Continuous-Continuous:

Pearson's Correlation if normally distributed:

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} \quad r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

Spearman correlation (works with ranks) if not:

$$r_s = \rho_{\mathrm{R}(X),\mathrm{R}(Y)} = \frac{\mathrm{cov}(\mathrm{R}(X),\mathrm{R}(Y))}{\sigma_{\mathrm{R}(X)}\sigma_{\mathrm{R}(Y)}}$$

se todos os ranks forem distintos

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

Categorical-Categorical:

Goodman Kruskal Lambda:

$$\lambda_{B|A} = \frac{P_E - P_{E|A}}{P_E}$$

P_E is the error in predicting B **without** knowing A

$P_{E|A}$ is the error in predicting B **with** knowledge of A

- without knowing A: $P_E = 1 - \frac{\max_C N_{+C}}{N_{++}} = 1 - \frac{700}{300+700}$

- knowing A: $P_{E|A} = 1 - \frac{\sum_r \max_C N_{rC}}{N_{++}} = 1 - \frac{118+628}{300+700}$

Chi Square:

$$\chi 2 = \sum \frac{(O^i - E^i)^2}{E^i}$$

, O_i = Observed frequency for the i[th] category. E_i = Expected frequency for the i[th] category,

$$E_{ij} = \frac{(Row\ Total) * (Column\ Total)}{Grand\ Total}$$

df = (nRows – 1) * (nColumns – 1)

Continuous-Categorical:

2 classes: **t-Test** (normally distributed), **Mann-Whitney U Test** (one is not normally distributed)

More than 2: **ANOVA**, **Kruskal-Wallis**

Color: [Red, Blue, Green, ...]
- Nominal Scale

Size: [Small, Medium, Large]
- Ordinal Scale

Price: [3.65, 2.98, 6.12, ...]
- Continuous

Petals: [5, 6, 4, 7, ...]
- Discrete

Nominal – Tags
Ordinal – Rank

Error Rate (tudo menos diagonal):

$$\mathrm{ER} = \frac{FP+FN}{TP+TN+FP+FN} = \frac{FP+FN}{S}$$

C: number of classes
N_i: number of samples of class i

Accuracy (diagonal):

$$Acc = 1 - \mathrm{ER} = \frac{TP+TN}{S}$$

Balanced Accuracy:

$$BAcc = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i+FN_i} = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{N_i}$$

Z-Score:

$$Z = \frac{X-\mu}{\sigma}, \quad \sigma = \sqrt{\frac{\sum(x_i-\mu)^2}{N}}$$

|Z| > k = outlier

SMOTE – Oversampling Technique:

New Point = Starting Point + Fraction × (Target Point − Starting Point)

$$x = x_i + rand\ (0-1)(\hat{x}_i - x_i)$$

K-Nearest Neighbors:

- Calcular **distância** entre um ponto e todos os outros. – Selecionar os **k pontos** mais **próximos**. – Para **classificar**, entre os k, ver a classe que existe em **maioria**.

PCA:

- Centralizar usando **z-score**. – Achar a **matriz de co-variância**, para conseguir achar os **eigenvalues** e os **eigenvectors**. – Ordenar eigenvalues e consoante essa ordenação, ordenar os eigenvectors. – **Projetar** na matriz original (**matrizO x eigenvectors ordenados**)

Explained Variance:

$$\beta_i = \frac{\lambda_i}{\sum \lambda_i}$$

Confusion Matrix:

		Predicted	
		P	N
Real	P	TP	FN
	N	FP	TN

$P = TP + FN$

$N = TN + FP$

Correct classification:
- **TP = true positives (P classified as P)**
  - Malignant tumors classified as malignant
- **TN = true negative (N classified as N)**
  - Benign tumors classified as benign

Misclassification:
- **FN= false negatives (P classified as N)**
  - Malignant tumors classified as benign
- **FP= false positives (N classifies as P)**
  - Benign tumors classified as malignant

