Machine Learning for Classification.

A Modern Theory of Detection and Estimation.
Block-2: Detection

Emilio Parrado-Hernández, emilio.parrado@uc3m.es

November 7, 2022

Index

Index

Detection problems defined in terms of data

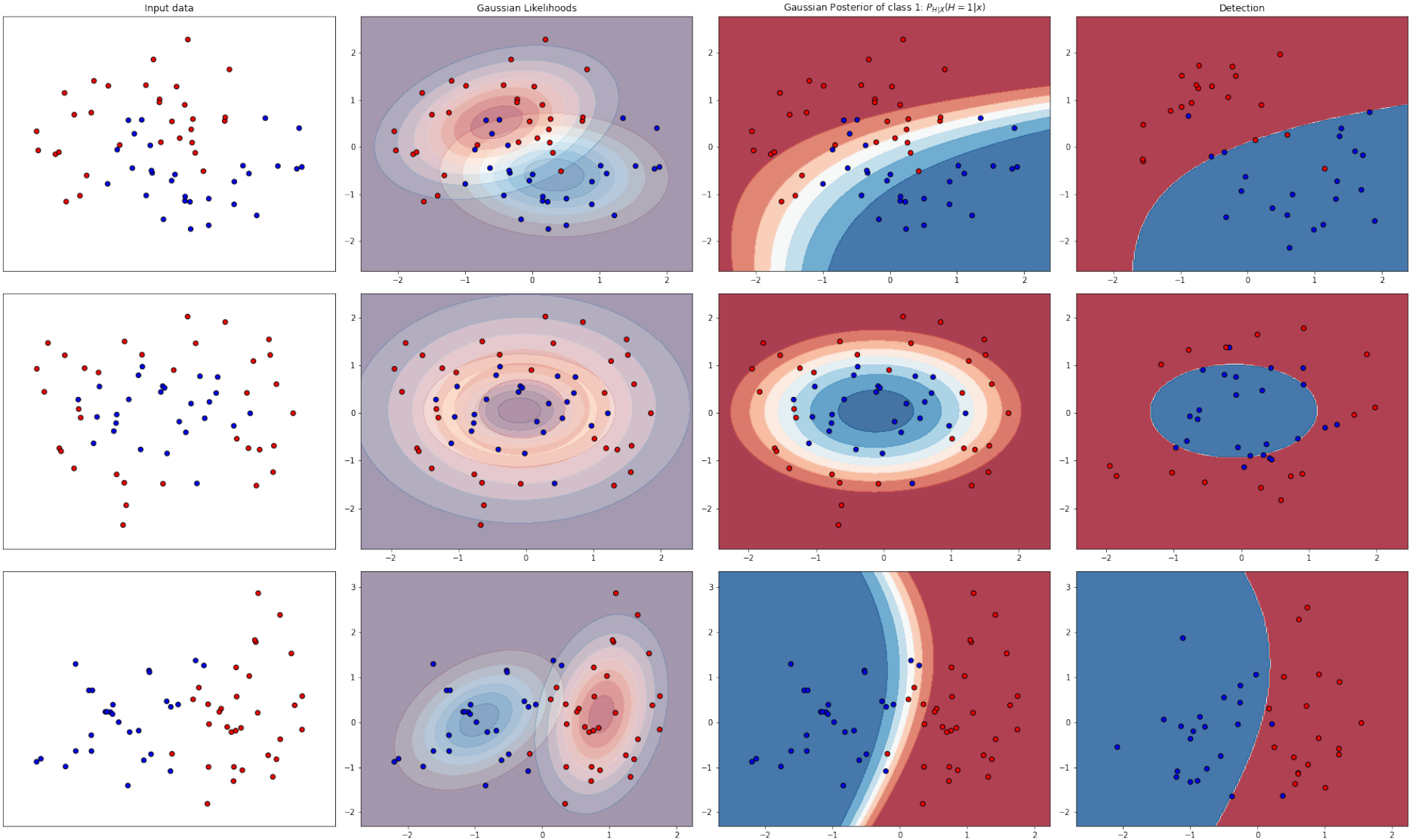
- **Detection**
  - Several **hypotheses** generate observations $\Rightarrow$ each observation comes from one of these hypothesis
  - **Detector** finds out which hypothesis was responsible for each observation
  - Statistical characterization:
    - Prior $P(H = h)$, $h = 1, \ldots, L$
    - Likelihood: $p_{X|H}(\mathbf{x}|h)$, $h = 1, \ldots, L$
    - **costs** c_{dh}, $d, h = 1, \ldots, L$
    - **observations are random** until we get a test observation to form the **posterior** $P_{H|X}(h|\mathbf{x})$, $h = 1, \ldots, L$
- **Classification**
  - **Data collection** of N pairs (observation, target) $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$
  - Classes can be assimilated to hypotheses generating the data; $y_i \in \{1, \ldots, L\}$ is the true class of observation $\mathbf{x}_i$
  - These N data pairs are not random, we **have observed** them
  - **Classifier** finds out the **true class** of every test observation (those test observations are random)

Analytic, semi-analytic and machine methods

Three strategies to solve detection problems. Solving a detection problem means **find a discriminant function** to classify (find true hypothesis for) test observations

- The problem is defined in terms of **Statistics** (prior and likelihoods): Use **Detection Theory** to find the discriminat function
- The problem is defined with a dataset:
  - **semi-analytic methods**: Use the data to **estimate** the priors and the likelihoods and then apply **detection theory** with these *learned* pdfs to get the discriminant function
  - **Machine Learning**: Use the data to **fit** a discriminant function able to produce correct classifications, but without aiming at learning the true joint pdf.

Semi-analytic methods example



Machine Learning

So far we have studied how to construct detectors using a mathematical model of the problem we need to solve: likelihoods, priors, posteriors, costs, etc
The common situation in real life is that problems are defined by sets of examples. In fact, human beings are most of the time learning by examples.

Machine learning pipeline

1. Receive training set: observations and targets (true class of each observation)
2. Choose a family of models
3. Choose hyperparameters: control the optimization of the model
4. Fit the model with the training data (optimization)
5. Receive the test set: observations without targets
6. Predict a target for each test observation with the model
7. Receive the test set true targets and **evaluate** the accuracy of the model

Training set and test set

The **training set** is formed by N pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. The **test set** is formed by N_t pairs $\{(\mathbf{x}_t, y_t)\}_{t=1}^{N_t}$.

The training and test observations are assumed to be drawn iid from a same data probability distribution.

- Observations $\mathbf{x}_i$ live in a d-dimensional space $\mathbf{x}_i \in \mathbb{R}^d$
- targets are discrete and finite. They take the role of hypotheses in analytic detection. In multiclass problems we usually use integers to denote the different classes. In binary classification we usually employ $y_i \in \{-1, 1\}$ (negative and positive classes) or $y_i \in \{0, 1\}$

Data Matrices

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,d} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Each row of X stores an observation

Generalization and overfitting

The goal of machine learning is to learn a model capable of achieving a great **classification accuracy in the test set**.

In general learning to classify correctly the training set is a good strategy to correctly classify those test observations not used for fitting the model

However, the optimization that fits the model can incur in **overfitting**. This phenomenon happens when the performance achieved in the training set is much better than the performance experimented in the test set.

This generalization with the test set is related to the concept of expected risk of a classifier: The expected error rate in a test observation is the **true risk** of a machine learning classifier. In machine learning the average error rate in the training set is called **empirical risk**. Notice the empirical risk can always be measured, but in most of the cases the true risk can only be estimated.

Along the course we will present several ways to estimate the true risk, like using a **separate test dataset** or **crossvalidation**

Index

Parametric and non-parametric methods

The core of the machine learning pipeline is to fit the parameters of the model. This is equivalent to coming up with the **discriminant function** $g(\mathbf{x})$.

Typically a model is a software template with **free parameters**. **Learning** involves an optimization that uses the training data to set these free parameters to values that endow the model with the best possible generalization capability.
Models can be classified into two great groups according to the number of parameters to fit:

- **Parametric models**: The number of parameters is fixed before obtaining the training data. For instance a linear classifier with $4-$dimensional data needs to fix 5 parameters independently of the size of the training set
- **Non-parametric models**: The number of parameters is a function of the training dataset size. Some examples at the end of the notes

Index

Index

kNN outline

- Non parametric model. There is one **hyperparameter**, k, number of neighbors.
- Underlying assumption: If two observations $\mathbf{x}_i$ and $\mathbf{x}_j$ are close enough, it is reasonable to think they will belong to the same class.
- Locality classification: Each test observation is classified exclusively according to its neighbors: closest observations in the training set.
- The value of k determines the size of the neighborhood, and controls the locality of the classifier:
  - $k = 1$: each test observation is assigned to the class of its closest observation in the training set.
  - $k = N$: all the test observations are assigned to the **majority class**
  - intermediate values of k modulate the sizes of the neighborhood of each observation.
- The value of k has to be fixed with **prior domain knowledge**. The range of values of k that works well will be different for each problem. Notice the density of observations can vary across the input space. That is, for a test observation in a densely populated area of the input space its k nearest neighbors can fit in a ball of a smaller radius compared to the case of a test observation in a scarcely populated area.

kNN discriminant function

The discriminant function for kNN is constructed in an algorithmic way. For every observation $\mathbf{x}_t$ that you need to classify with kNN follow these steps:

1. Compute the **distance** between $\mathbf{x}_t$ and all the observations in the training set.
2. Sort these distances in ascending order
3. Find the training observations that occupy the first k positions in the ranking of distances to $\mathbf{x}_t$. Let's denote by $\mathcal{I}_k = \{i_1, i_2, \ldots, i_k\}$ the indices of these training observations in the training set.
4. Retrieve the true targets of the kNN: $\{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$
5. $\hat{y}_t = \text{mode}\{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$

The last step can be modified replacing the mode by a voting, where each neighbor $\mathbf{x}_{i_j}$ votes for its class with its vote inversely proportional to the distance between $\mathbf{x}_{i_j}$ and $\mathbf{x}_t$.

kNN model and posterior pdf

Uniform weighting of votes

Local approximation to the posterior pdf of each class

$$P_{H|X}(H = h|\mathbf{x}_t) \approx \frac{N_h}{k}$$

where N_h is the number of training observations of class h in the neighborhood of $\mathbf{x}_t$

Distance weighting of votes

Local approximation to the posterior pdf of each class

$$P_{H|X}(H = h|\mathbf{x}_t) = \frac{p_{X|H}(\mathbf{x}|H = h)P_H(h)}{p_X(\mathbf{x})}$$

$$p_{X|H}(\mathbf{x}|H = h)P_H(h) \approx \sum_{\mathbf{x}_i \in n_k(\mathbf{x}_t), y_i \in h} \frac{1}{d(\mathbf{x}, \mathbf{x}_i)}$$

where $d(\mathbf{x}, \mathbf{x}_i)$ is the distance between the test observation and each of the training observations in the neighborhood of $\mathbf{x}$

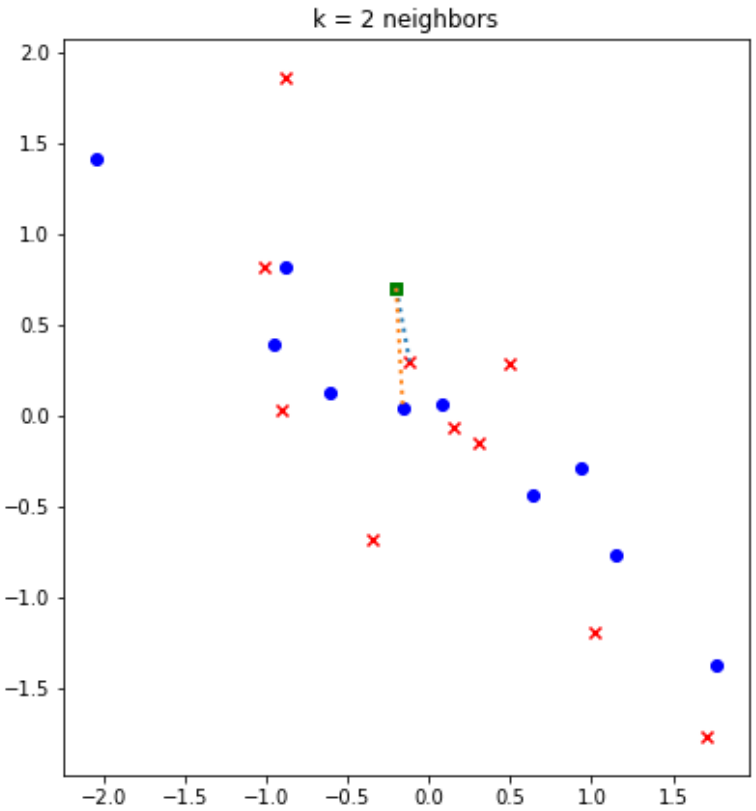
Learning the kNN model

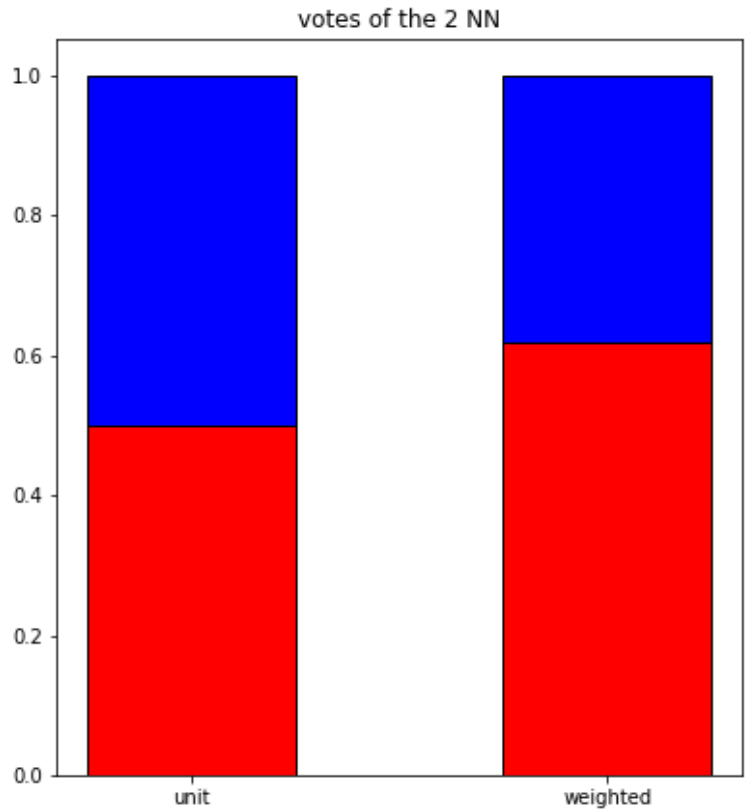
- In fact there is no such a learning process as with other machine learning models, where one need to find values for free parameters running an optimization with the training set observations. kNN just needs to setup an indexed **table that stores the training observations** and their **targets**.
- Choose a **distance** or **metric** to compare the observations, typically the **Euclidean distance** aka L_2 norm:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{j=1}^{d} (x_{1,j} - x_{2,j})^2} = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^\top (\mathbf{x}_1 - \mathbf{x}_2)}$$

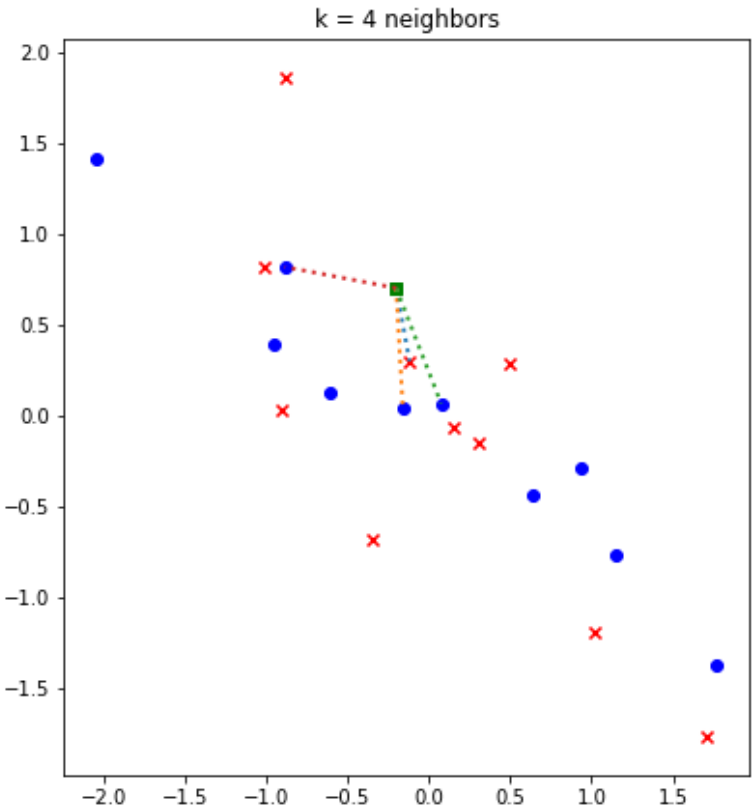
  in python `d_x1_x2 = numpy.linalg.norm(x1-x2)`
- Fix k (number of neighbors)
- Fix a voting strategy to determine the majority class from the k neighbors:
  - Each neighbor contributes a unit vote to its corresponding class (same as computing the mode of the targets)
  - Each neighbor vote is **weighted inversely proportional** to its distance to $\mathbf{x}_t$.

