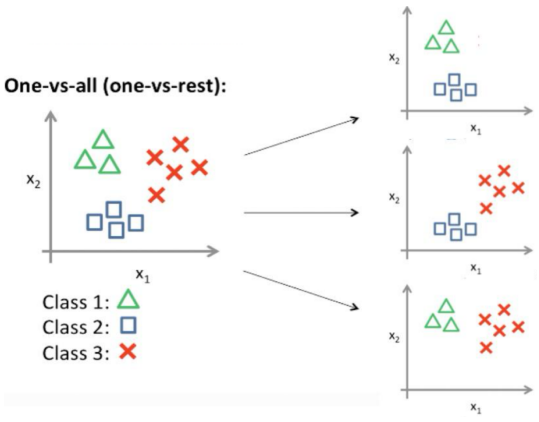
13.1 One vs All

We train a classifier per class. Each binary classifier in 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 decisions. At prediction time we select the classifier with the highest confidence score.
The multiclass label is the one of the binary classifier with the highest score $S(x)$.



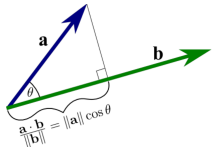
13.2 One vs One

If there are k classes in our problem, we train $k(k-1)/2$ binary classifiers. Each binary classifier is trained to discriminate between two classes.



14 Support Vector Machine (SVM)

Is a batch algorithm using generally all the training set.
Dot product: $a * b = ||a|| * ||b|| * cos(\Theta)$
Len of a projection of a onto b: $||a||cos(\Theta)$



14.1 Decision Boundary

Key Idea: instead of fitting all the points focus on boundary points.
Notation: let's represent the distance between X and W in terms of W. We are looking for the length of the projection of $x - x_p$ onto w. In our case:

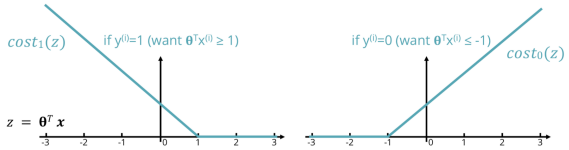
- $\frac{ab}{||b||}$ is $\frac{w}{||w||}(x - x_p)$
- $\frac{wx - wx_p + b - b}{||w||} = \frac{xw+b}{||w||}$
- $-wx_p - b = 0$

We can rescale w such that $wx + b = 1$ so the distance is $\frac{1}{||w||}$.
Finally, the margin is $\frac{2}{||w||}$ and also negative points are considered.
Objective: maximize the margin while classifying correctly: $min_w \frac{1}{2}||w||^2 = \frac{w^T w}{2}$ and $\forall i : y_i(xw_i + b) \geqslant 1$.

14.2 Hypothesis and Loss Function

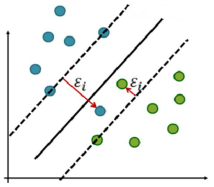
SVM's loss: $J(\boldsymbol{\theta}) = \sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$

Hinge Loss: $cost(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} \max(0, 1 - \boldsymbol{\theta}^T x^{(i)}) & \text{if } y^{(i)} = 1 \\ \max(0, 1 + \boldsymbol{\theta}^T x^{(i)}) & \text{if } y^{(i)} = -1 \end{cases}$



14.3 Non Separable Cases

If the examples are not separable, we just have to introduce slack variables to account for the violation of the constraints.
$min_w \frac{1}{2}||w||^2 + C\sum_j \epsilon_i$ and $\forall i : y_i(xw_i + b) \geqslant 1 - \epsilon_i$.
C is a penality for misclassification.



14.4 Prediction

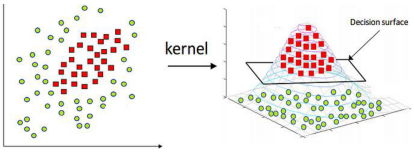
While in the training phase, we impose $\forall i : y_i(xw_i + b) \geqslant 1 - \epsilon_i$.
During prediction the discriminant functions is the same as the perceptron:

- $h(x) = sign(wx + b)$
- $h(x) = +1$ if $sign(wx + b) \geqslant 0$
- $h(x) = -1$ if $sign(wx + b) < 0$

The support vectors are those examples for which $\epsilon > 0$. The generalisation error of the SVMs is correlated to the number of support vectors (SV): $\frac{\#SV}{N-1}$.

