MachLe - Olivier D'Ancona

Evaluation Metrics

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$
$$Precision = \frac{TP}{TP+FP}$$
$$Recall = \frac{TP}{TP+FN}$$
$$Specificity = \frac{TP}{TN+FP}$$
$$Fscore = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$
$$error\ rate = 1 - accuracy$$
$$macro\ average = \frac{1}{n}\sum_{i=1}^{n} avg_i$$

Activation Functions

Sigmoid : $\sigma(x) = \frac{1}{1+e^{-x}}$

Hyperbolic tangent : $\frac{e^x - e^{-x}}{e^x + e^{-x}}$

Relu : $\begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$

Gaussian : e^{-x^2}

Softmax : $\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

Normalization

Min-max [0,1] : $x' = \frac{(x - x_{min})}{(x_{max} - x_{min})}$

Min-max [-1,1] : $x' = 2 \cdot min\_max(x) - 1$
min-max doesn't handle outliers.

Z-norm : $x' = \frac{(x-\mu)}{\sigma}$

Logistic Regression

$$h_\theta(x_n) = \sigma(x\theta^T)$$

- $h_\theta(x_n)$: predicted value
- θ : model's parameters
- X : input vector

Goal : Find the θ that maximizes the likelihood of the data.
Loss :

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(h_\theta(x_n)) + (1-y_i)\log(1-h_\theta(x_n))$$

Bayes

$$P(C_k|x) = \frac{P(x|C_k) \cdot P(C_k)}{P(x)}$$

où

- C_k : Classe ciblée
- x : Évidence
- $P(C_k)$: Probabilité a priori de la classe C_k
- $P(x|C_k)$: probability of observing x given class j
- $P(C_k|x)$: Probabilité a posteriori de la classe C_k après observation de x
- $P(x)$: Probabilité de l'évidence x

avec

$$P(x) = \sum_{\text{toutes classes } C_k} P(x|C_k) \cdot P(C_k)$$

Classifier H/F:

- $P(C_f) = \frac{4}{70}, P(C_g) = \frac{66}{70}$
- $p(x|C_g) = 0.8, p(x|C_f) = 0.2$
- Calcul de $p(x)$:

$$p(x) = 0.2 \times \frac{4}{70} + 0.8 \times \frac{66}{70}$$

- Calcul de $P(C_f|x)$ et $P(C_g|x)$:

$$P(C_f|x) = \frac{0.2 \times \frac{4}{70}}{p(x)}, \quad P(C_g|x) = \frac{0.8}{p($$

(+)Can deal with imbalanced dataset, prior can be changed

KNN

Hyperparameters :

- Number of neighbours k
- Distance metric
- normalization type
- strategy if no majority

Big k :
(+) More confidence, probabilistic (-) No locality, heavier

Linear Regression

Soit un tableau de données :
x = Surface(g) , y = Price(cm) , $x \cdot y$, x^2

$$X = [1, Surface]$$

$$X^T X = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} 7 & 38.5 \\ 38.5 & 218.95 \end{bmatrix}$$

$$X^T y = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix} = \begin{bmatrix} 348 \\ 1975 \end{bmatrix}$$

$$\hat{\theta} = (X^T X)^{-1} X^T y = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} -2.67 \\ 9.51 \end{bmatrix}$$

$$\hat{y} = \theta_0 + \theta_1 x$$

Matrix Inversion (2x2)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Similarity Measures

$$Pearson = R^2 = 1 - \frac{\sum_1^n (y_i - \hat{y}_i)^2}{\sum_1^n (y_i - \overline{y_i})^2}$$

$$Euclidian = \sqrt{\sum (I_1 - I_2)^2}$$

$$Manhattan = \sum |I_1 - I_2|$$

$$MSE = \frac{1}{N}\sum_1^n (y_i - \hat{y}_i)^2$$

$$Cosine\ similarity = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

$WSS = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i)^2$ Within-cluster-sum (distortion) is the sum of the squared distances between each point in a cluster x_j and its cluster center.