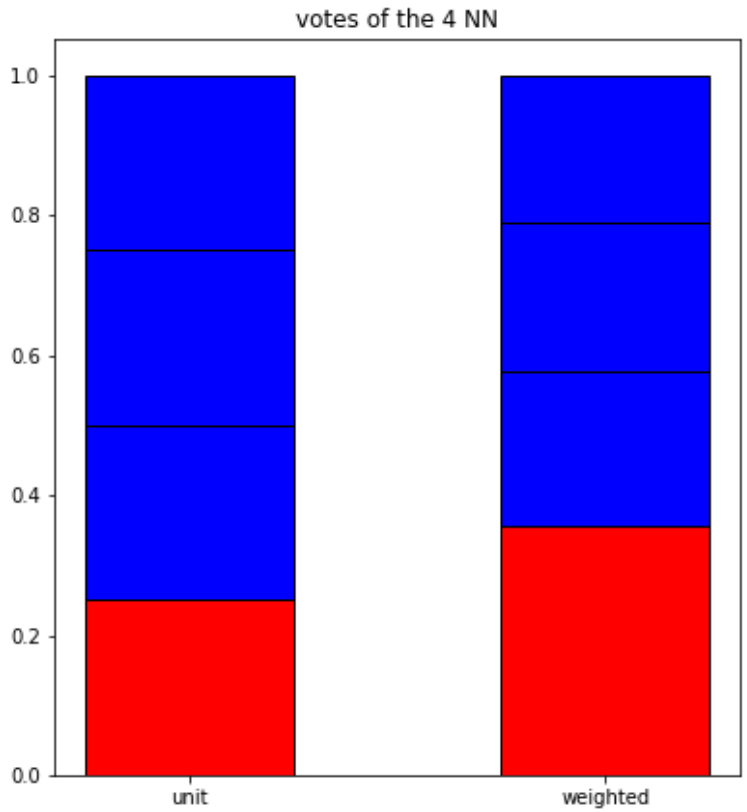
Example


k = 2 neighbors
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

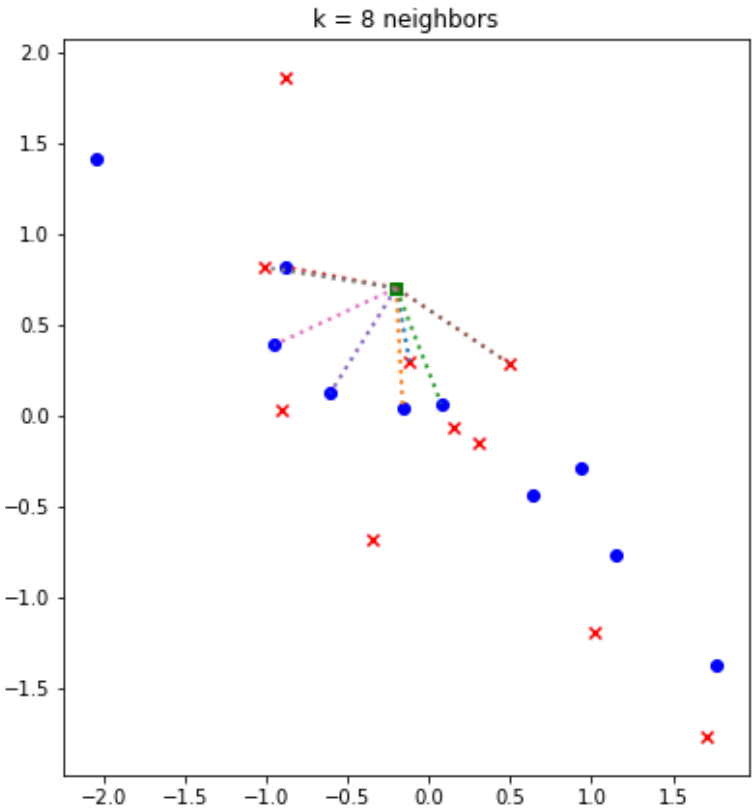

votes of the 2 NN
1.0
0.8
0.6
0.4
0.2
0.0
unit
weighted

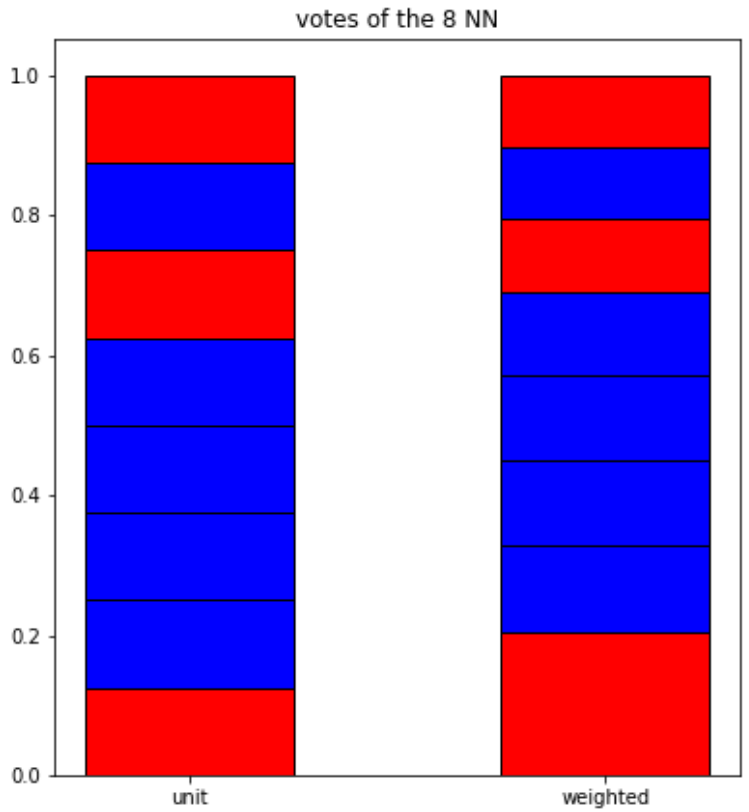
Example


k = 4 neighbors
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


votes of the 4 NN
1.0
0.8
0.6
0.4
0.2
0.0
unit
weighted

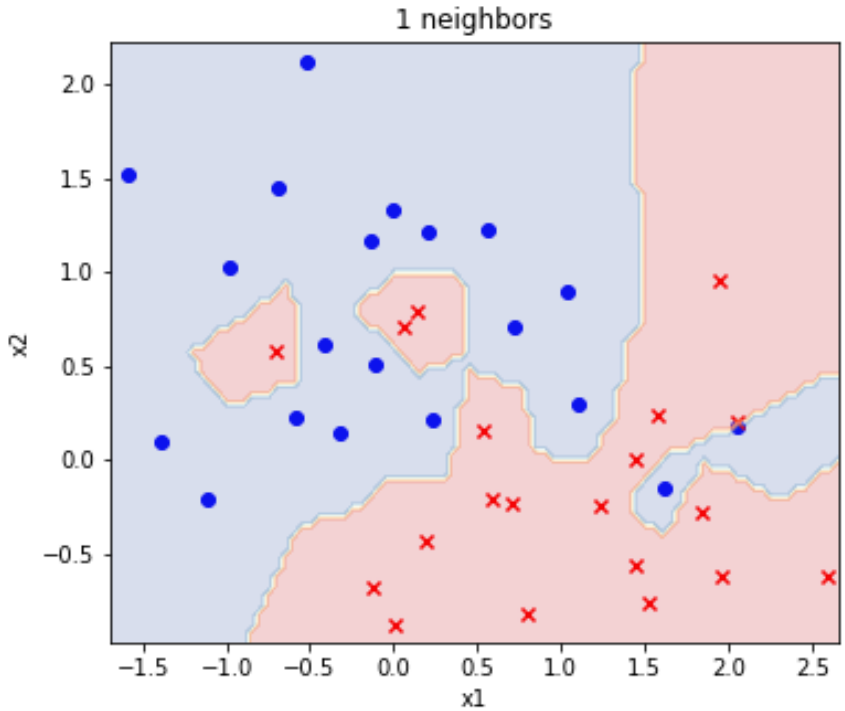
Example

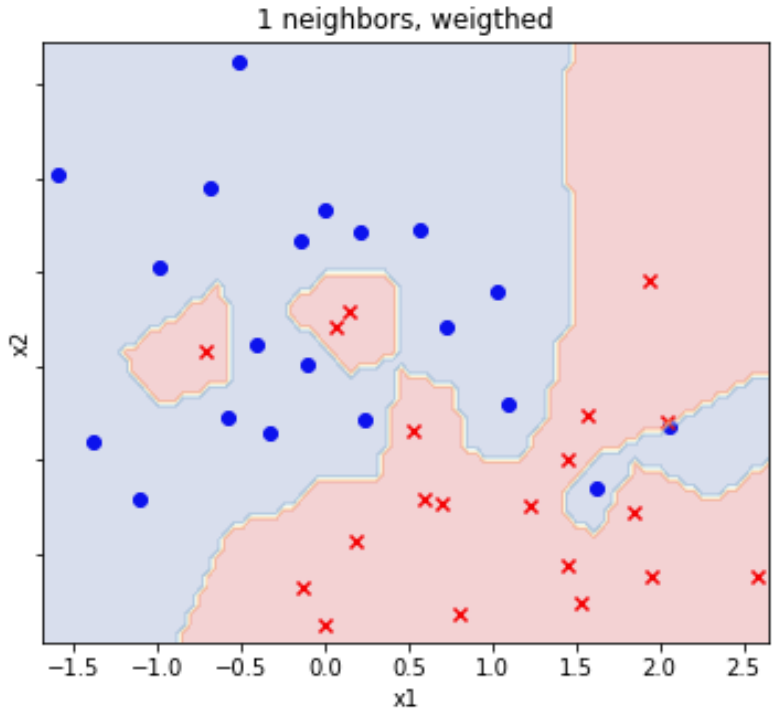

k = 8 neighbors
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


votes of the 8 NN
1.0
0.8
0.6
0.4
0.2
0.0
unit
weighted

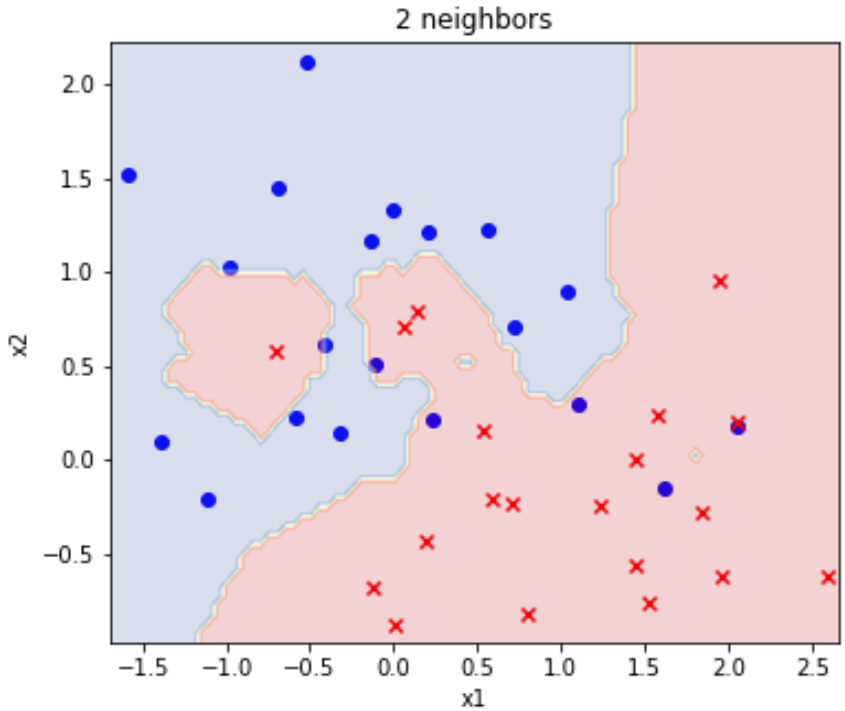
Index

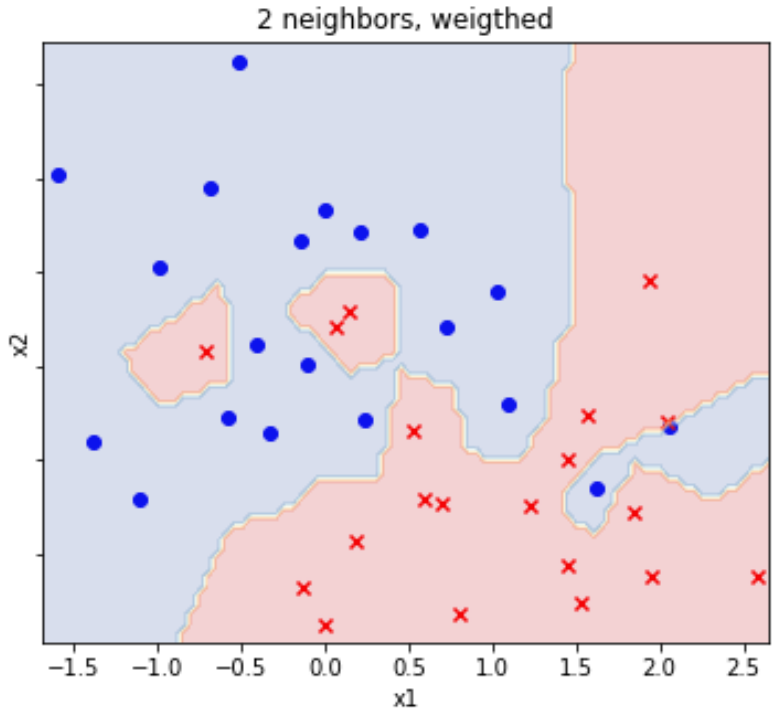
Classification regions



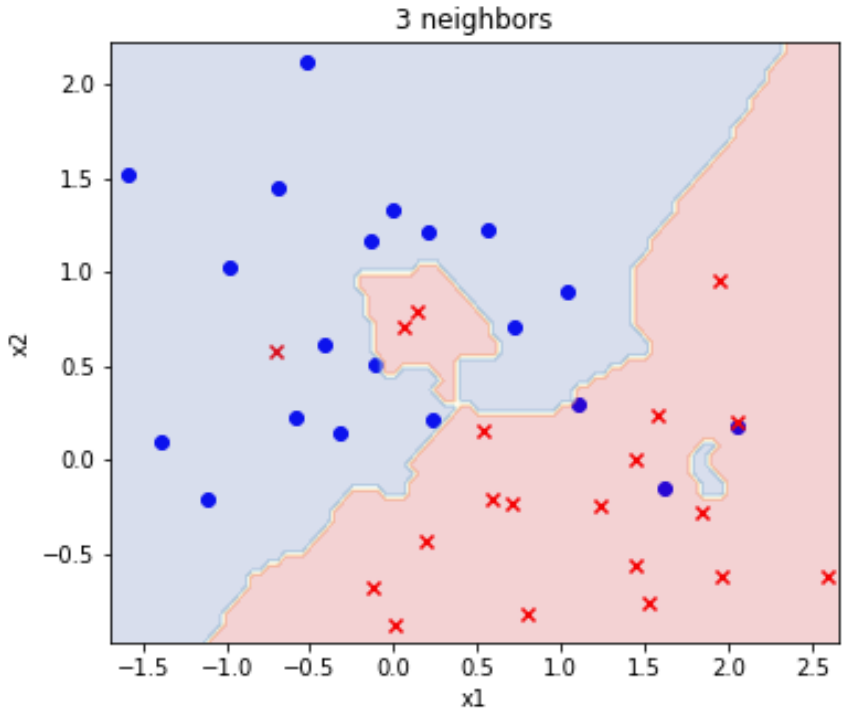


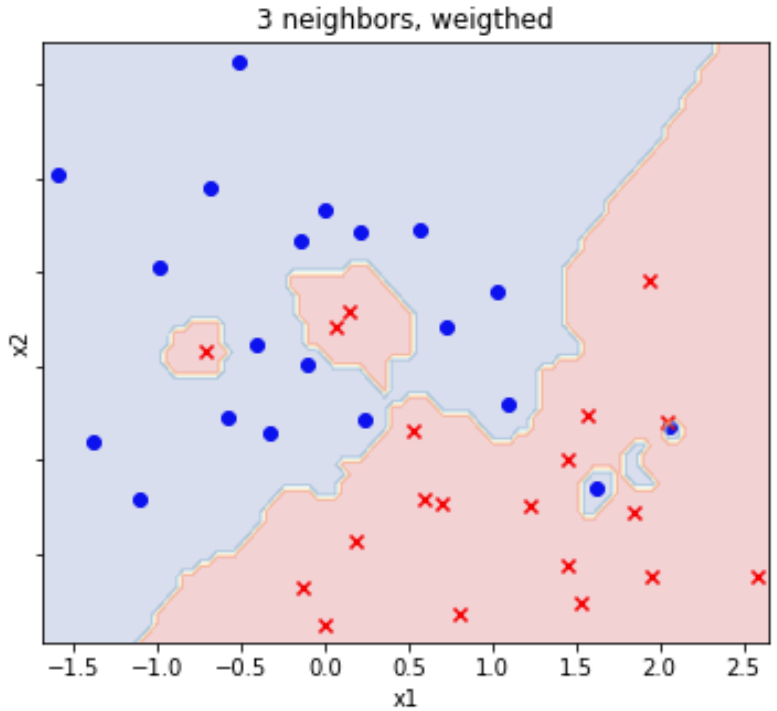
Classification regions



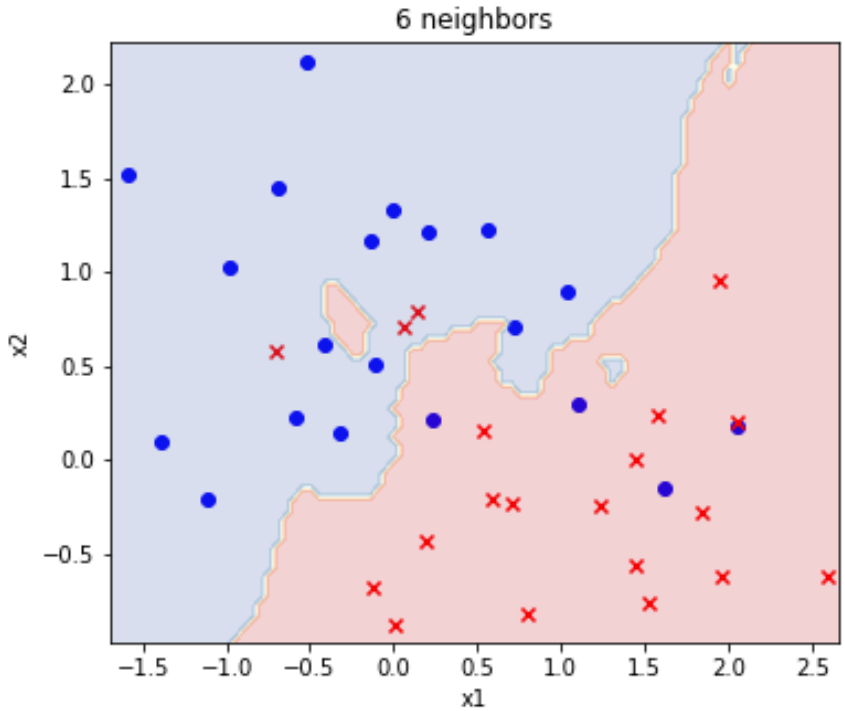


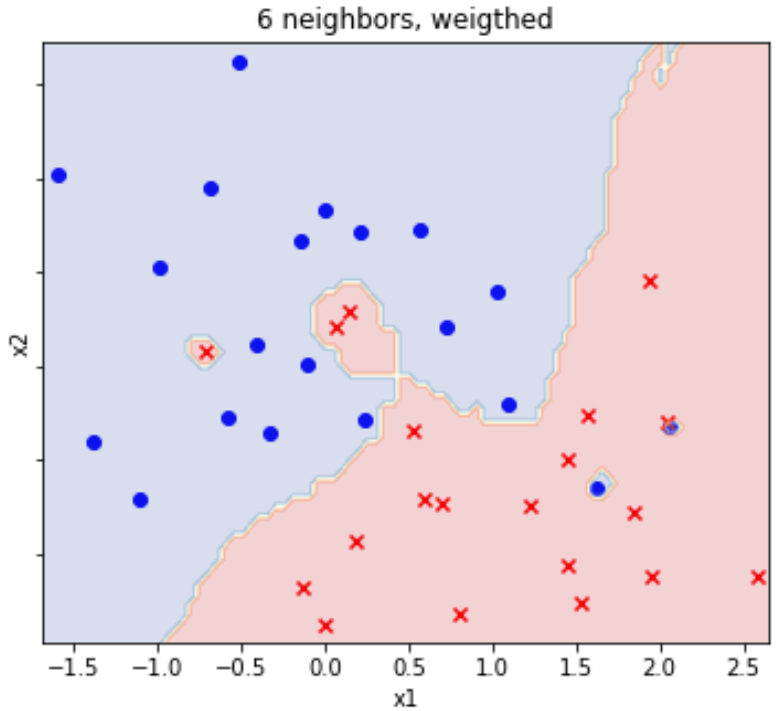
Classification regions





Classification regions





Index

Index

Stump classifier in 1D

A **decision tree** can be regarded as a hierarchical structure constructed through a recursive application of stump classifiers.

Stump classifier

Machine learning version of a binary threshold test on 1D observations

- Binary classifier: two ouput classes
- Defined by a threshold η

$$f_s(x) = x \begin{array}{c} D=1 \\ \gtrless \\ D=0 \end{array} \eta \rightarrow \text{ turns into } \rightarrow f_s(x) = \begin{cases} \text{True} & \text{if } x > \eta \\ \text{False} & \text{if } x \leq \eta \end{cases}$$

- Divides the input space in two, those observations for which the threshold test is True and those for which the threshold test is False.