15 Kernel Methods

A **kernel method** is any learning algorithm that uses input examples only to compute dot products.
$\phi : R^n \rightarrow R^m$, which maps our vectors in R^n to some space R^m.
Kernels are called **generalized dot products**.



Representation with kernel matrices has some advantages: same algorithm for different typologies of data.
Modularity of the design of kernel and algorithms.
The dimensionality of data depends on the number of objects and not from their vector dimensionality.

15.1 Kernelised Perceptron

In the base perceptron, if the problem is not linearly separable we can create new features by modifying the input examples via a function $\phi(x)$.
The perceptron can be written in such a way that $\phi(x)$ values are only inside dot products.
It can be shown that, for any $\phi()$ for which a dot product can be computed, there exists a (kernel) function $K()$ such that: $\phi(x^{(j)})phi(x^{(i)}) = K(x^{(j)}, x^{(i)})$.

- $i = 1; \Theta^{(1)} = 0$
- Get next example $(x^{(i)}, y^{(i)})$
  - if $\Theta^i \phi(x^{(i)}) y^{(i)} \leqslant 0$
    - $\Theta^{i+1} = \Theta^{(i)} + \eta \phi(x^{(i)}) y^{(i)}$
- $i = i + 1$

- $i = 1; M^{(1)} = \{\}$
- Get next example $(x^{(i)}, y^{(i)})$
  - if $\eta \sum_{(x,y) \in M^{(i)}} y y^{(i)} \phi(x) \phi(x^{(i)}) \leqslant 0$
    - $M^{(i+1)} = M^i + \{(x^{(i)}, y^{(i)})\}$
- $i = i + 1$

$$\eta \sum_{j=1}^{k} y^{(j)} y^{(i)} \phi(x^{(j)}) \phi(x^{(i)}) \leqslant 0 \rightarrow \eta \sum_{j=1}^{k} y^{(j)} y^{(i)} K(x^{(j)}, x^{(i)}) \leqslant 0.$$

$K()$ is a **similarity function** defined on the input space. If we know the $K()$ corresponding to the dot product in the $\phi()$ space, we could avoid to waste memory and computational time to create $\phi()$.

15.1.1 Example of Finding a Kernel

Given two vectors x and z and the following mapping $\phi()$:

$$x = (x_1, x_2); \phi(x) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$
$$z = (z_1, z_2); \phi(z) = (z_1^2, z_2^2, \sqrt{2} z_1 z_2)$$

Than:

$$\langle \phi(x), \phi(z) \rangle = \langle (x_1^2, x_2^2, \sqrt{2} x_1 x_2), (z_1^2, z_2^2, \sqrt{2} z_1 z_2) \rangle$$
$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 z_1 x_2 z_2 = \langle x, z \rangle^2$$
$$= (x_1 z_1 + x_2 z_2)^2 = K_2(x, z)$$

A dot product between $\phi(x)$ and $\phi(z)$ corresponds to evaluate the function
$K_2(x, z) = \langle x, z \rangle^2$
- $K_2(x, z)$ s faster to evaluate than $\langle \phi(x), \phi(z) \rangle$!

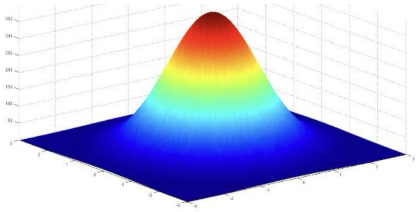
15.2 Valid Kernels

$$K = \begin{pmatrix} 1 & 0.2 & 0 & 0 \\ 0.2 & 1 & 0.8 & 0.1 \\ 0 & 0.8 & 1 & 0 \\ 0 & 0.1 & 0 & 1 \end{pmatrix}$$

A **matrix** is symmetric positive semidefinite if, for each eigenvalue $\lambda, \lambda \geqslant 0$.
A **kernel function** is positive semidefinite if every possible dataset is positive semidefinite.
If the kernel function is positive semidefinite, the learning problem of algorithms such as the SVM is convex and therefore has a global optimum.

15.3 Kernel Functions

RBF Kernel: $K(x, x') = exp(-\frac{||x-x'||^2}{2\sigma^2})$
The feature score of the kernel has an infinite number of dimensions.
You can build **novel kernel functions** with some linear transformations.