Support Vector Machine

Concept : SVM finds the hyperplane that best separates different classes by maximizing the margin between the closest points of different classes (support vectors).
hw(x) = sign(b + wx)

Formulation

$$\max_{\omega,b} \frac{1}{\|\omega\|} \quad \text{s.t.} \quad y_i(\omega \cdot x_i + b) \geq 1 \, \forall i$$

where

- ω : Normal vector to the hyperplane
- b : Bias term
- x_i, y_i : Training data points and labels

Kernel Trick

SVM can be extended to non-linearly separable data using kernel functions, which implicitly map input space to a higher-dimensional feature space

Common Kernels

- Linear : $\langle x, x' \rangle$
- Polynomial : $(\gamma \langle x, x' \rangle + r)^d$
- Gaussian (RBF) : $e^{(-\gamma \|x - x'\|^2)}$

(+) Effective in high-dimensional spaces, Memory efficient, Versatile (different kernel functions)

(-) Sensitive to the choice of kernel and regularization parameters, Not suitable for very large datasets
hinge loss : $max(0, 1 - y_i(w \cdot x_i + b))$ (0 if correct classification) (1 if falls on the hyperplane) (>1 if misclassified)

Objective function to min

$$\min_{\omega,b} \quad \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(\omega \cdot x_i + b))$$

where C nutch the hinge loss term (how far are we predicting from ground truth) and regularization term (impeach big value, min w => maximize margin)

Clustering

Clustering partitions data into clusters with high intra-class similarity and low inter-class similarity.
Needs : distance measure, criterion, algorithm.

Partitions

Distortion : How close are we to a "centroid" defining the partition ?
Connectivity of points : How close are points to each other ?

Elbow Method

Heuristic used in determining the number of clusters in a data set. It selects the value of k that corresponds to the elbow of the curve (#cluster, WSS).

Silhouette Coefficient

$$s = \frac{b - a}{\max(a, b)}$$

- a is the mean distance between a sample and all other points in the same class (cohesion)
- b is the mean distance between a sample and all other points in the next nearest cluster (isolation)

s range is $[-1, 1]$. A high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

Davies-Bouldin Index

$$DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} \frac{R_i + R_j}{d(C_i, C_j)}$$

- R_i is the average distance between a point in cluster C_i and all points in C_i (cluster diameter))
- $d(C_i, C_j)$ is the distance between the centroids of C_i and C_j

zero is the lowest possible score. Values closer to zero indicate a better partition.

K-Means

1. Initialize k centroids randomly.
2. Assign each point to the nearest centroid.
3. Recompute centroids as the mean of assigned points.
4. Repeat steps 2-3 until convergence.

minimize distortion : $J = \sum_{i=1}^{k} d(x_n, \mu_c)$

(+) Will converge
(-) Sensitive to initial conditions (may converge to local minima)

DB-Scan

1. Classify points as core, border, or noise based on density.
2. Form clusters around core points.
3. Assign border points to clusters or mark as noise.

(+) Identifies clusters of varying shapes; robust to noise.
(-) Sensitive to parameters; struggles with varying density clusters.

Mean Shift Clustering

1. Choose bandwidth and initialize centroids.
2. Shift each centroid to the mean of points within the bandwidth.
3. Repeat until centroids converge.

(+) Can find clusters of arbitrary shape; robust to outliers.
(-) Computationally intensive; bandwidth parameter can be tricky to set.

Hierarchical Clustering

Algorithm (Agglomerative) :

1. Start with each point as a separate cluster.
2. Merge the closest pair of clusters.
3. Repeat step 2 until desired number of clusters is reached.

(+) No need to specify the number of clusters; intuitive dendrogram representation.
(-) Computationally expensive for large datasets; sensitive to outliers.

Entropy

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

where
- $p(x_i)$: Probability of class x_i

Information Gain :

$$IG(X, Y) = H(X) - H(X|Y)$$

where
- $H(X)$: Entropy of the parent node
- $H(X|Y)$: Entropy of the child node

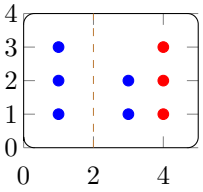
Gini Impurity

Gini Set :$G(X) = 1 - \sum_{i=1}^{n} p(x_i)^2$ where
- $p(x_i)$: is the proportion of points in a set that belongs to a class i : $\frac{N_i}{N}$.
- G = 0.5 : maximum value of impurity, classes are balanced in the set.
- G = 0 : minimum value of impurity, all the values belong to a single class.