Class True Positive Rate:

$$TPR_i = \frac{TP_i}{P_i} = \frac{TP_i}{TP_i+FN_i}$$

Pi: positives for class i
C: Nº of classes
S: Nº of test instances

Global TP Rate weighted = Acc

Global TP Rate macro = BAcc

Global TP Rate micro = Acc

Class False Positive Rate:

$$FPR_i = \frac{FP_i}{N_i} = \frac{FP_i}{FP_i+TN_i}$$

Global FP Rate w:

$$FPR_w = \frac{1}{S}\sum_{i=1}^{C} P_i \cdot FPR_i$$

Global FP Rate ma:

$$FPR_M = \frac{1}{C}\sum_{i=1}^{C} FPR_i$$

Global FP Rate mi:

$$FPR_m = \frac{\sum_{i=1}^{C} FP_i}{\sum_{i=1}^{C} N_i} = \frac{1}{2S}\sum_{i=1}^{C} FP_i$$

Class Precision:

$$Pr_i = \frac{TP_i}{TP_i+FP_i}$$

Global Precision w:

$$Pr_w = \frac{1}{S}\sum_{i=1}^{C} P_i \cdot Pr_i$$

Global Precision ma:

$$Pr_M = \frac{1}{C}\sum_{i=1}^{C} Pr_i$$

Global Precision mi = Acc

Class Recall:

$$Re_i = \frac{TP_i}{TP_i+FN_i}$$

All Global Recall = Acc

Class F1-Score:

$$F1_i = 2\frac{Pr_i \cdot Re_i}{Pr_i + Re_i}$$

Global F1 w:

$$F1_w = \frac{1}{S}\sum_{i=1}^{C} P_i \cdot F1_i$$

Global F1 ma:

$$F1_M = \frac{1}{C}\sum_{i=1}^{C} F1_i$$

Global F1 mi = Acc

Fisher Score:

$$F(x) = \frac{\sum_{k=1}^{K} N_k(\mu_k-\mu)^2}{\sum_{k=1}^{K} N_k\sigma_k^2}$$

- K is the number of classes.
- N_k is the number of samples in class k.
- μ_k is the mean of the feature x in class k.
- μ is the overall mean of the feature x across all classes.
- σ_k^2 is the variance of the feature x in class k.

Binning:

Divide the range of continuous attributes into intervals.

Equal width binning: n intervals of equal size

$$width = \frac{B-A}{N}$$

Equal depth binning: same number of samples in each interval

Entropy based binning:

Find the binary split boundary that minimizes the entropy function over all possible boundaries:

$$H(S_i) = -\sum_{j=1}^{N} p_{ij}\log_2 p_{ij} \quad H(S,T) = \sum_{i=1}^{K}\frac{|S_i|}{|S|}H(S_i)$$

Greater the gain: better the split $Gain = H(S) - H(S,T)$

Character Based Similarity:

Edit Distance: Insert, delete and replace – cost = 1

Affine Distance: Insert, delete, replace, open gap, extend gap

S1 = Chris R Lang
S2 = Christopher Richard Lang

Open gap – 1
Extend "topher" – 2.5 (5 * 0.5)
Open gap – 1
Extend "ichard" – 2.5 (5 * 0.5)
- Distance = 1 + 2.5 + 1 + 2.5 = 7

Smit-Waterman Distance: Beginning and end mismatches- < weights; ignored.

Token Based Similarity:

$$Overlap(S_1,S_2) = \frac{|tok(S_1)\cap tok(S_2)|}{\min(|tok(S_1)|,|tok(S_2)|)} \quad Jaccard(S_1,S_2) = \frac{|tok(S_1)\cap tok(S_2)|}{|tok(S_1)\cup tok(S_2)|} \quad Dice(S_1,S_2) = \frac{2|tok(S_1)\cap tok(S_2)|}{|tok(S_1)|+|tok(S_2)|}$$

Discard Data:

Complete case – eliminar linha

Pairwise – eliminar o valor

Dropping variables – eliminar coluna

Phonetic Similarity:

(1) - Keep the first letter of the surname and ignore all occurrences of W and H (Peter – P) **(2)** - Assign following codes: (B, F, P, V = 1), (C, G, J, K, Q, S, X, Z = 2), (D, T = 3), (L = 4), (M, N = 5), (R = 6) (Pe3e6) **(3)** - A, E, I, O, U and Y are not coded and serve as separators (0) (P0306) **(4)** - Consolidate sequences of identical codes by keeping only the first occurrence (P0306) **(5)-** Drop separators (P36) **(6)** -Keep the letter prefix and the three first codes, padding with zeros if there are fewer than 3 codes (P360)

Undersampling:

Random Undersampling: random elimination of majority samples

Tomek Links: For every sample Ei in the dataset, find its nearest neighbor Ej. - If Ei and Ej belong to **different classes**, proceed to the next step. - Verify that Ei and Ej are the **closest to each other** compared to other points in the dataset. - If the conditions above are met, remove one or both points in the pair.