- To complete the classifier, one needs to assign one of the classes to each outcome of the threshold test. This is made so that the classification accuracy is maximised.

Training a stump classifier in 1D

The algorithm to set the value of the threshold and the output class of each branch of the stump is the following

η, class_true, class_false = stump1d(X, $\mathbf{y}$)

1. Sort all the training observations along the value of the observation x.
2. There is a maximum of N possible ways of partitioning N observations with a single threshold on the value of η:
   - $(x_i + x_{i+1})/2$, $i = 1, \ldots, N-1$ and
   - either $\eta = -\infty$ or $\eta = \infty$, that is, all the observations are in the same class and thus there is no point in dividing the group.
3. Loop for all the N possibles values of η
   1. Assign a class to each side of the stump
      - class_true_n = mode of targets with $x > \eta_n$
      - class_false_n = mode of targets with $x \leq \eta_n$
   2. Evaluate the quality of the stump (for instance with the classification error or any other score)
4. Return the value of η and the corresponding branch class assignment that yielded the best value of the score

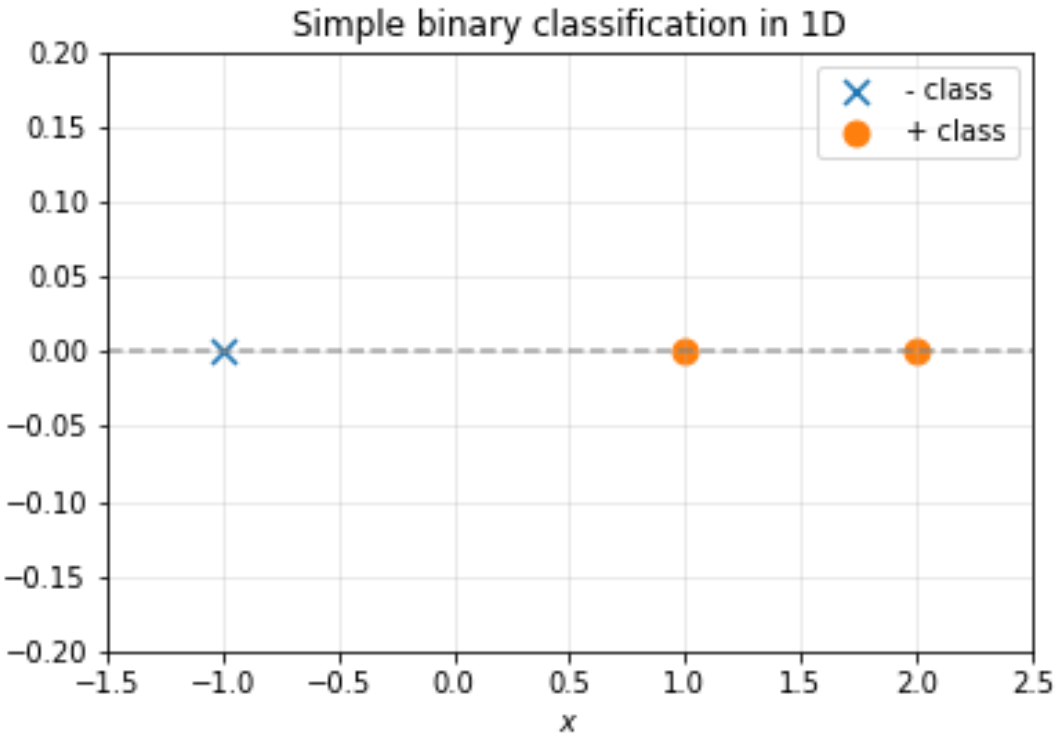
Evaluation of a stump classifier in 1D

$\hat{y}$ = evaluate_stump1d(x, η, class_true, class_false)

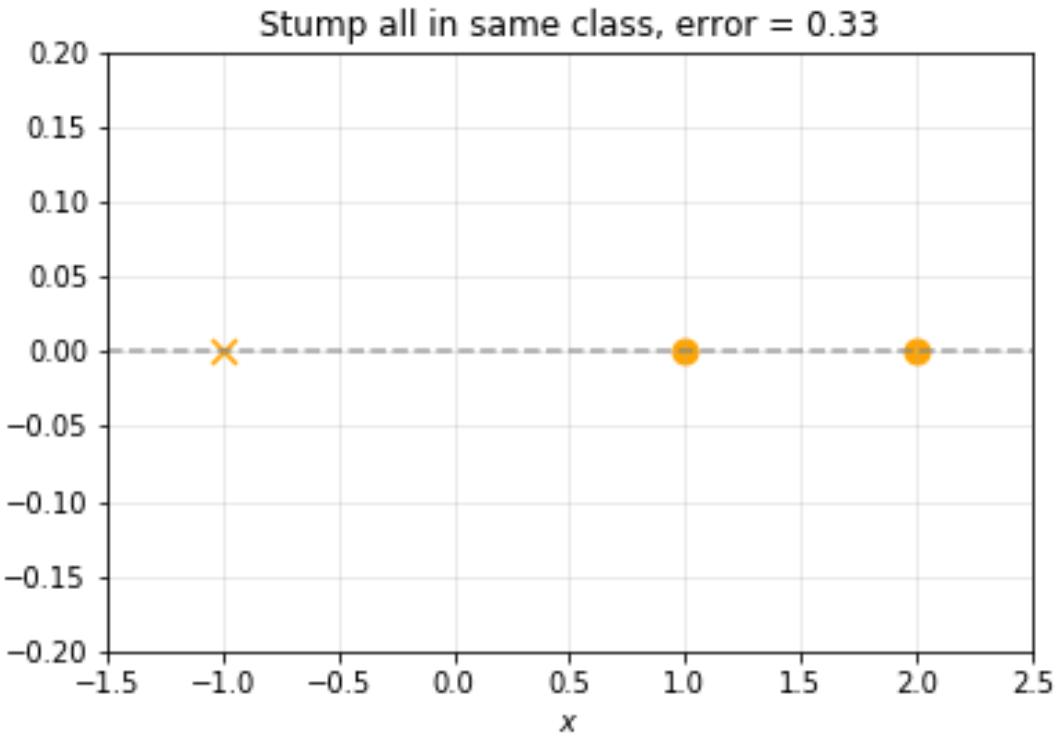
$$f_s(x) = \begin{cases} \text{class\_true} & \text{if } x > \eta \\ \text{class\_false} & \text{if } x \leq \eta \end{cases}$$

Return $\hat{y} = f_s(x)$

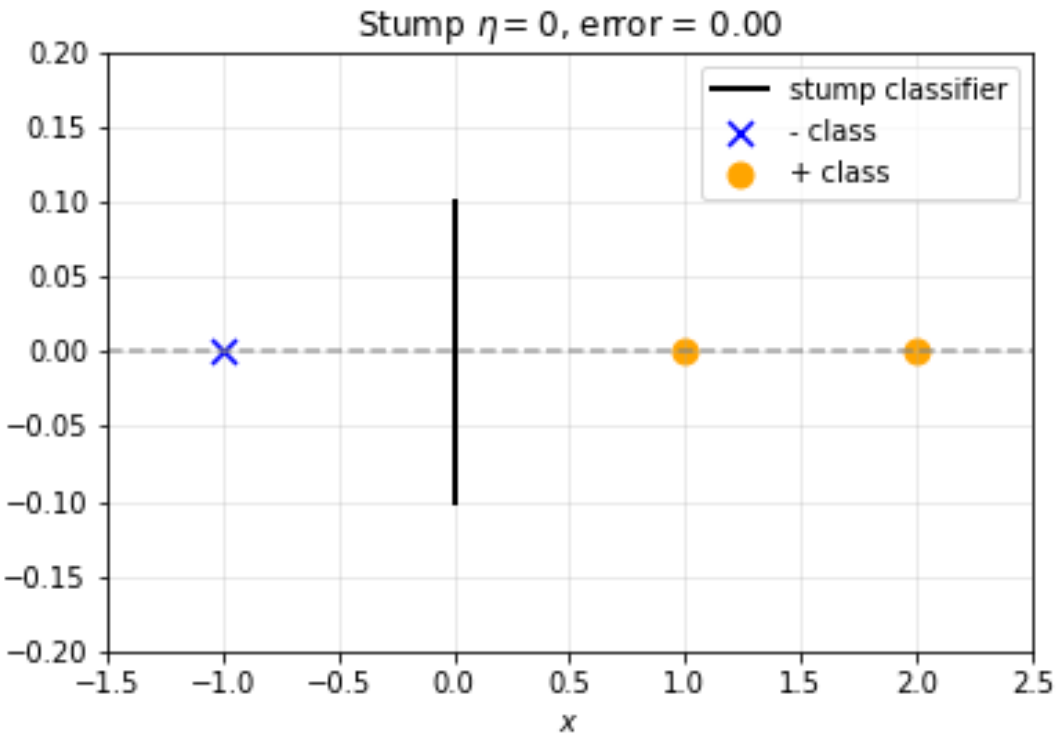
Example


Simple binary classification in 1D
- class
+ class
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