16 SVM Dual

$y_i(wx_i+b) \geqslant 1$ becomes $\alpha_i(y_i(wx_i+b)-1)$ and the whole problem becomes $\frac{1}{2}w^Tw-\sum_i \alpha_i(y_i(wx_i+b)-1)$.
we obtain the new formula: $max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_iy_j\alpha_i\alpha_j(x_i * x_j)$ subject to $\forall i : \alpha_i \geqslant 0, \sum_i y_i\alpha_i = 0$.
We want to learn α_i and α_j.
Most $\alpha_i = 0$ and the non-zero ones are called **support vectors**.

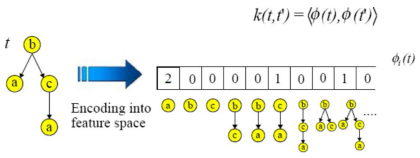
16.1 Non Separable Case

$max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_iy_j\alpha_i\alpha_j(x_i * x_j)$ subject to $\forall i : \alpha_i \geqslant 0, \sum_i y_i\alpha_i = 0$.
The support vectors are still the $\alpha \neq 0$ which now corresponds to the examples whose $\epsilon > 0$.
Primal and dual formulations have the same solution if the problem is convex.

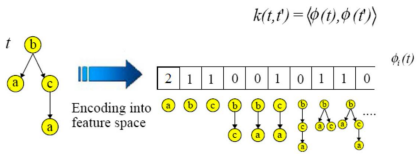
17 Machine Learning for Structured Data

17.1 Kernel Functions for Trees

Subtree Kernel: counts all matching proper subtrees of the two input trees.



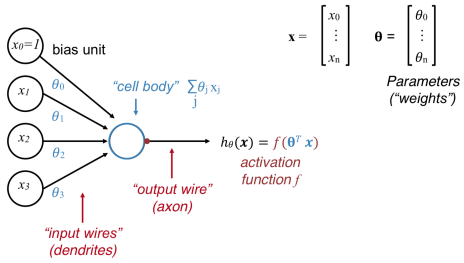
Subset Tree Kernel: counts the number of matching subset trees.

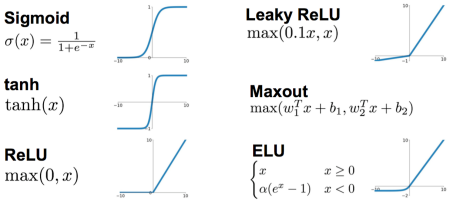


With K subsets the complexity is $O(|T_1| * |T_2|)$.

18 Neural Networks

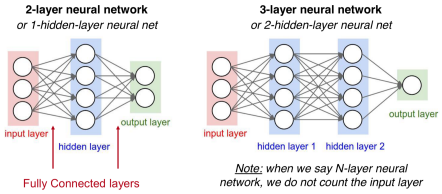
A **neural network** is just a group of this different neurons stacked up together.





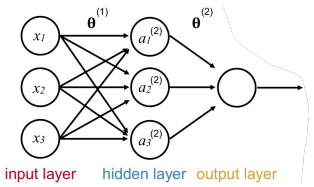
18.1 Architectures

The **architecture** refers to how the different neurons are connected to each other.
Sizing Neural Networks: the two metrics that are used to measure the size of a Neral Net are the **#neurons**, also known as **#parameters**.



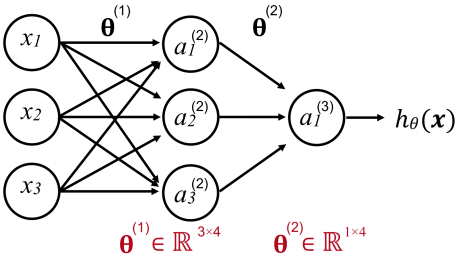
18.2 Model representation

- $a_i^{(j)}$: activation of unit i in layer j.
- $\Theta^{(j)}$: matrix of weights controlling function mapping from layer j to layer $j+1$.



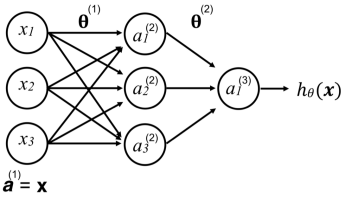
18.3 Feed-Forward Computation

Input units are set by some exterior function which causes their output links to be activated at the specified level.
Working forward through the network, these outputs are going to be the input for the next layer.



$$a_1^{(2)} = f(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = f(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = f(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$
$$h_\theta(\mathbf{x}) = a_1^{(3)} = f(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

If network has u_j units in layer j and u_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $u_{j+1} * (u_j + 1)$.