Undersampling:

Condensed Nearest Neighbor Rule: Eliminate the samples of the majority class that are **far away from the decision border.**

One Sided Selection: Tomek Links + CNN

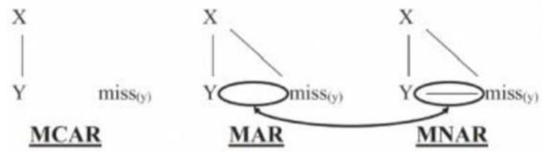
Neighborhood Cleaning Rule: Clean the data while reducing it. - Get the 3-NN of a data sample: If data sample of majority class and 3-NN contradict classification eliminate data sample else if data sample of minority class and 3-NN contradict classification eliminate data samples of 3-NN that belong to majority class

Oversampling:

Naïve Random Oversampling: Randomly duplicate (with replacement) minority data points

SMOTE – já falado antes

MCAR, MAR, MNAR:



Imbalance Ratio:

$\boxed{\text{IR} = \#\text{Positive Class}/\#\text{Negative Class}}$

Linear Regression:

$$\underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{p}\left(y_i - \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_p x_{i,p}\right)^2 = \underset{\beta}{\operatorname{argmin}} \|Y - X\beta\|^2$$

$$\frac{\partial}{\partial \beta}\|Y - X\beta\|^2 = 0 \Leftrightarrow \beta = \underbrace{\left(X^T X\right)^{-1} X^T}_{X^+} Y$$

Pseudo-inverse

LDA e FDA:

Detect direction that maximizes the separation between classes

Maximum separation between means of projected classes and **Minimum variance within** each projected class

$$\max\left\{\frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}\right\}$$

Find a vector w that maximizes:

$$\underset{w}{\operatorname{argmax}}\left(\frac{(\mu_1^T w - \mu_2^T w)^2}{w^T \Sigma_1 w + w^T \Sigma_2 w}\right) \Leftrightarrow \underset{w}{\operatorname{argmax}}\left(\frac{\left(\overbrace{(\mu_1^T - \mu_2^T)}^{m^T} w\right)^2}{w^T \underbrace{(\Sigma_1 + \Sigma_2)}_{S} w}\right) \Leftrightarrow \underset{w}{\operatorname{argmax}}\left(\frac{(m^T w)^2}{w^T S w}\right)$$

- Standard deviation is the same for all classes

$$\underset{w}{\operatorname{argmax}}\left(\frac{\left((\mu_1^T - \mu_2^T) w\right)^2}{w^T S w}\right) \Leftrightarrow \underset{w}{\operatorname{argmax}}\left(\frac{w^T \overbrace{(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T}^{S_b} w}{w^T S w}\right) \Leftrightarrow \underset{w}{\operatorname{argmax}}\left(\frac{w^T S_b w}{w^T S w}\right)$$

ReliefF: (ver código)

For each sample xi update feature score W[f] for feature f based on feature distance from point xi to nearest hit Hi (same class) and nearest miss Mi (other class)

$$W[f] = W[f] - \operatorname{diff}(f, \mathbf{x}_i, \mathbf{H}_i) + \operatorname{diff}(f, \mathbf{x}_i, \mathbf{M}_i)$$

Find the closest element of the same class to be the **Hit**, and find the closest element from an-other class to be the **Miss**. When there are more than two classes, find all the misses, sum them all up, and