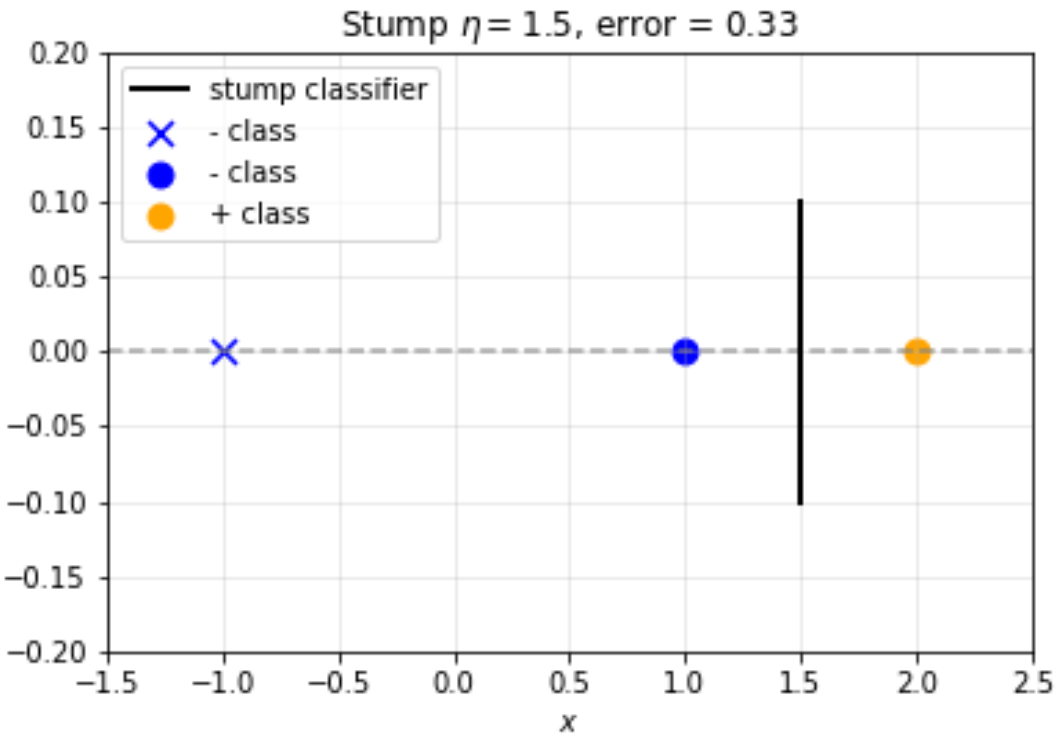
Example


Stump all in same class, error = 0.33
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

Example

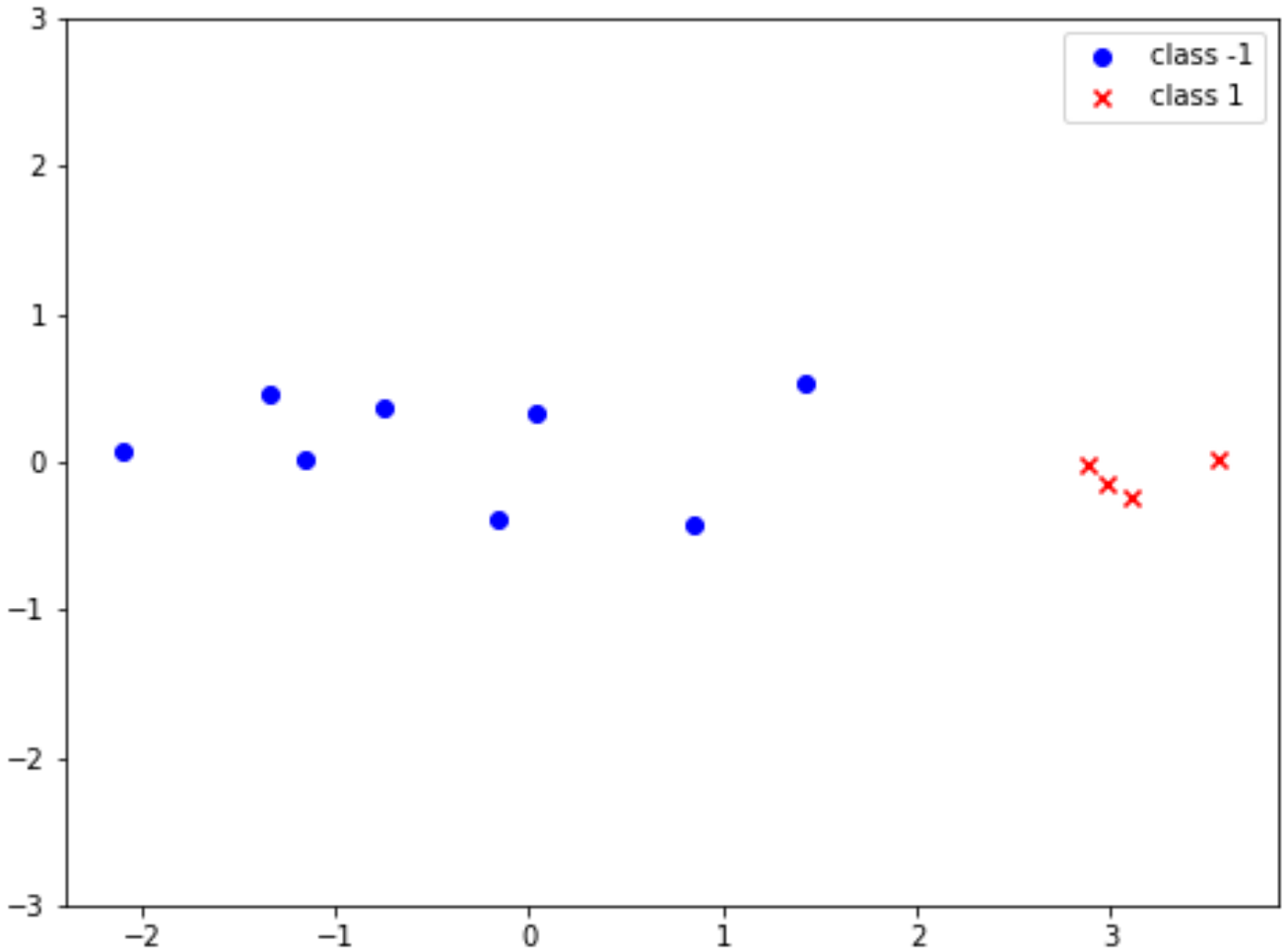

Stump $\eta = 0$, error = 0.00
stump classifier
- class
+ class
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

Example


Stump $\eta = 1.5$, error = 0.33
stump classifier
- class
- class
+ class
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

Stump classifiers in multidimensional problems

The stump classifier algorithm can be extended to observations with more than 1 dimension, $\mathbf{x} \in \mathbb{R}^d$ with $d > 1$.



Training stump classifiers in multidimensional problems

Basically one needs to run the `stump1d` algorithm within a nested loop that iterates over all the d dimensions of the observations.
After processing each dimension $j = 1, \ldots, d$, the `stump1d` will output a best stump for dimension j.
Once all the d dimensions are explored, the algorithm will return the stump corresponding to the dimension that achieved the best value of the score

j, η, class_true, class_false = stump(X, $\mathbf{y}$)

1. Loop for all the dimensions of the observations, in this case, the number of columns of data matrix X.
   `For` $j = 1, \ldots, d$
   - Z is an N dimensional array with the j-th column of X
   - η_j`, class_true_j, class_false_j = stump1d(`Z`, y)`
   - store in a variable s_j the score yielded by (Z,$\mathbf{y}$) in the stump defined with η_j`, class_true_j, class_false_j`
2. Find the best score out of all the iterations
3. Return the dimension j_* that achieved the best score and the corresponding stump defined with η_{j_*} and the corresponding branch class assignment

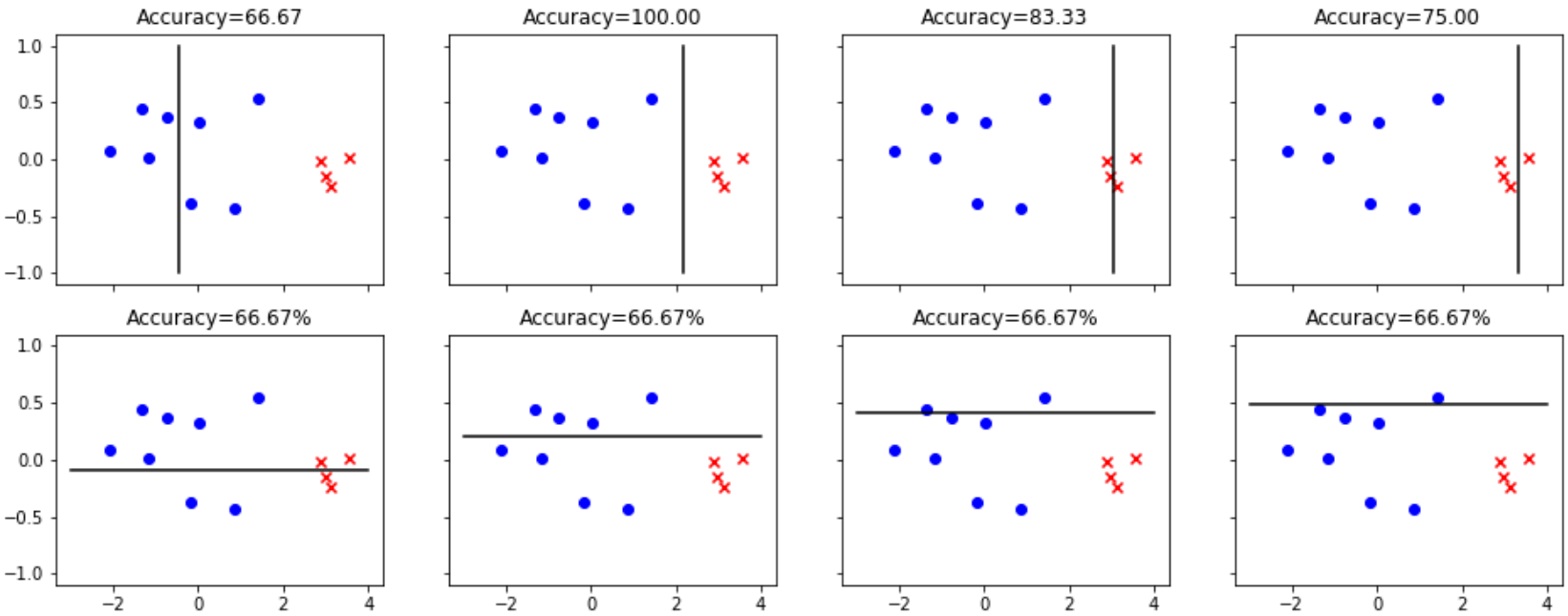
Evaluation of a stump classifier with multidimensional data

$\hat{y}$ = evaluate_stump($\mathbf{x}$, j, η, class_true, class_false)

$$f_s(\mathbf{x}) = \begin{cases} \text{class\_true} & \text{if } x_j > \eta \\ \text{class\_false} & \text{if } x_j \leq \eta \end{cases}$$

Return $\hat{y} = f_s(\mathbf{x})$

Example


Accuracy=66.67
Accuracy=100.00
Accuracy=83.33
Accuracy=75.00
Accuracy=66.67%
Accuracy=66.67%
Accuracy=66.67%
Accuracy=66.67%
1.0
0.5
0.0
−0.5
−1.0
−2
0
2
4

Index

Tree construction

Tree construction

1. Start with a **root node**, s_0, holding the whole training set
2. Divide s_0 into two **branch nodes** (s_1 and s_2) with the algorithm `stump`.
3. Place s_1 and s_2 in a list of branch nodes to explore $\mathcal{B}$. Notice each branch node also receives a subset of the training set: s_1 receives the observations that the stump in s_0 classifies as True while s_2 receives the observations that s_0 classifies as False.
4. Apply the following **recursion** to each node s_k in $\mathcal{B}$. Explore s_k with the algorithm `stump` and the training subset received from its parent:
   - If s_k is **homogeneous** (the stump does not split the observations) s_k becomes a **leaf node**. Then s_k gets a label (the majority class among all the training observations that arrived to it) to classify the test observations that end in s_k.
   - If s_k gets divided by the stump in two branches s_j and s_l:
     - Place s_j and s_l in $\mathcal{B}$
     - Assign to s_j and to s_l the corresponding subsets of the training observations that had arrived to s_k
   - Remove s_k from $\mathcal{B}$
5. Run the recursion until $\mathcal{B}$ is empty

Impurity scores for classification

Each node of the tree is explored with the `stump` algorithm. The stump needs a score to assess the **quality** of the split. Common scores for classification problems are:

- Classification error rate
- **Gini Index**. In a problem with K classes, we define p_k as the fraction of observations in the node that belong to class t_k

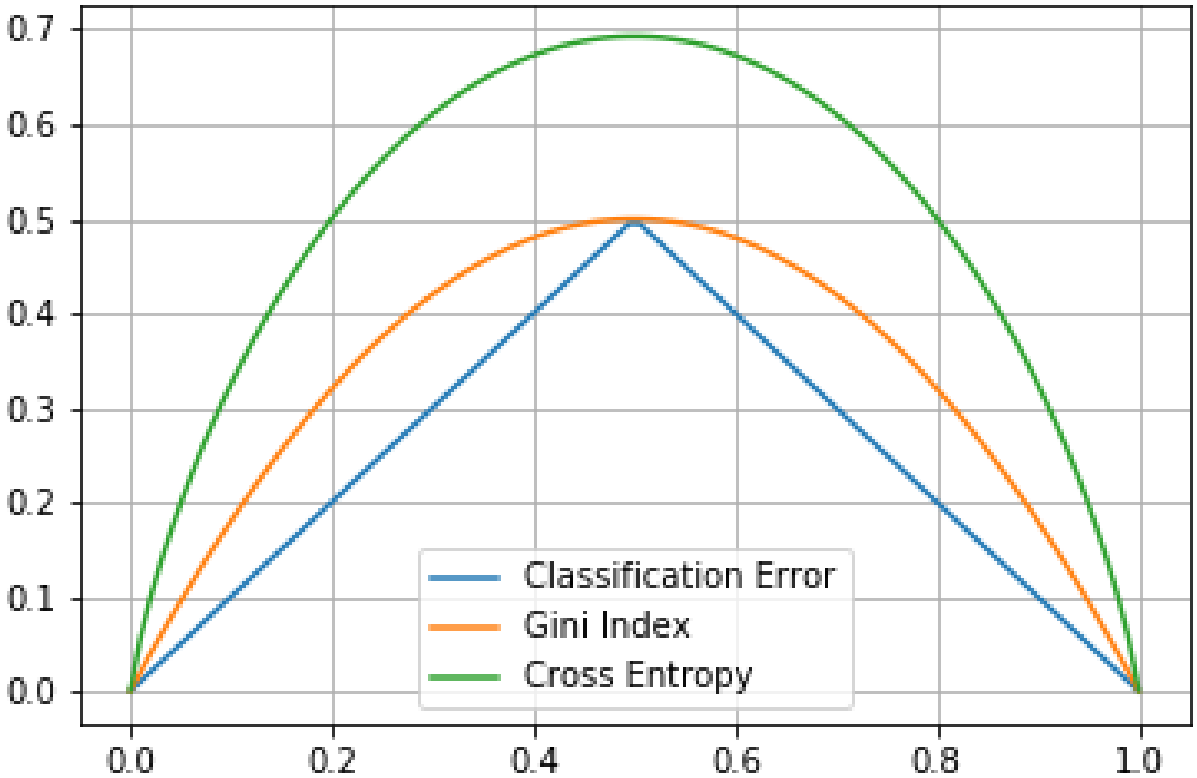
$$\text{Gini} = \sum_{k=1}^{K} p_k(1 - p_k)$$

- **Cross entropy**

$$-\sum_{k=1}^{K} p_k \log p_k$$

Intuitively the Gini index or the cross entropy seem a better choice over the classification error since in cases in which two splits present a same classification error, they favor the split with a purer descendent.