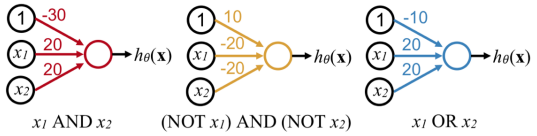
$$\mathbf{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_3 \end{bmatrix} \quad \mathbf{z}^{(2)} = \begin{bmatrix} z_1^{(2)} \\ \vdots \\ z_3^{(2)} \end{bmatrix}$$

$$a_1^{(2)} = f(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3) = f(z_1^{(2)})$$
$$a_2^{(2)} = f(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3) = f(z_2^{(2)})$$
$$a_3^{(2)} = f(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3) = f(z_3^{(2)})$$
$$h_\theta(\mathbf{x}) = a_1^{(3)} = f(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}) = f(z_1^{(3)}) = \mathbf{a}^{(3)} = f(\mathbf{z}^{(3)})$$

$$\mathbf{z}^{(2)} = \boldsymbol{\theta}^{(1)} \mathbf{a}^{(1)}$$
$$\mathbf{a}^{(2)} = f(\mathbf{z}^{(2)})$$
Add bias: $a_0^{(2)} = 1$
$$\mathbf{z}^{(3)} = \boldsymbol{\theta}^{(2)} \mathbf{a}^{(2)}$$

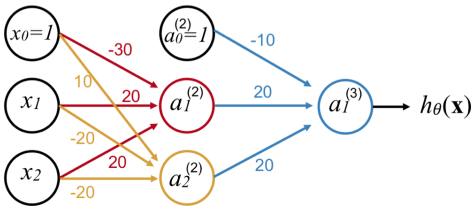
18.4 Example

Combining representations for non-linear functions:



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



19 Back Propagation in NN

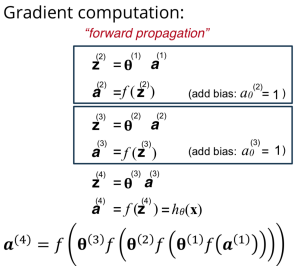
The NN **cost function** contains:

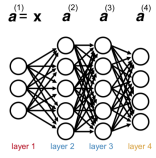
- $h_\Theta(x) \in R^k$ and $h_\Theta(x)_k = K_{th}$ output.
- L number of layers in the network
- u_i number of units in layer l.

$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)})_k) + (1 - y_k^{(i)}) log(1 - h_\Theta(x^{(i)})_k)] + \frac{\lambda}{2m}\sum_{j=1}^{L-1}\sum_{i=1}^{\mu_1}\sum_{l=1}^{\mu_1+1}(\Theta_{ij}^{(l)})^2$
We can learn our parameters with gradient descent. Remember that $\Theta_{ij}^{(l)} \in R$ connects unit j in layer l to unit i in layer $l+1$. We need to compute $\frac{d}{d\Theta_{ij}^{(l)}} J(\Theta)$.

19.1 Example

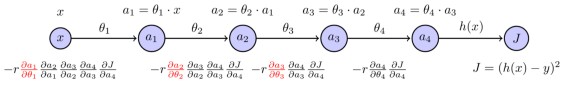
Gradient computation:





Next, in order to compute the partial derivatives, we are going to use an algorithm called "backpropagation" .

- We need to compute: $\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\boldsymbol{\theta})$
- In the example below a_4 is the output layer and θ_i are the parameters
  - **Note**: $\frac{\partial J}{\partial \theta_1}$ can be calculated directly!

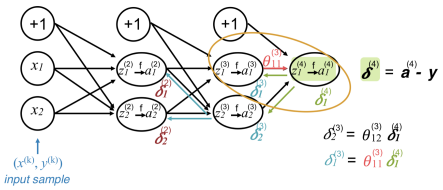


Considering the derivates of the network outputs with respect to the inputs:

- **Jacobian Matrix:** the matrix with all the first derivates.
- **Hessian Matrix:** the matrix with all the second derivates.