Sum Squared Error:

$$SSE = \sum_{i=1}^{N}(y_i - \hat{y}(x_i, w))^2$$

y_i: real values
$\hat{y}$: predicted values
N: size of the test set

Mean Squared Error:

$$MSE = \frac{1}{N} \cdot SSE$$

Root Mean Square Error:

$$RMSE = \sqrt{MSE}$$

Coefficient of Determination:

$$R^2 = 1 - \frac{SSE}{SST}$$, where $$SST = \sum_{i=1}^{N}(y_i - \bar{y})^2$$

Extreme Value Analysis:

$[\mu - k\sigma, \mu + k\sigma]$

Interquartile range:

$[Q_1 - k \times IQR, Q_3 + k \times IQR]$

IQR = Q3-Q1

K-Means:

- **k** initial cluster **centers**. – each point **allocated** to the **closest center**. - calculate the **mean** of x and y for each cluster's values (new values are the new centers). – stop when nothing changes

DBSCAN:

A point *p* is a **core point** if at least **minPts points** are **within distance ε** of it (including *p*). - A point *q* is **directly reachable** from *p* if point *q* is **within distance ε** from core point *p*. Points are only said to be directly reachable from core points. - A point *q* is **reachable** from *p* if there is a **path** *p*1, ..., *pn* with *p*1 = *p* and *pn* = *q*, where each *pi*+1 is directly reachable from *pi*. Note that this implies that the initial point and all points on the path must be core points, with the possible exception of *q*. - All points **not reachable** from any other point are **outliers** or **noise points**.

Normalization:

If it's not normally distributed

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Standardization:

It's normally distributed

$$Z = \frac{X - \mu}{\sigma}$$

Mutual Information:

$$I(X,Y) = I(Y,X) = H(Y) - H(Y \mid X) = H(X) - H(X \mid Y)$$

$$I(X;Y) = D_{KL}\left(P(X,Y), P(X)P(Y)\right) = \sum_{x \in A_x} \sum_{y \in A_y} P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} = \sum_{x \in A_x} \sum_{y \in A_y} P(x,y) \log_2 \frac{P(y \mid x)}{P(y)}$$

Entre F e C quero apresentar o ranking usando IM: - calcular probabilidades. – calcular IM para (F=S,R=B), (F=S,R=A), (F=N,R=B), (F=N,R=A). – somar esses IMs, o que vai dar IM(F,R). – fazer o mesmo para C. – se I(F,R) maior que I(C,R), então ranking vai ser 1º:F, 2º:C

ANOVA:

Determine degrees of freedom

$df_{between} = k - 1$, k=nGroups

$df_{within} = n - k$, n=nPoints

$$SS_{within} = \sum_{i=1}^{k} \sum_{j=1}^{n_i} (x_{ij} - \mu_i)^2$$

$$Var_{within} = \frac{SS_{within}}{df_{within}}$$

$$SS_{Total} = \sum_{i=1}^{k} \sum_{j=1}^{n_i} (x_{ij} - \bar{\mu})^2$$

$$SS_{Total} = SS_{between} + SS_{within}$$

$$Var_{between} = \frac{SS_{between}}{df_{between}}$$

$$F = \frac{Var_{between}}{Var_{within}}$$

Overfitting: occurs when the model learns specific patterns in the training data that do not generalize to new unseen data – learns noise (ajuste exagerado dos dados de treino)

Underfitting: it happens when the model can't capture the underlying patterns in the data, resulting in poor performance on both training and test datasets

Training Set: no partition (training=test)

Train-Test: 70/30

Stratified Hold Out: TT but train and test have balanced classes

Repeated Stratified TT: repeat TT several times with different random samples

Repeated, Stratified, TVT: 40/30/30

Stratified K-Fold Cross Validation: randomly separates the dataset into K groups. Only 1 group for testing. Repeat K times

Repeated Stratified K-Fold: repeats k-fold several times after reshuffling and re-stratifying the data

Nested Repeated Stratified K-Fold: repeated stratified k-fold using TVT

Leave One Out Cross-Validation: each fold contains only 1 sample

ZeroRule (ZeroR):

Algorithm that **classifies all samples** in the test set into one class: the **majority class** in the train set.

Random Classifier:

Randomly assigns a class to each instance

OneRule (OneR):

Learn rules from a **single feature** (not a single rule).