Impurity scores for classification with 2 classes



Tree evaluation

Tree evaluation

1. The evaluation of each observation starts with the stump in the root node s_0. The outcome of this test drives each observation through one of the two nodes (s_1 or s_2) that descend from s_0
2. The observation recursively travels down the hierarchy of the tree, guided by the stumps in the branches
3. The observation finishes its travel when it arrives to a leaf node. It then gets classified with the class of the leaf node.

Index

Controlling the growth of the tree

- The algorithm to grow the tree ends when $\mathcal{B}$ is empty. This means all the training set observations has ended in a leaf node. In a worse case the tree will end up with many leaves classifying a single training observation, what surely will lead to a poor generalization capability.
- Note also that the quality of the stump in the branch nodes depends on the size of the training subset that arrives to them. So as the tree grows deeper the quality of the stumps worsens.
- To overcome this situation one can introduce some external rules to control the growth of the tree. These rules are encoded into **hyperparameters** that are checked along the corresponding steps of the growing tree algorithm.

Hyperparameters for growing trees

- criterion for the quality of the stumps: "gini" or "entropy"
- maximum depth of the tree
- minimum number of samples required to split an internal node. If s_k receives less samples it becomes a leaf
- minimum number of samples required to be at a leaf node. If a branch receives less samples, then the parent becomes a leaf
- number of features to consider when looking for the best split. This means the algorithm `stump` does not explore all the columns of X, just a random subset in each split
- maximum number of leafs. Explore $\mathcal{B}$ in a bf best-first fashion until this limit is reached. Then all the nodes in $\mathcal{B}$ become leaves
- minimum decrease in the impurity score to split a node. If the weighted impurities of the two branches of a node s_k do not improve this limit then s_k becomes a leaf.

Parameters and hyperparameters

During the training of a machine learning model such as a decision tree for classification on needs to find values for two kinds of parameters:

Fittable parameters

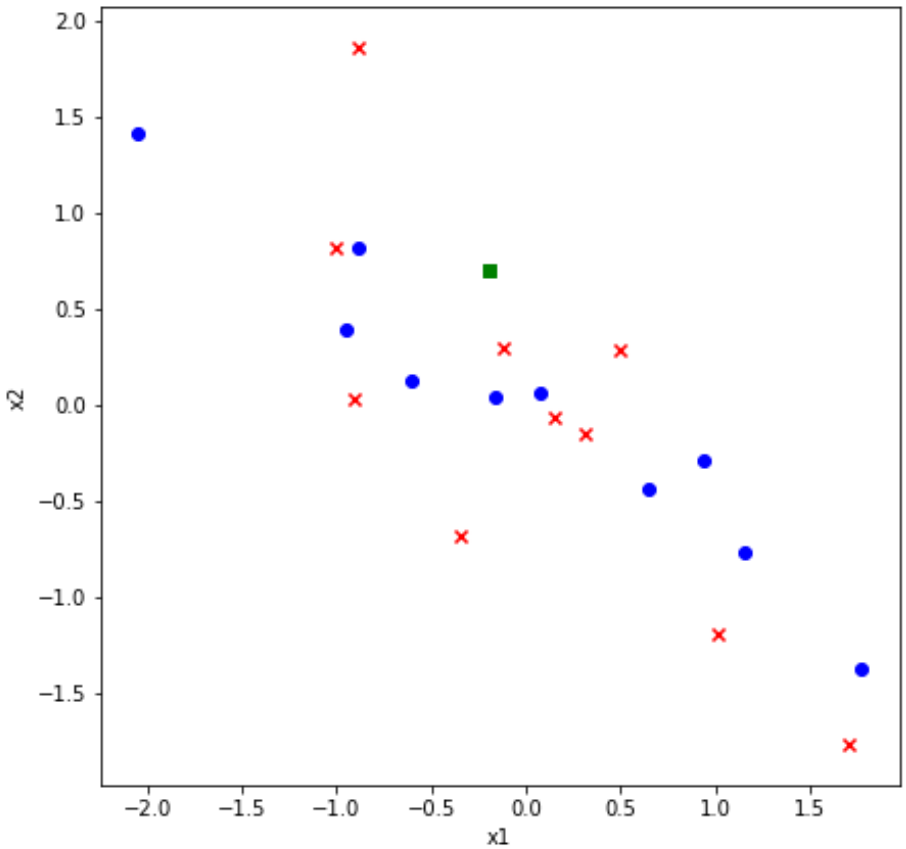
The parameters that define each stump in the tree: the feature j and the threshold η. Their values are adjusted during the optimization that involves the fitting of the model

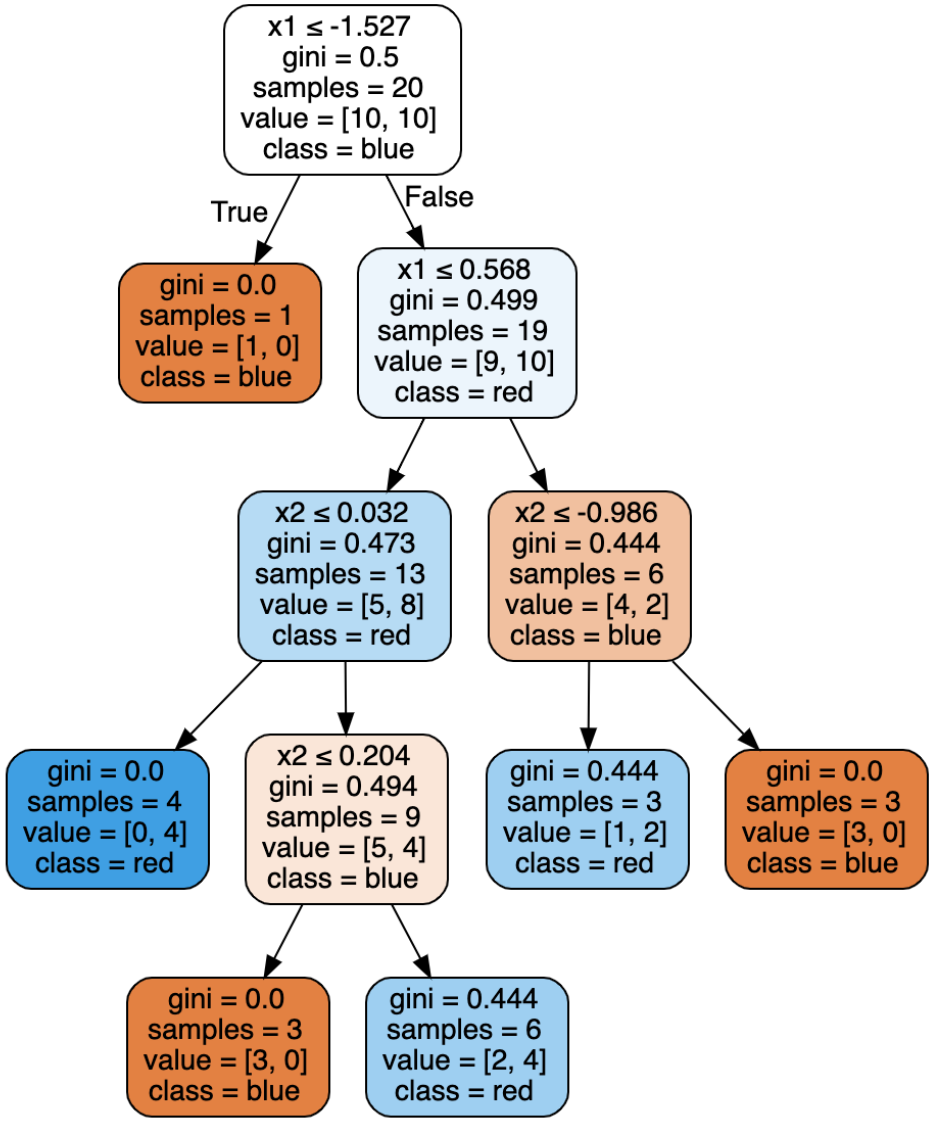
Hyperparameters

They are set during the instantiation of the model. Their role is to control the optimization within the fitting of the model. Their values are adjusted using **prior domain knowledge**. This way we control the generalization capability of the model.
Examples of hyperparameters are k in kNN or the maximum deep of the tree in a decision tree.

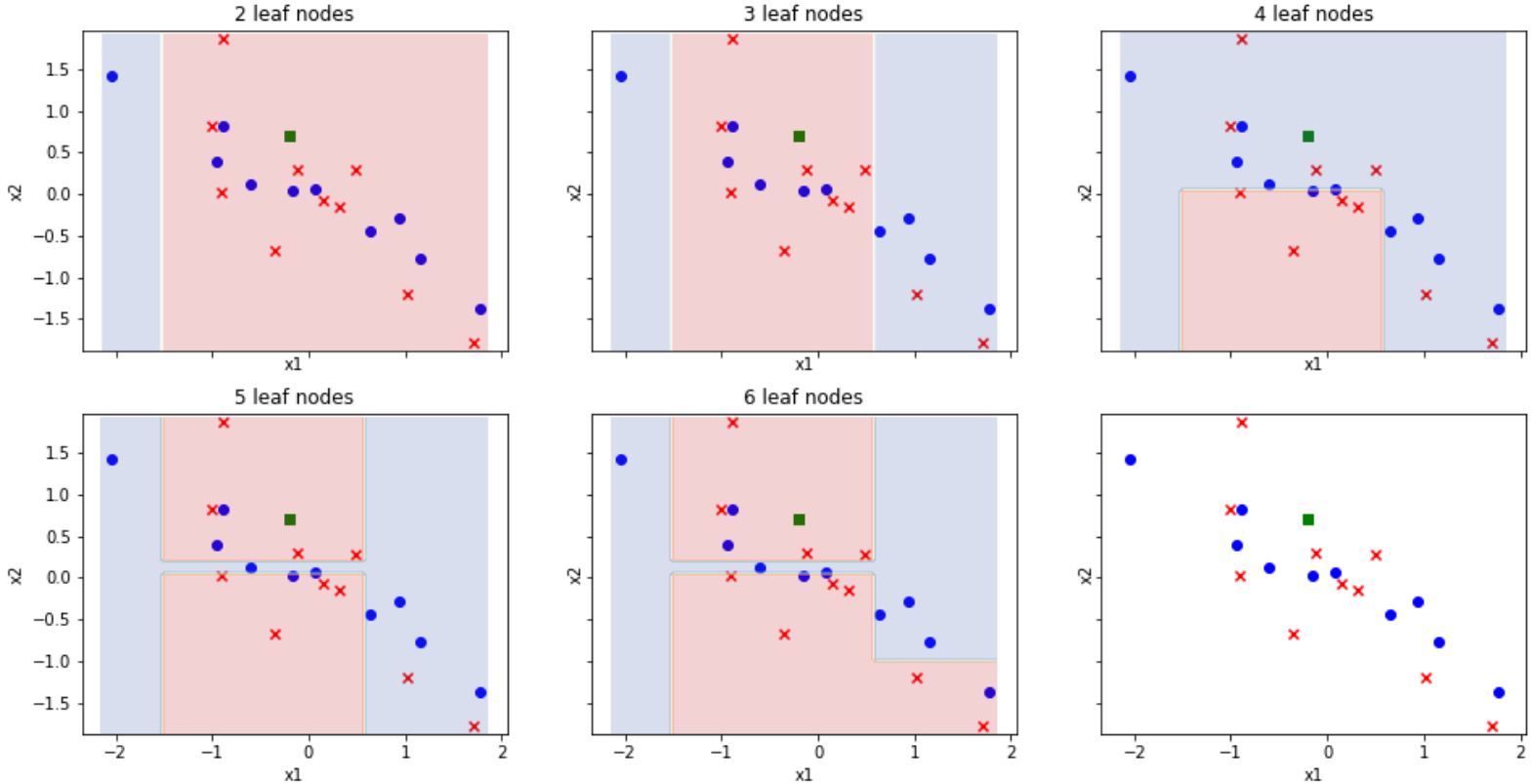
Index

Classification with 2 classes in 2D





Decision trees classification regions



Index

Index

Ensemble methods

A common way of developing robust machine learning models is to learn an **ensemble** of models with a same training set but introducing a small amount of **diversity** in each member of the ensemble:

- **subsampling** the common training set to get a slightly different set for every learner
- using different hyperparameters (maybe chosen at random) for every learner.

Weak Learners

Every member of the ensemble is called **weak learner** since you don't need its using a very accurate learning algorithm. In binary classification there are guarantees that as long as every weak learner is able to perform slightly above 50% accuracy the ensemble can converge to best possible performance ranges.

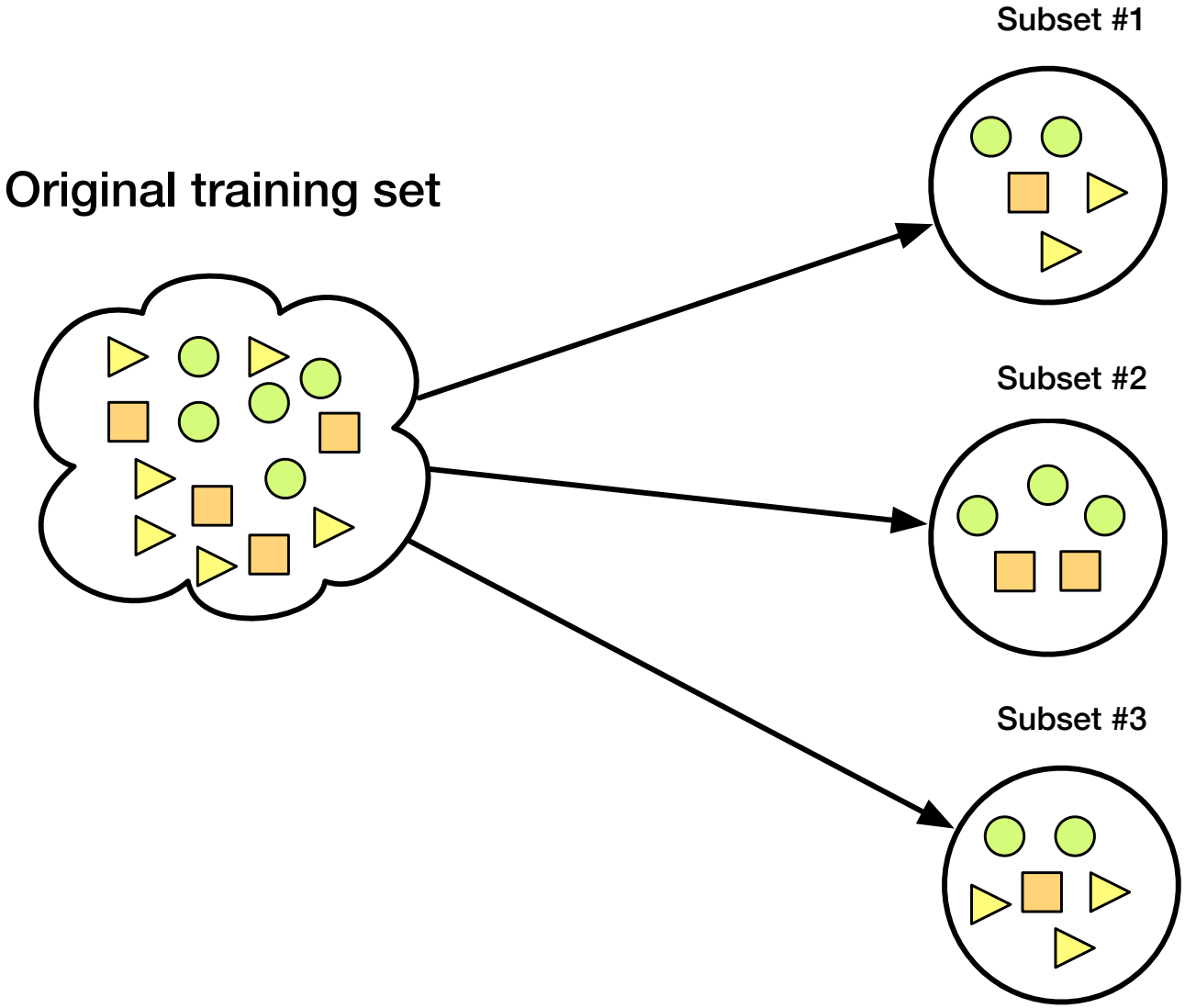
Once the ensemble is built, the prediction for each test observation is made by combining all the weak learners individual predictions

- Taking the average of the individual predictions
- Taking the mode of the individual predictions
- Weighting each individual learner prediction by a coefficient that captures the confidence on its predictions.