19.2 Back Propagation

Each unit j is responsible for a fraction of the error $\delta_j(l)$ in each of the output units to which it connects.
$\delta_j(l)$ is divided according to the strength of the connection between hidden and output units.
Then, the score is propagated back to provide the error values for the hidden layer.



19.3 Inductive Bias

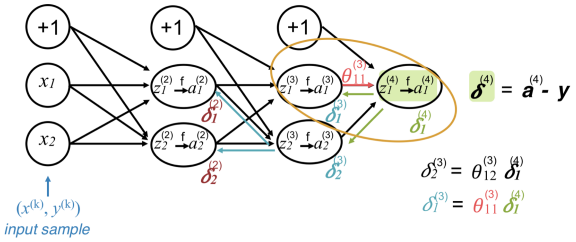
A NN with one hidden layer is able to approximate any function (with undefined # of neurons). There is no representation bias. However, the search for the best parameters is limited.

19.4 Implementation

1. Random initialization of the parameters, number of epochs. Then set mini batches, activations, ...
2. Check the loss (train and val curves) and plot also train and validation accuracy.
3. Apply gradient checking to validate your back propagation implementation.

Symmetry Breaking: a random initialization of the weights solves the problem ($\Theta_{ij}^{(l)} \in [\epsilon, -\epsilon]$).

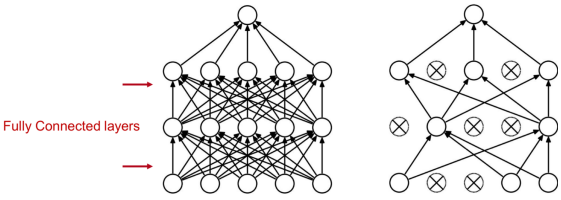
19.5 Gradient Checking



$$\frac{d}{d\Theta_j} J(\Theta) = \frac{J(\Theta_1,\ldots,+\Theta_j+\epsilon,\Theta_n) - J(\Theta_1,\ldots,+\Theta_j-\epsilon,\Theta_n)}{2\epsilon}$$

19.6 Dropout

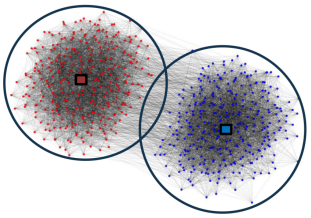
Consists in randomly deactivating some of the neurons during training (also forwarding).
Its goal is to avoid for a neuron to beccome predominant.

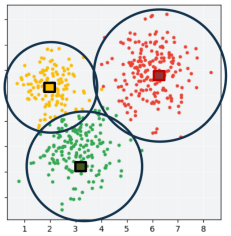


(a) Standard Neural Net (b) After applying dropout.

20 Clustering

Given a dataset $\{x^1, \ldots, x^m\}, x^n \in R^D$, we want to partition it into k clusters.
Cluster: a group of data points whose intra-class distances are small compared with the distances to points outside of the cluster.





- **Internal criteria:** depends on the notion of similarity and/or on the chosen representation:
  - We aim to maximize the intra-class similarity and minimize inter-class similarity.
  - **ill-posed problem:** the outcome depends on the similarity used and the clustering algorithm.
- **External criteria:** given an external ground truth, measure its proximity to the cluster produced.

20.1 K-Means

We want to minimize $J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$.

1. Generate k points μ_k **(prototype vectors)** in the space (cluster centroids).
2. Assign each example to the closest cluster (centroid); assign r_{nk} (= 1 if x_n has been assigned to cluster k, 0 otherwise.
3. Recalculate the position of k centroids as means of the vectors of the examples belonging to the respective clusters.
4. Repeat steps 2-3 until centroids stabilize.

μ_k and r_{nk} can't be optimized together. We want to minimize J with respect to μ_k and r_{nk}:

1. We chose some initial values for the μ_k.
2. Then we minimize J with respect to the r_{nk} keeping μ_k fixed.
   - The x_n are independent, so we can simply set $r_{nk} = 1$ for the μ_k closer to x_n, i.e. minimizing $||x_n - \mu_k||^2$.
3. We minimize J with respect to the μ_k keeping r_{nk} fixed.
   - We set the derivate of J with respect to μ_k to 0 and solve for μ_k.

$$2 \sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0 \qquad \mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$$

Convergence is assured. However it may converge to a local than rather global minimum of J. The initial choice of μ_k might have influence of the outcome of the clusters, so we mights try different initializations.