OneRule – Example:

Feature: Age

1. **Discretise**: one approach is to find the more accurate binary split. Ex: 39 or 43 years.
2. **Rules**
   - 39:
     $\leq$ 39: 5 Yes, 1 No: **If Age $\leq$ 39 → yes**
     > 39: 5 No, 4 Yes: **If Age > 39 → no**
   - 43:
     $\leq$ 43: 9 Yes, 3 No: **If Age $\leq$ 43 → yes**
     > 43: 3 No, 0 Yes: **If Age > 43 → no**
3. **Error rate**
   - ER39= $\frac{1+4}{15}$ = **33 %**
   - ER43= $\frac{3+0}{15}$ = **20%**

Feature: Income Range

1. Discrete feature
2. Rules
   - 50-60: 2 Yes, 0 No: **If IR 50-60 → yes**
   - 40-50: 3 No, 1 Yes: **If IR 40-50 → no**
   - 30-40: 4 Yes, 1 No: **If IR 30-40 → yes**
   - 20-30: 2 Yes, 2 No: **If IR 20-30 → no** (because **no** is the less represented class)

3. Error Rate
$ER = \frac{0+1+1+2}{15} = 27\ \%$

Choose the rule with the **smallest ER**

Support Vector Machines:

Find the **hyperplane** that **best** divides a dataset into **two classes.**

Artificial Neural Network:

Input, hidden and output variables are represented by **nodes**. Nodes are organized in **layers**. The weights are the **links** between nodes. The network can be **fully connected** or **sparse**, i.e. not all connections are present

Feed Forward NN: Data flows forward between layers, with no loops or cycles. (**Gradient Descent**: the $\boldsymbol{w}$ update comprise a small step in the direction of negative gradients: at each step, $\boldsymbol{w}$ is moved in the direction of the greatest rate of decrease of E($\boldsymbol{w}$), **Backpropagation**: the derivatives of the errors are propagated back through the network, to adjust $\boldsymbol{w}$)

Convolutional NN: Nearby pixels are more strongly correlated than more distant pixels.

Naive Bayes:

Bayes Theorem:

$$\boxed{P(Y \mid X) = \frac{P(X \mid Y) \cdot P(Y)}{P(X)}}$$

For **classification** problems:

$$\boxed{P(C_k \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid C_k) \cdot P(C_k)}{P(\boldsymbol{x})}}$$

x: features vector
C_k: output classes

Because features are assumed **independent**:

$$\boxed{P(\boldsymbol{x} \mid C_k) \cdot P(C_k) = P(C_k, \boldsymbol{x})}$$

M = P(x)

Probability of class Ck:

$$\boxed{P(C_k \mid \boldsymbol{x}) = \frac{P(\boldsymbol{x} \mid C_k) \cdot P(C_k)}{P(\boldsymbol{x})} = \frac{1}{M} \cdot P(C_k) \prod_{i=1:n} P(x_i|C_k)}$$

Assigned class to a new sample will be:

$$\boxed{\widehat{\boldsymbol{C}} = \underset{\boldsymbol{k}}{\operatorname{argmax}} P(C_k) \prod_{i=1:n} P(x_{new}|C_k)}$$

ID3:

- Select the most important attribute to place at the root node, i.e. attribute with the highest **Information Gain**. - Make a branch for each possible value. - Repeat the process (attribute selection) for each branch using only the instances in the branch. Any attribute can appear at most once along any branch through the tree.

Entropy:

$$\boxed{H(S) = -\sum_{k=1}^{K} p_k \cdot log_2(p_k)}$$

Information Gain: measures how well a given attribute separates the training data according to their target classification

$$\boxed{IG(S, A) = H(S) - \underbrace{\sum_{v \in A} \frac{S_v}{S} \cdot H(S_v)}_{\text{Entropy after partitioning S using attribute A}}}$$

A: attribute with possible values *v*
S_v: *samples with value v for attribute A*

$$\boxed{IG(S, A) = H(S) - H(S|A)}$$

Naïve Bayes example:

- New instance:

Outlook	Temperature	Humidity	Windy	Play
sunny	cool	high	true	?