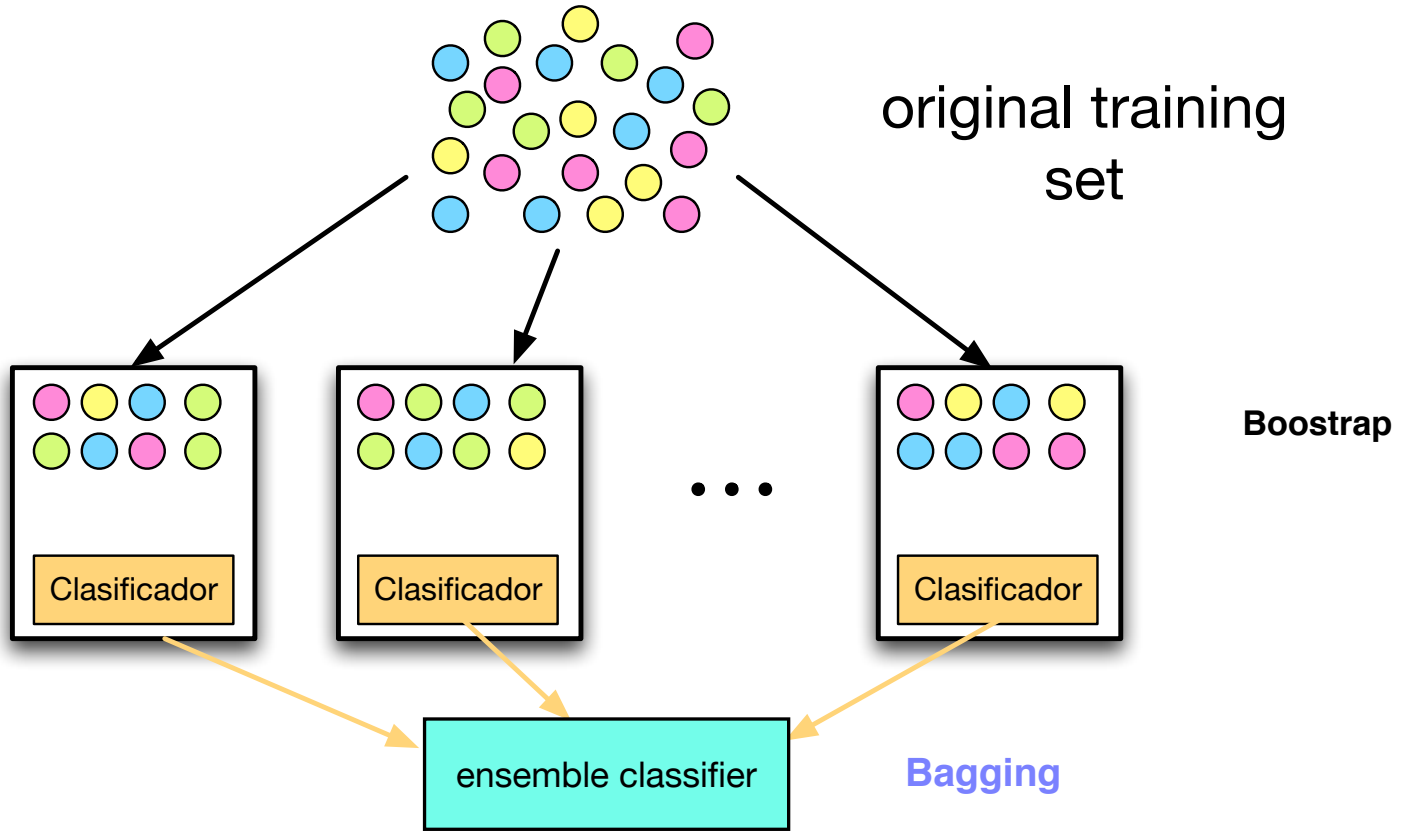
Bagging

- Every weak learner is fitted with a same algorithm but with a different training set that is drawn at random from the original training set (**bootstrap**)
- Weak learners can be fitted in parallel
- The output of the ensemble is the average of the outputs of the weak learners
- Examples: Random Forest

Bootstrap sketch



Bagging sketch



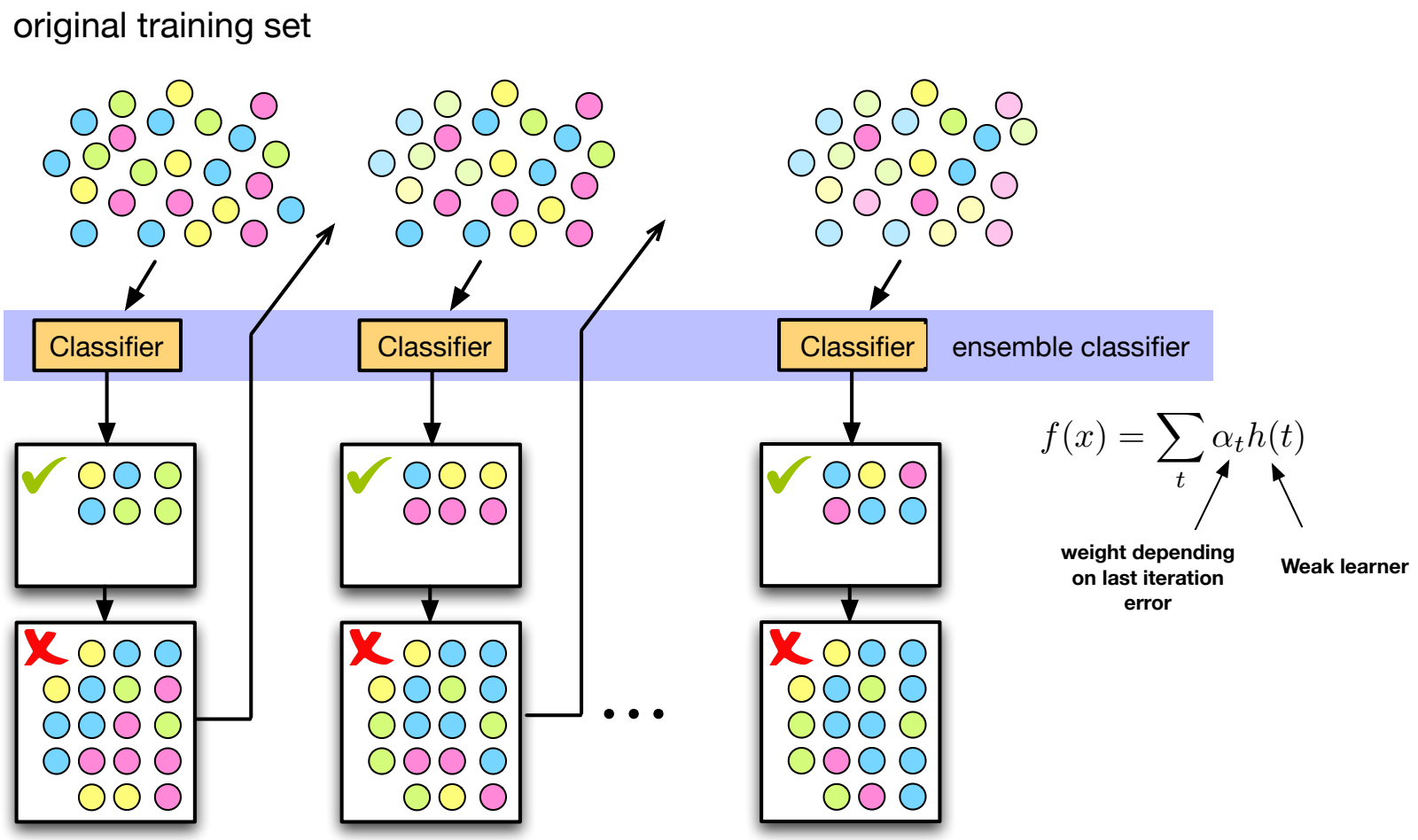
$$f(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

weak learner

Boosting

- Weak learners are trained sequentially using a same algorithm but with slightly different training sets.
- The first weak learner is fitted with the original training set
- The second weak learner receives a training set constructed by resampling the original set in a way that gives more weight to those observations missclassified by the first learner.
- Subsequent weak learners receive a training set constructed by resampling the original set in a way that gives more weight to those observations missclassified by the ensemble formed by the previous weak learners.
- Every weak learner added to the ensemble therefore aims at correcting the errors of its predecessors
- The output of the ensemble is the weighted average of the outputs of the weak learners. The weighting coefficients are proportional of the accuracy of each weak learner.

Boosting sketch



Committee of experts

- This ensemble is not formed by weak learners, but for more sophisticate and diverse algorithms.
- Each algorithm specializes in learning a part of the problem.
- A machine learning model called **gate** is fitted to decide which of the expert would classify each test observation using as information the performance of the different experts on the training set.

Index

Random Forest fitting

Random Forest form an ensemble by growing B decision trees.
The algorithm basically consists in a loop that runs an iteration for each tree in the forest (B iterations). The algorithm receives as input a training set with N observations in d dimensions, with their corresponding targets, and a forest size B.

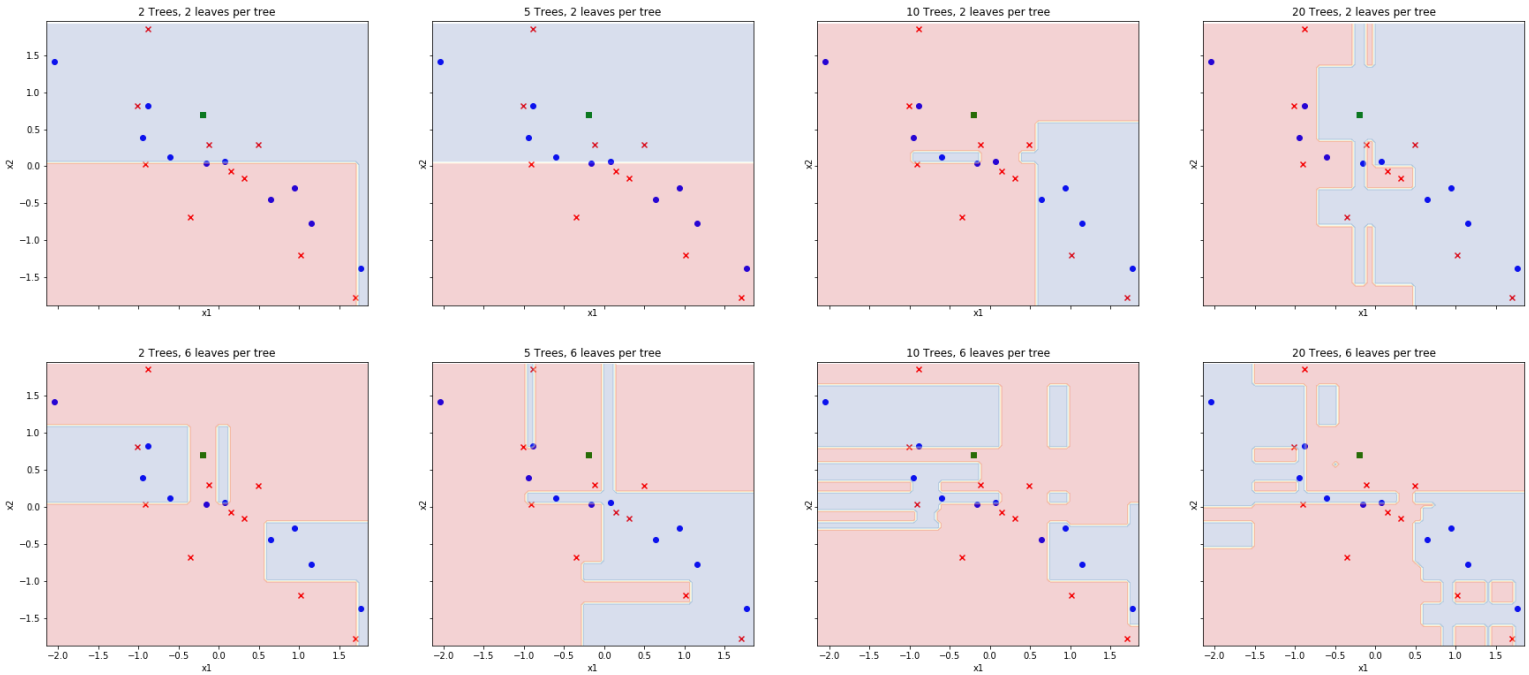
Random Forest Algorithm

For $b = 1, \ldots, B$:

1. Choose at random a subset of N_b observations of the training set
2. Learn tree T_b with this training set of size N_b but with this modification in the standard tree growing algorithm: Before optimizing every stump for every branch node, choose at random m_b features from the available d. This way, each node only access a subset of the input features to optimize its stump, what intoduces further diversity in the growing of the forest
3. Inference. Each tree outputs a target for each test observation. The final target output by the forest is the most voted class

Index

Random Forest for classification regions



Linear Classification.

A Modern Theory of Detection and Estimation.
Block-2: Detection

Emilio Parrado-Hernández, emilio.parrado@uc3m.es

November 14, 2022

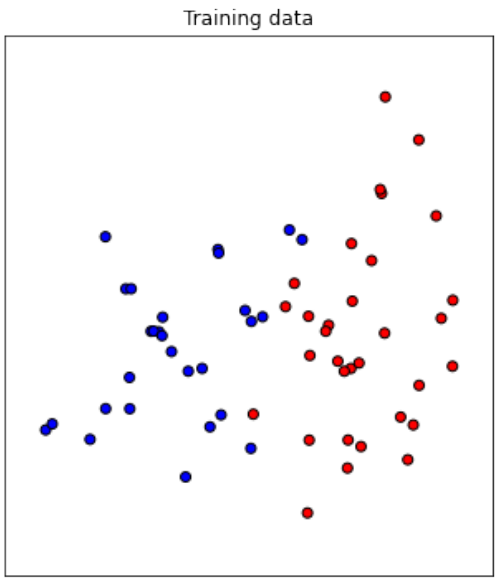
Index

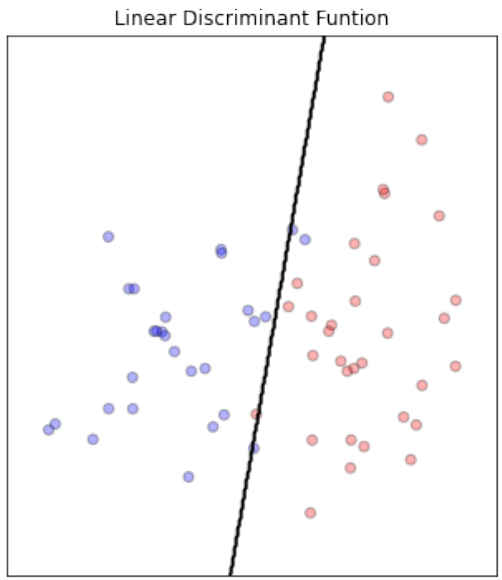
Index

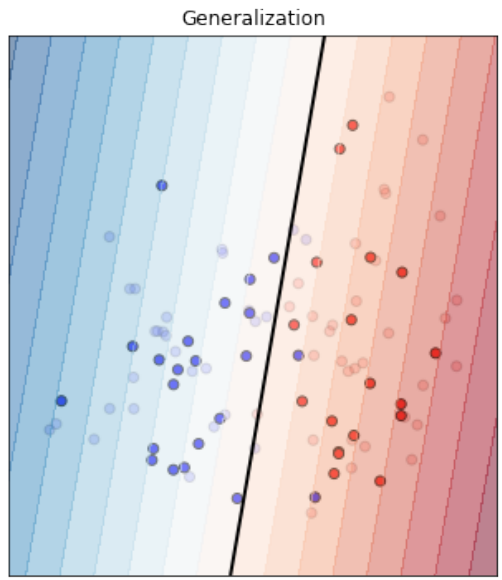
Linear Classifier

Linear discriminant function

$$g(\mathbf{x}) = w_0 + \mathbf{w}^\top \mathbf{x} \begin{array}{c} D=1 \\ \gtrless \\ D=0 \end{array} 0$$







Linear Classifier Motivations

- Problems are generally defined in terms of a **data collection**, not with the true likelihood or the true priors of the hypothesis.
- Sometimes we have the true pdfs but they become **untractable** or very **difficult to handle**
- **Advantages of Linear Classification**
  - Easy to implement
  - Fast operation
  - Optimal when the likelihoods are Gaussian with identical covariance matrices (very common situation in real life applications)

Compact notation

Define targets

$$y_i = \begin{cases} +1 & \text{if } \mathbf{x}_i \in H_1 \\ -1 & \text{if } \mathbf{x}_i \in H_0 \end{cases}$$

Linear discriminant function

$$g(\mathbf{x}) = w_0 + \mathbf{w}^\top \mathbf{x} \begin{array}{c} D=1 \\ \gtrless \\ D=0 \end{array} 0 \rightarrow \hat{y}_t = \text{sign}\left(g(\mathbf{x})\right) = \text{sign}(w_0 + \mathbf{w}^\top \mathbf{x})$$

Compact notation

$$\mathbf{w}_e = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} \qquad \mathbf{x}_e = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$

Therefore

$$g(\mathbf{x}) = \text{sign}(\mathbf{w}_e^\top \mathbf{x}_e)$$

During the rest of the lecture, unless explicitely stated, $\mathbf{x}$ and $\mathbf{w}$ will refer to the compact notations.