- **Course of dimensionality:** in high dimensional spaces, data points become sparse, making cluster differentiation difficult.
- **Overfitting and Noisy Features:** In high dimensional datasets, irrelevant features can distort clustering results.

We can run the K-nearest for $k = 2, 3, \ldots$ and again compute the J values (model selection) and we pick the k for which the error reduction significantly decreases.

- **Rand Index(RI):** evaluates the agreement between the clustering results generated by an algorithm and a reference (ground truth) clustering $RI = \frac{TP+TN}{TP+TN+FP+FN}$.
- **Adjusted RI:** takes into account the random probability of assignment.

21 Ensemble Methods

General idea: get predictions from multiple models and aggregate the predictions.
Classification: an ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples.
Let t be the true value, $x_1 \dots x_M$ predictions. With the ambiguity theorem we have

$$\epsilon_{\text{rand}} = \epsilon_{\text{aver}} + \frac{1}{M}\sum_{i=1}^{M}(x_i - \bar{x})^2 \implies \epsilon_{\text{aver}} \leqslant \epsilon_{\text{rand}}$$

Why and when a combination of classifiers is justified?

- **Statistical:** many hypotheses can have the same level of accuracy on the training data. By averaging the votes of several classifiers, the risk of choosing the wrong classifier reduces.
- **Computational:** an ensemble constructed by running the local search from many different starting points may provide a better approximation to the true unknown function.
- **Representational:** by forming weighted sums of hypotheses drawn from H it may be possible to expand the space of representable functions.
- **Variance-bias decomposition:** averaging multiple hypotheses reduces variance, but can also reduce the bias.

21.1 Types of Ensemble

Parallel: these methods take advantage of the independence between the base learners

- Voting = pick the prediction with the highest number of votes $\implies$ Errors of voters are not independent (big problem).
- Bagging = multiple base learners are built from different samples of the training set to make predictions.

Suppose that each binary classifier h_i has an equal and independent generalization error ϵ. Let us combine T of such classifiers according to $H(x) = \text{sign}\left(\sum_{i=1}^{T} h_i(x)\right)$
Sequential: These methods take advantage of the dependence between the base learners since the overall performance can be boosted incrementally (**boosting**).

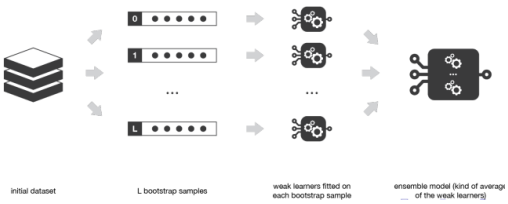
21.2 Bootstrapping

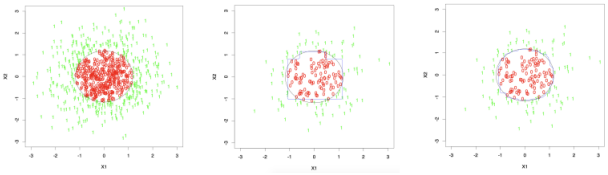
Bootstrapping: sample with replacement M overlapping groups of instances of the same size.
Feature randomization: each model sees a random subset of features.

21.2.1 Bootstrap Aggregating Approach

Ideally, bagging eliminates variance altogether while keeping the bias almost unchanged; Averaging reduces variance: let $Z_1, \dots, Z_N$ be iid random variables, then $Var(\frac{1}{N}\sum_i Z_i = \frac{1}{N}Var(Z_i)$.
In practice, **weak learners are not independent** hence bagging tends to reduce variance and increase bias.
Bagging is bad if models are very similar (not independent enough). This happens if the learning algorithm is **stable**, i.e. models do not usually change much after changing a few instances.
Bagging is strongly affected by the quality of individual models.

- Create k bootstrap samples.
- Train a distinct classifier on each sample.
- Classify new instance by majority vote/average.