- P(C_k | Sunny, Cool, High, Windy): $\widehat{C} = \underset{k}{\operatorname{argmax}} P(C_k) \prod_{i=1:n} P(x_{new}|C_k)$

Outlook			Temperature			Humidity			Windy			Play	
sunny	2/9	3/5	hot	2/9	2/5	high	1/3	4/5	false	2/3	2/5	9/14	5/14
overcast	4/9	0/5	mild	4/9	2/5	normal	2/3	1/5	true	1/3	3/5		
rainy	1/3	2/5	cool	1/3	1/5								

$$P(C_{yes}) \prod_{i=1:n} P(x_i|C_{yes}) = \frac{9}{14} \cdot \frac{2}{9} \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{3} = \frac{18}{3402} = 0.0053$$

$$P(C_{no}) \prod_{i=1:n} P(x_i|C_{no}) = \frac{5}{14} \cdot \frac{3}{5} \cdot \frac{1}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} = \frac{180}{8750} = 0.0206$$

$$P(C_{yes}|x) = \frac{0.0053}{0.0053 + 0.0206} = 0.205$$

$$P(C_{no}|x) = \frac{0.0206}{0.0053 + 0.0206} = \mathbf{0.795}$$

ID3 – example:

$$H = -p_{No} \cdot log_2(p_{No}) - p_{Yes} \cdot log_2(p_{Yes})$$

$p_{No} = 5/14 = 0.36$

$p_{Yes} = 9/14 = 0.64$

$H = -0.36 \cdot log_2(0.36) - 0.64 \cdot log_2(0.64)$

$H(S) = 0.94$

$$IG(S, \boldsymbol{Wind}) = H(S) - \sum_{v \in A} \frac{S_v}{S} \cdot H(S_v)$$

Wind=weak → (6 Y, 2 N)
Wind=strong → (3 Y, 3 N)

$H(S_{v=weak}) = -\frac{6}{8} \cdot log_2\left(\frac{6}{8}\right) - \frac{2}{8} \cdot log_2\left(\frac{2}{8}\right) = 0.81$

$H(S_{v=strong}) = -\frac{3}{6} \cdot log_2\left(\frac{3}{6}\right) - \frac{3}{6} \cdot log_2\left(\frac{3}{6}\right) = 1$

$IG(S, \overset{Wind}{\cancel{Hum}}) = 0.94 - \frac{8}{14} \cdot 0.81 - \frac{6}{14} \cdot 1$

$IG(S, Wind) = \mathbf{0.05}$ ***Low IG***

C4.5:

Gain Ratio: incorporates a term sensitive to how broadly and uniformly the attribute split the data

$$\boxed{\boldsymbol{GR(C, A) = \frac{IG(C, A)}{H(A)}}}$$

Example:

$IG(\boldsymbol{S, Wind}) = H(S) - \sum_{v \in A} \frac{S_v}{S} \cdot H(S_v) = \mathbf{0.05}$

Wind=weak → 8
Wind=strong → 6

$H(A = Wind) = -\frac{8}{14} \cdot log_2\left(\frac{8}{14}\right) - \frac{6}{14} \cdot log_2\left(\frac{6}{14}\right) = 0.985$

$GR(S, A) = \frac{IG(\boldsymbol{S, Wind})}{H(A = Wind)} = \frac{0.05}{0.985} \cong \mathbf{0.051}$

Missing values: assign the most common value within the instances of the same class.

Still ID3:

The training procedure looks to minimize the SD of the target on each decision node

Procedure: Compute the SD of the target: $\boldsymbol{SD(T)}$. - Compute the SD for each target and feature:

$\boxed{\boldsymbol{SD(T, x_i)} = \sum_{v \in x_i} p(v)\, SD(T_{x_i=v}) \qquad v \rightarrow \text{values of the feature } x_i}$. - Compute the SD reduction (**SDR**) considering each feature as decision node and select the feature with higher SDR: $\boxed{\boldsymbol{SDR(T, x_i)} = SD(T) - SD(T, x_i)}$

Another example:

Compute the SD(T)= 9.321

2. Compute the SD for each target and feature:

$$\boldsymbol{SD(T, x_i)} = \sum_{v \in x_i} p(v)\, SD(T_{x_i=v})$$

Outlook	P(v)	$SD(T_v)$
Overcast	4/14	3.491
Rainy	5/14	7.782
Sunny	5/14	10.87
SD(T, Outlook) = 7.659		