Gini Split :$\sum_{i=1}^{n} \frac{N_i}{N} G(X_i)$

Gini Gain(big=good) :$Gini_{set} - Gini_{split}$

Example



Gini set : $1 - \left(\frac{5}{10}\right)^2 - \left(\frac{5}{10}\right)^2 = \frac{1}{2}$

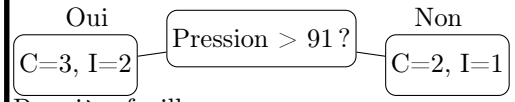
Left Set : $1 - \left(\frac{3}{3}\right)^2 - \left(\frac{0}{3}\right)^2 = 0$

Right Set : $1 - \left(\frac{2}{7}\right)^2 - \left(\frac{5}{7}\right)^2 = \frac{20}{49}$

Gini Split : $\frac{3}{10} \times 0 + \frac{7}{10} \times \frac{20}{49} = \frac{2}{7}$

Gini Gain : $\frac{1}{2} - \frac{2}{7} = \frac{3}{14}$

Example2



Première feuille :

$$Gini = 1 - \left(\frac{3}{3+2}\right)^2 - \left(\frac{2}{3+2}\right)^2 = \frac{12}{25}$$

Deuxième feuille :

$$Gini = 1 - \left(\frac{2}{2+1}\right)^2 - \left(\frac{1}{2+1}\right)^2 = \frac{4}{9}$$

Decision Tree

A flowchart-like structure in which each internal node represents a test on a feature. Each leaf node represents a class label(decision taken after computing all features).

Boosting

Build additional trees while considering earlier trees to compensate for their weaknesses

Bagging

Build a lot of trees using different parts of the data without looking at earlier ones. A very popular algorithm that is close to this approach is called "Random Forests"
Pruning : Removing sections of the tree that provide little power to classify instances.
(+)
(-)

Overfitting

Overfitting occurs when a model learns the training data too well, including noise and outliers, resulting in poor generalization to new data.
Signs :
- High accuracy on training data
- Poor performance on validation/test data

Techniques to Reduce Overfitting :
- **Early Stopping** : Stop training when performance on validation data begins to degrade.
- **Regularization** : Add a penalty term to the loss function (L1, L2 regularization).
- **More Data** : Increase the size of the training dataset.
- **Data Augmentation** : Artificially increase the diversity of the training dataset by creating modified versions of the data.
- **Dropout** : Randomly omit a subset of features/neurons during training to prevent co-adaptation.
- **Simplifying the Model** : Reduce the complexity of the model (fewer layers or hidden units).
- **Cross-Validation** : Use cross-validation to ensure the model's ability to generalize.
- **Ensemble Methods** : Combine predictions from multiple models to reduce variance.

Convolutional Neural Networks

Features

colors, terrain texture, size, presence of straight lines, border
1. Extracting localized low-level features
2. Incrementally allow the system to appropriately bind together features and their relationships
3. Gradually build-up overall spatial invariance

Pooling layer

Maxpool after a convolution layer eliminates non maximal values : it is a form of non-linear down-sampling that reduces computation for upper layers and provides a "summary" of statistics of features in lower layers.

Convolution layer

Different kernel sizes (3x3, 5x5, 7x7, etc) allows the identification of features at different scales and multiple layers of 3x3 kernels can implement other kernel sizes
The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. We can compute the spatial size of the output volume as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (F),the stride with which they are applied (S), and the amount of zero padding used (P) on the border.
formula for calculating how many neurons "fit" is given by $\frac{W-F+2P}{S}+1$

Input Volume

Size : $W_1 \cdot H_1 \cdot D_1$

Hyperparameters

K : Number of filters
F : Their spatial extent
S : Stride
P : The amount of zero padding

Output Volume

Size : $W_2 \cdot H_2 \cdot D_2$
H2 : $\frac{W_1-F+2P}{S}+1$
W2 : $\frac{H_1-F+2P}{S}+1$
D2 : K

Parameters

It introduces $(F \cdot F \cdot D_1) \cdot K$ weights plus K biases.