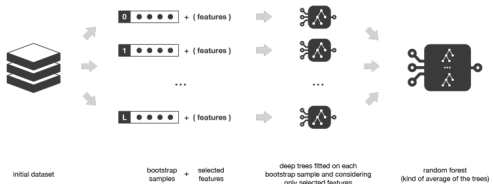


(a) Data (b) Single DT (c) Ensemble of DTs

21.3 Bagging

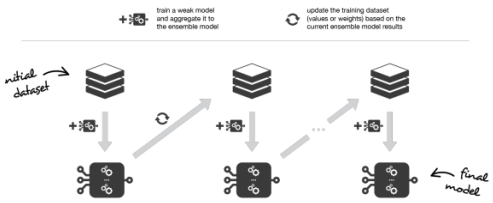
21.3.1 Random Forest Algorithm

- Use k bootstrap replicates to train k different decision trees (DTs).
- At each node, pick a subset of features at random.
- Aggregate the predictions of each tree to make classification decision.



21.4 Boosting

- Use the training set to train a simple (weak) predictor.
- Re-weight the training examples, putting more weight on examples that were poorly classified in the previous predictor.
- Repeat n times.
- Combine the simple hypothesis into a single accurate predictor.

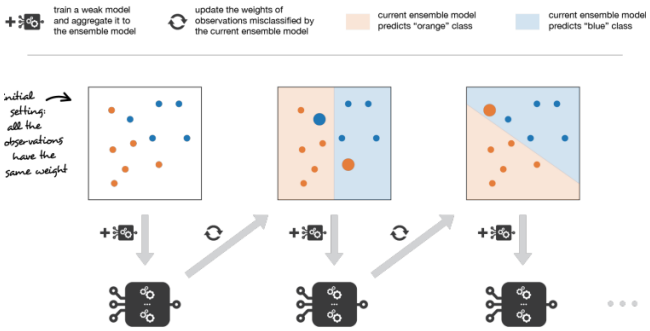


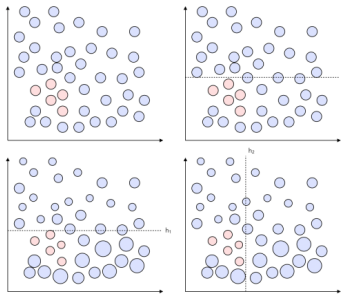
Boosting assumes weak learners are slightly better than random guessing.
It reduces bias by making each classifier focus on previous mistakes.
Through **sequential training with example re-weighting** we can take a weak classifier slightly better than chance and boost it to get a low training error.
Hypothesis: $H(x)0sign(\sum_t \alpha_t h_t(x))$. Note that we sum predictions (after sign) so, for example, sum of linear classifier isn't a linear classifier.

21.4.1 Adaboost (Adaptive Boosting)

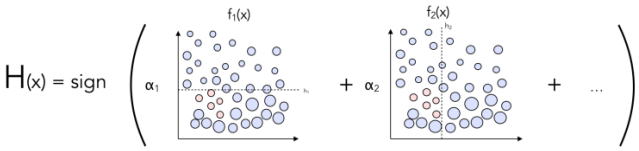
IDEAM: At each iteration t the training sample is reweighted (D_t), giving larger weights to point that were classified wrongly and train a new weak classifier.
Weak learners need to maximize **weighted accuracy** ($e_t = P_{i\sim D_t}[h_t(x_i) \neq y_i] = \sum_i D_t(i)[h_t(x) \neq y_i]$.
The weight of each classifier is computed according to its weighted error $\alpha_t = \frac{1}{2} log(\frac{1-\epsilon_t}{\epsilon_t})$.
Instances weights D_t are updated using an exponential rule, so harder examples weight exponentially more than easy ones.
Loss Function: $E = \sum_{i=1}^{n} e^{-y_i H(x_i)}$ where $H(x) = \sum_t \alpha_t h_t(x)$.

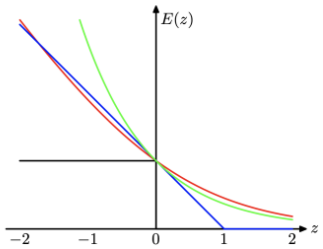
Bagging may fail if the considered weak learners are mistaken in the same region, because boosting solves the problem by concentrating the efforts on those regions.





$$E = \sum_{i=1}^{n} e^{-y_i H(\mathbf{x}_i)}$$





21.5 Stacking

Both bagging and boosting assume we have a single base learning algorithm. What if we want to combine an arbitrary set of classifiers?

Stacking is a technique for combining an arbitrary set of learning models using a meta-model.
Stacking works at its best when the base models have **complementary strengths and weaknesses**.