Temp	P(v)	$SD(T_v)$
Hot	4/14	8.955
Mild	6/14	7.652
Cool	4/14	10.512
SD(T, Temp) = 8.841		

Humidity	P(v)	$SD(T_v)$
High	7/14	9.363
Normal	7/14	8.734
SD(T, Outlook) = 9.048		

Windy	P(v)	$SD(T_v)$
True	6/14	10.593
False	8/14	7.873
SD(T, Outlook) = 9.039		

3. Compute the SDR:

SDR(T, Outlook)= 9.321 – 7.659 = **1.662** **→ Maximum SD reduction: root node**
SDR(T, Temp)= 9.321 – 8.841 = 0.48
SDR(T, Humidity)= 9.321 – 9.048 = 0.273
SDR(T, Windy)= 9.321 – 9.039 = 0.282

```python
# 6.2 - Auxiliary method to calculate distances between sounds for the ReliefF algorithm
def smallerDistance(sound, classIndexes):
    classDistances = []
    for index in classIndexes:
        # Calculate the Euclidean distance between the given sound and the sound at the current index
        distance = np.sqrt(np.sum((sound - featureMatrix[index]) ** 2))
        # Append the distance and the corresponding index as a tuple to the classDistances list
        classDistances.append((distance, index))
    # Sort the classDistances list based on the distance (first element of the tuple)
    classDistances.sort(key=lambda x: x[0])
    nearestIndex = classDistances[0][1]  # Get the index of the nearest sound (smallest distance)
    nearestSound = featureMatrix[nearestIndex]  # Retrieve the nearest sound from the featureMatrix
    return nearestSound
# 6.2 - Method to implement the ReliefF
def reliefF():
    relifFResults = np.zeros(featureMatrix.shape[1])  # Initialize an array to hold the ReliefF results for each feature
    nSamples = 100
    for i in range(nSamples):
        randomIndex = np.random.randint(0, featureMatrix.shape[0])  # Randomly select an index from the featureMatrix
        # Get the sound and its corresponding class for the randomly selected index
        randomSound = featureMatrix[randomIndex]
        randomClass = keyMatrix[randomIndex]
        # Find indexes of sounds that belong to the same class as the random sound
        sameClassIndexes = np.where(keyMatrix == randomClass)[0]
        # Exclude the randomly selected index from the same class indexes
        sameClassIndexes = sameClassIndexes[sameClassIndexes != randomIndex]
        nearestHit = smallerDistance(randomSound, sameClassIndexes)  # Find the nearest sound (hit) in the same class
        nearestMisses = []
        for keyIndex, key in enumerate(keys):
            # Check if the current key index is not the same as the random class
            if keyIndex != randomClass:
                diffClassIndexes = np.where(keyMatrix == keyIndex)[0]  # Find indexes of sounds that belong to a different class
                # Find the nearest sound (miss) from this different class
                nearestMiss = smallerDistance(randomSound, diffClassIndexes)
                nearestMisses.append(nearestMiss)  # Append the nearest miss to the nearestMisses list
        for featureIndex in range(featureMatrix.shape[1]):
            hitDiff = abs(randomSound[featureIndex] - nearestHit[featureIndex])  # Calculate the difference for the hit
            relifFResults[featureIndex] -= hitDiff # Update the ReliefF results for this feature by subtracting the hit difference
            # Loop through each miss and accumulate the differences
            missDiff = 0
            for miss in nearestMisses:
                missDiff += abs(randomSound[featureIndex] - miss[featureIndex])
            missDiff /= (len(keys) - 1)  # Average the miss differences over the number of classes (excluding the hit class)
            # Update the ReliefF results for this feature by adding the average miss difference
            relifFResults[featureIndex] += missDiff
    relifFResults /= nSamples  # Average the ReliefF results over the number of samples
    finalResults = []
    for i, result in enumerate(relifFResults):
        finalResults.append((result, i + 1))
    finalResults.sort(key=lambda x: x[0], reverse = True)  # Sort the final results in descending order based on the result value
    return finalResults
```