Linear separability

Linear separability

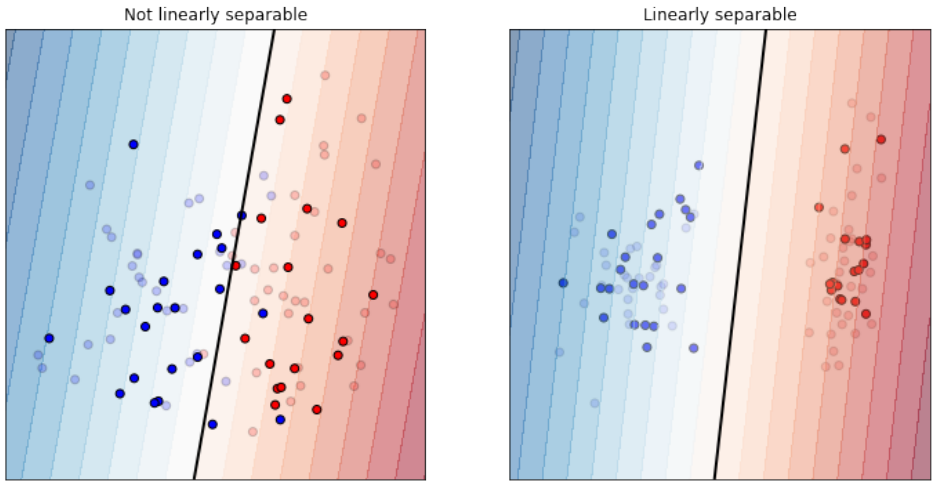
Define the score

$$yg(\mathbf{x}) = y\text{sign}(\mathbf{w}_e^\top \mathbf{x}_e)$$

If $yg(\mathbf{x}) > 0$ for all the observations then the problem is **Linearly separable**

In general we want $yg(\mathbf{x}) > 0$ for a majority of observations.

Any *coherent* set of $d + 1$ observations, where d is the number of variables (dimensionality) of these observations, is linearly separable independently of the values of the targets.



Index

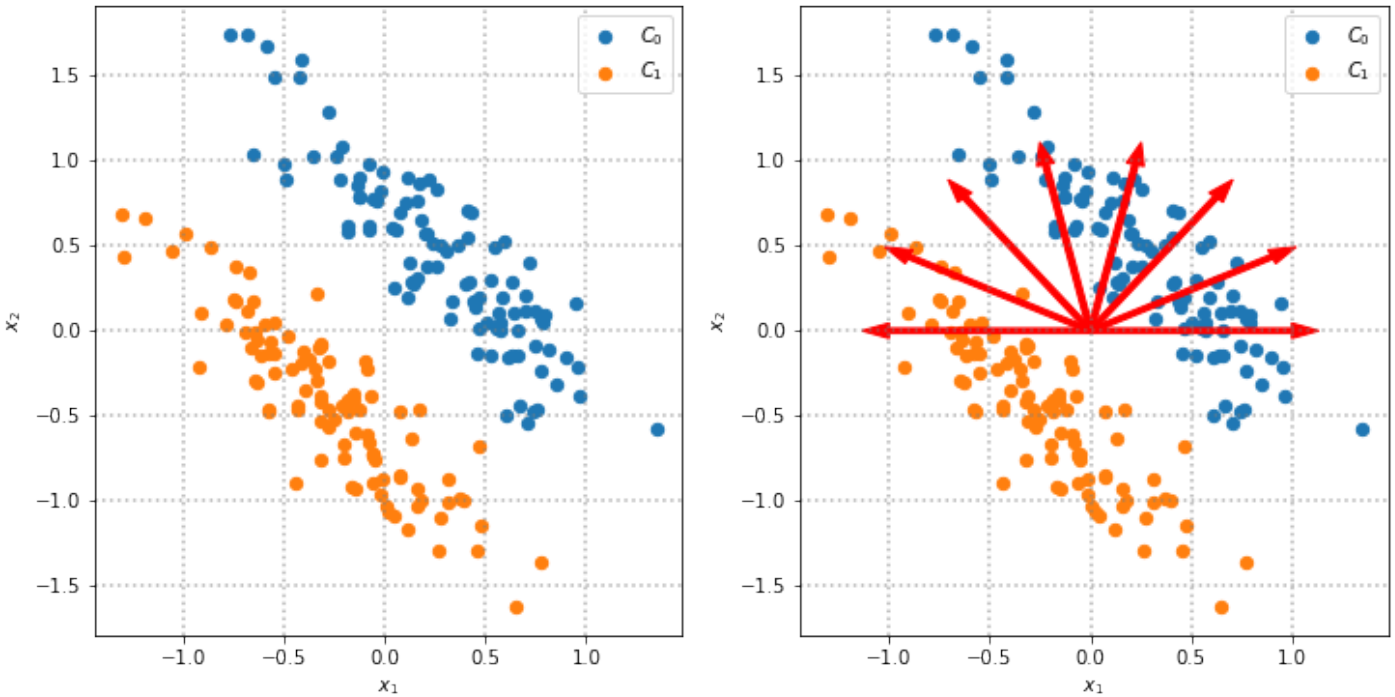
Index

Fisher's criterion of separability

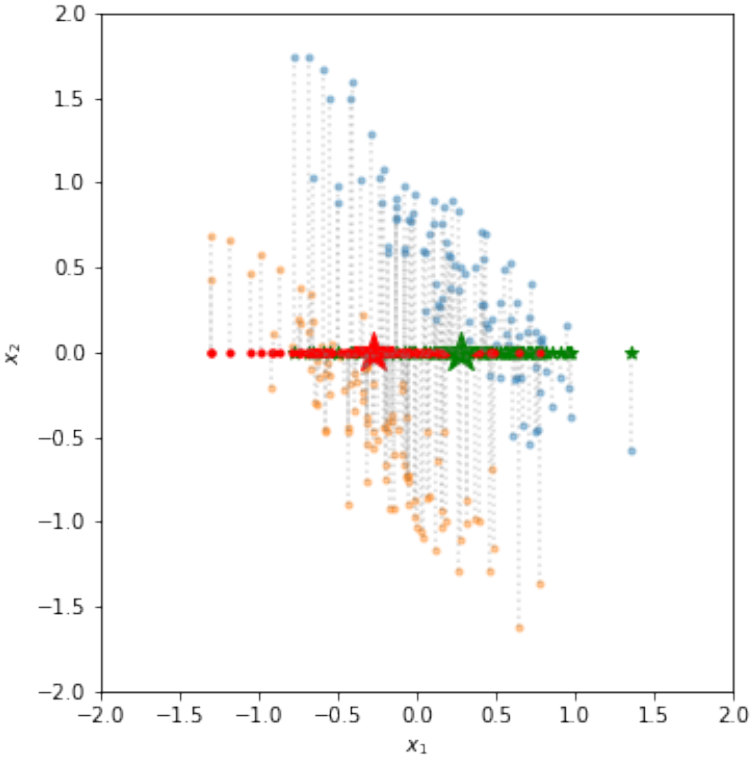
Fisher's criterion

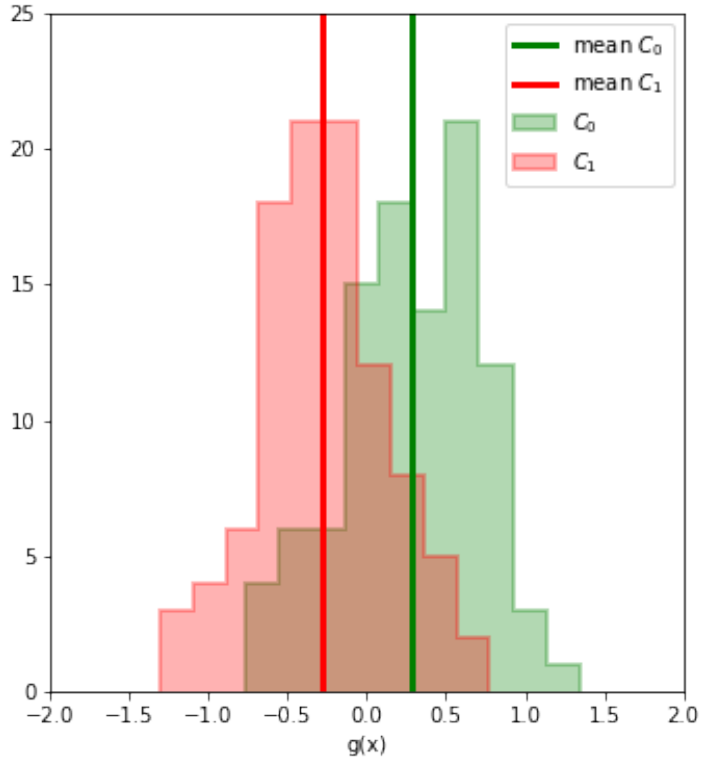
Find the **linear combination** of the input variables that **maximizes the separability** of the two classes

We need to *put into numbers* the criterion of **separability** and then maximize it.

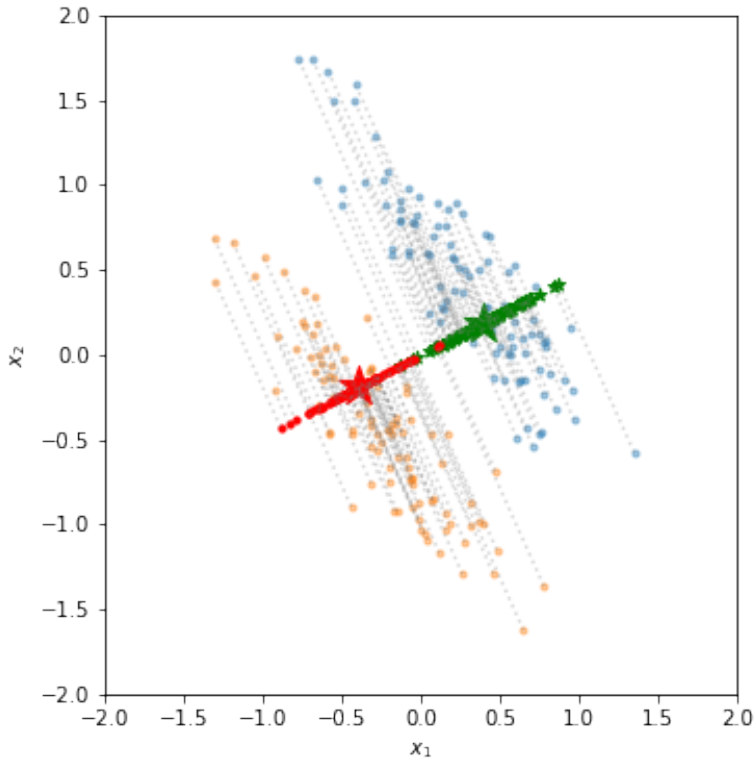


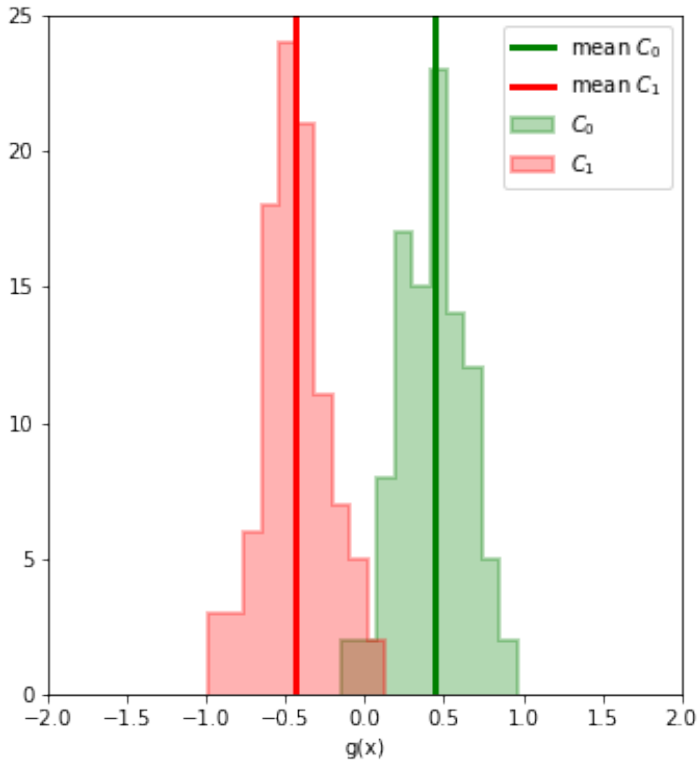
Separability in the output of the discriminant function



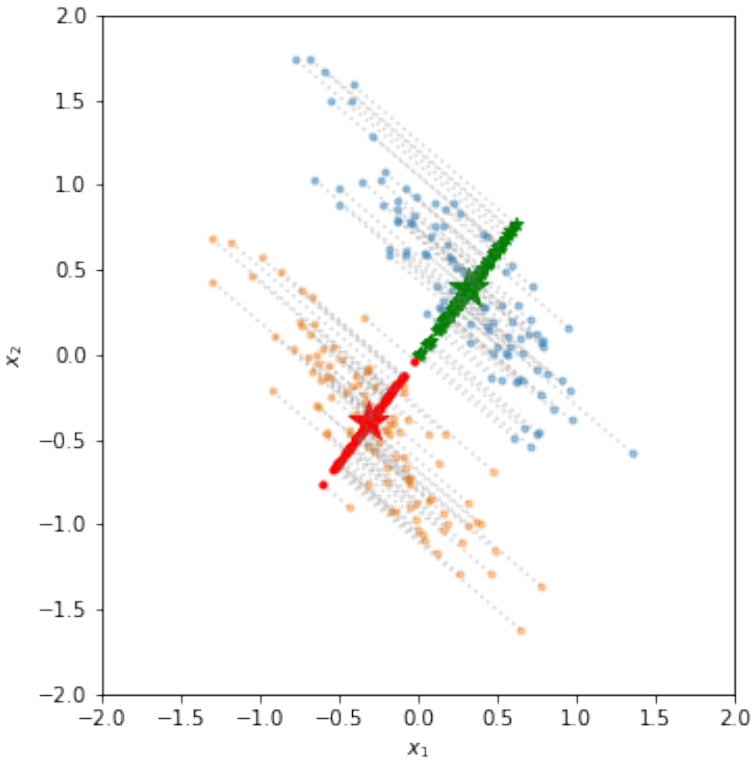


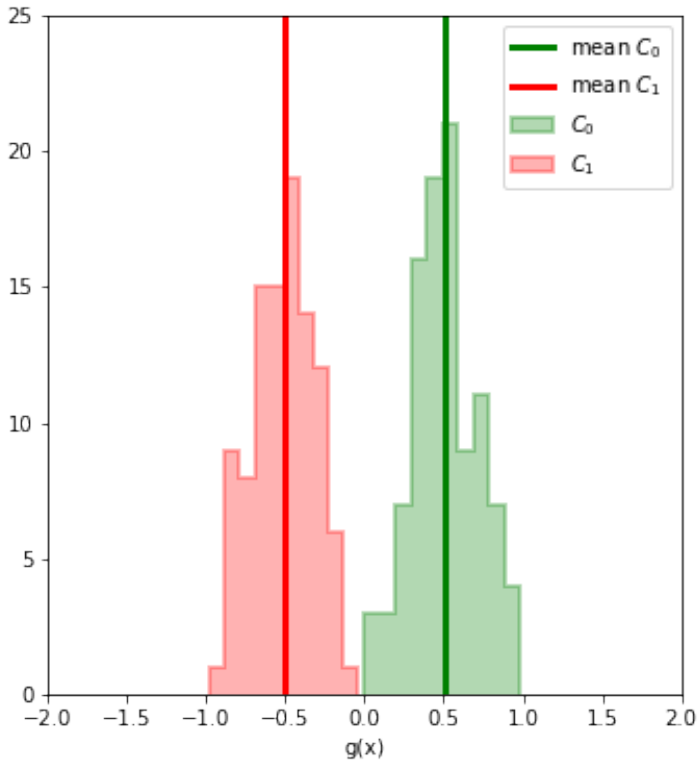
Separability in the output of the discriminant function