XG-Boost

Instead of yes/no leaf outputs, we have real-valued weights.
If $\sigma(w_i) > 0.5$ then output yes, else no
Use a loss function that considers the output errors and adapt the weights
New trees are built with the aim of incrementally reducing the error (boosting)
we use an iterative optimization approach : in every boosting iteration we choose a tree f_k that will get us one step closer to the minimum cost.
XGBoost, in short, uses several trees that contribute in an additive manner to compute a nal weight value for each leaf by minimising a loss function that measures the difference between the true labels y and the ensemble's prediction

Neural Network

Structure

Biais : b, An extra weight that can be learned using a learning algorithm. The purpose is to replace threshold.
Input : I, Input vector
Weights : W, Vector of weights

Learning algorithm

1. Randomly initialize weights
2. Compute the neuron's output for a fiven input vector X
3. Update weights : $W_j(t+1) = W_j(t) + \eta\,(\hat{y_i} - y)\,x$ with η the learning rate and $\hat{y_i}$ the desired output.
4. Repeat steps 2 and 3 for the number of epochs you need or until the error is smaller than a threshold.

Regularization

Regularization means providing constraints to limit the values of the parameters we are learning.

LSTM

Long Short-Term Memory NN are a type of RNN designed to remember information for long periods as part of the model's internal state.

Key Components :

- Input, output, and forget gates to regulate the flow of information.
- Cell state for storing long-term information.
- Hidden state for short-term information.

Functioning : Gates selectively add or remove information to the cell state, allowing LSTMs to capture long-term dependencies and mitigate the vanishing gradient problem.

(+) Better at capturing long-range dependencies than standard RNNs, More effective in learning from large sequences of data.
(-) More complex and computationally intensive to train than standard RNNs, May require more data to train effectively.

Number of Weights

$4x(N_{inputs} + N_{LSTMblocks} + bias)xN_{LSTMblocks}$
Weights for 32 LSTM units and 2-dim inputs : 4 x (2 + 32 + 1) x 32 = 4480

Applications

feedforward NN, text and video classification, image captioning, sequence to sequence task, generative text model.

Recurrent Neural Networks

Handle sequential data. Each neuron in an RNN has a self-loop that allows information to persist.Maintains a hidden state that captures information about the sequence.

Applications :

- Natural Language Processing (NLP)
- Speech Recognition
- Time Series Prediction

(+) Can handle variable-length sequences, Can capture long-term dependencies
(-) Computationally intensive, Suffers from vanishing/exploding gradients, hard to lwarn long-term dependencies

Principal Component Analysis

Kmeans

Hierarchical clustering

Autoencoders

Kmeans

Hierarchical clustering

Dimensionality Reduction

Reinforcement Learning

Computational Complexity of ML Algorithms

Algorithm	**Assumption**	**Train Time/Space**	**Inference Time/Space**
KNN (Brute Force)	Similar things exist in close proximity	$O(knd)$ / $O(nd)$	$O(knd)$ / $O(nd)$
KNN (KD Tree)	Similar things exist in close proximity	$O(nd\log(n))$ / $O(nd)$	$O(k\log(n)d)$ / $O(nd)$
Naive Bayes	Features are conditionally independent	$O(ndc)$ / $O(dc)$	$O(dc)$ / $O(dc)$
Logistic Regression	Classes are linearly separable	$O(nd)$ / $O(nd)$	$O(d)$ / $O(d)$
Linear Regression	Linear relationship between variables	$O(nd)$ / $O(nd)$	$O(d)$ / $O(d)$
SVM	Classes are linearly separable	$O(n^2d^2)$ / $O(nd)$	$O(kd)$ / $O(kd)$
Decision Tree	Feature selection by information gain	$O(n\log(n)d)$ / $O(\text{nodes})$	$O(\log(n))$ / $O(\text{nodes})$
Random Forest	Low bias and variance trees	$O(kn\log(n)d)$ / $O(\text{nodes}\times k)$	$O(k\log(n))$ / $O(\text{nodes}\times k)$
GBDT	High bias, low variance trees	$O(Mn\log(n)d)$ / $O(\text{nodes}\times M+\gamma_m)$	$O(M\log(n))$ / $O(\text{nodes}\times M+\gamma_m)$