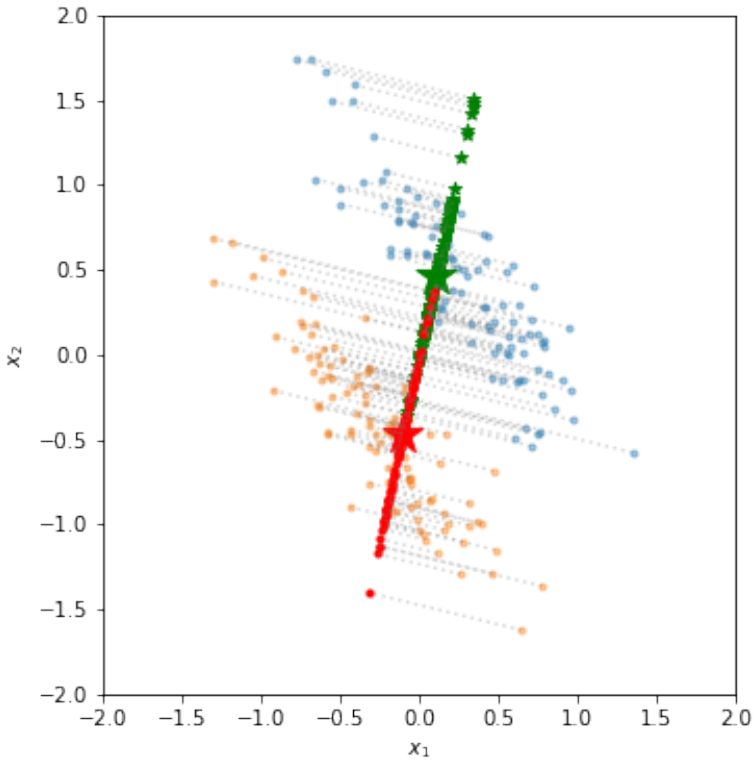


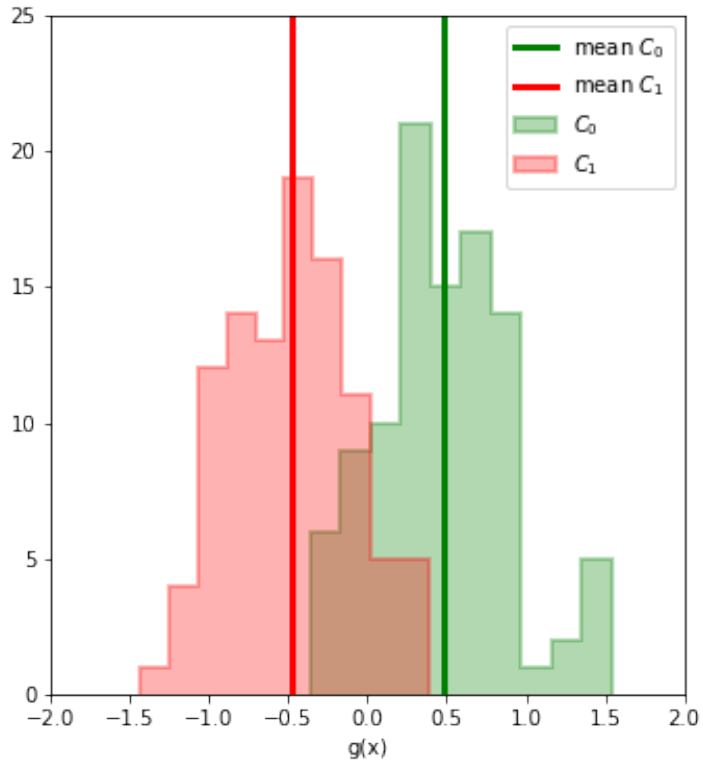
Separability in the output of the discriminant function



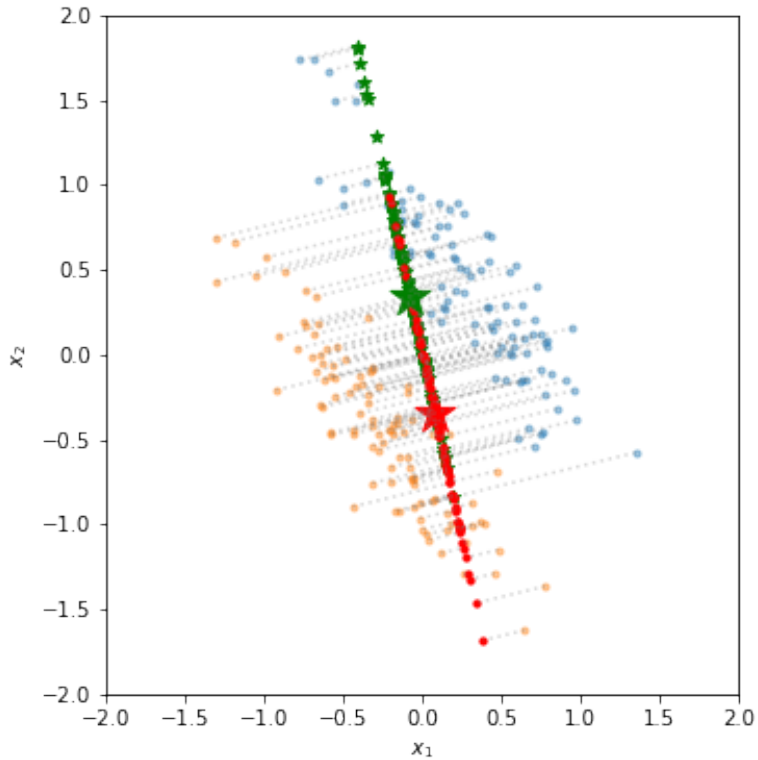


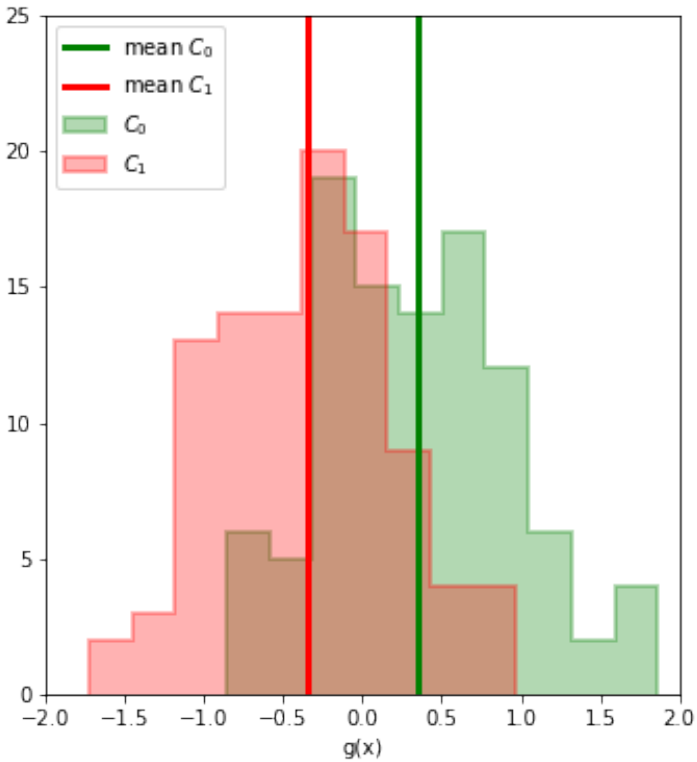
Separability in the output of the discriminant function



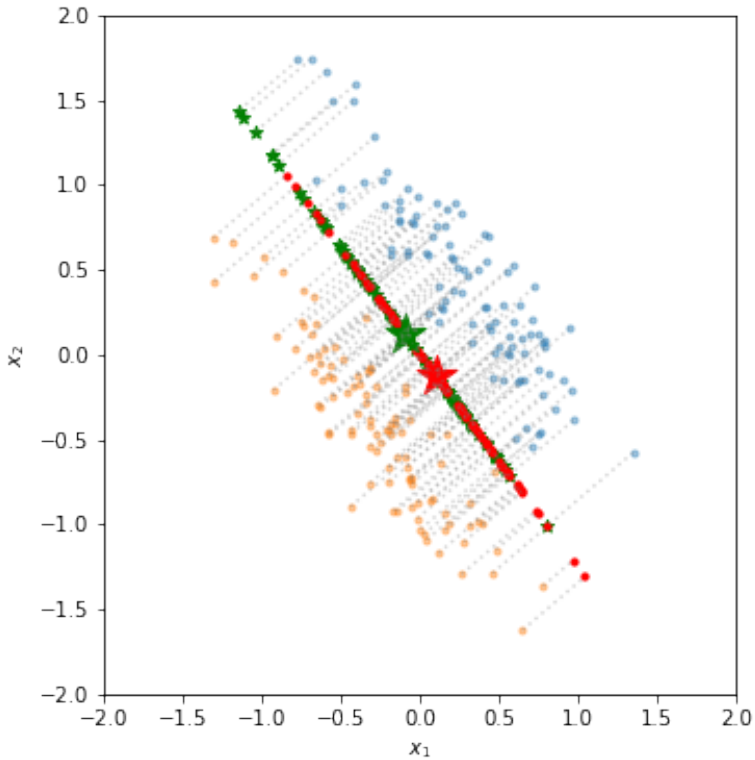


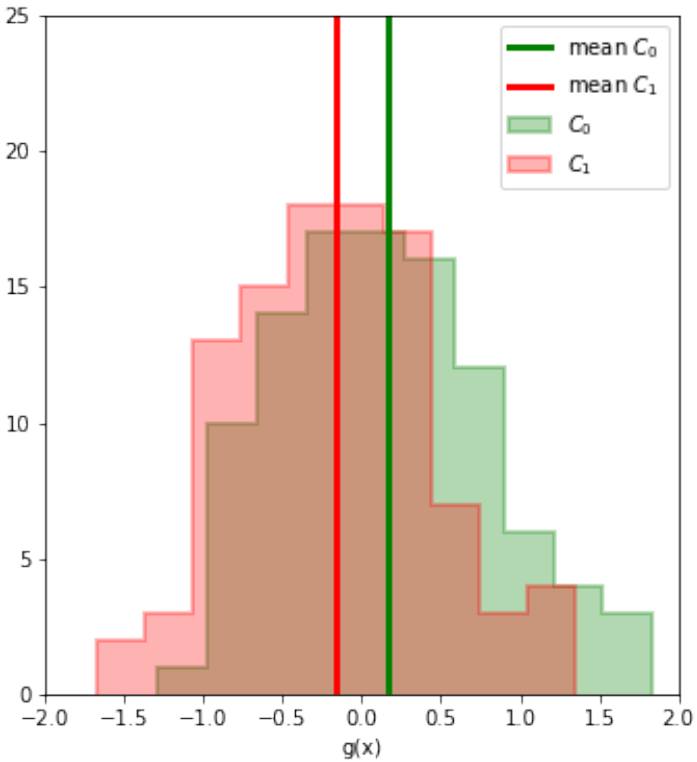
Separability in the output of the discriminant function



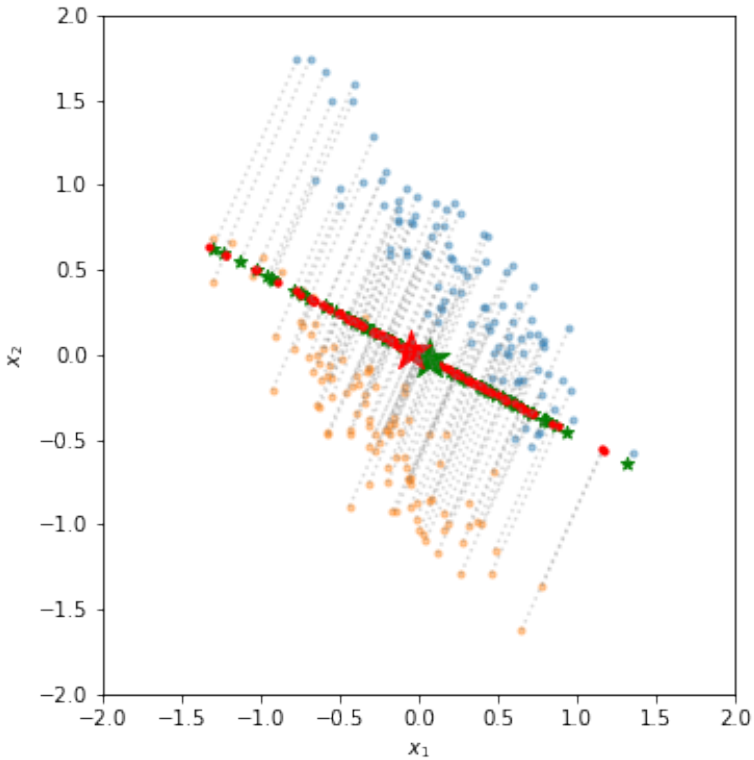


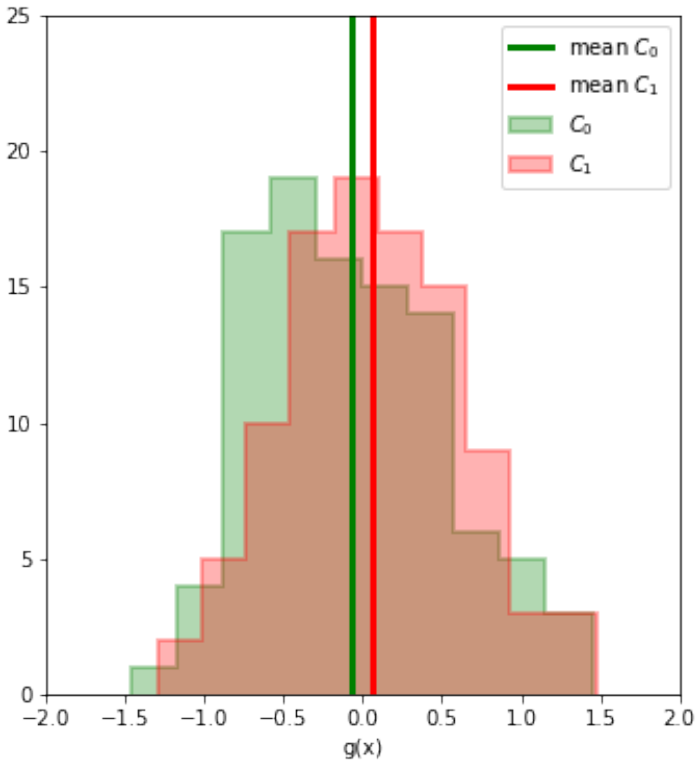
Separability in the output of the discriminant function



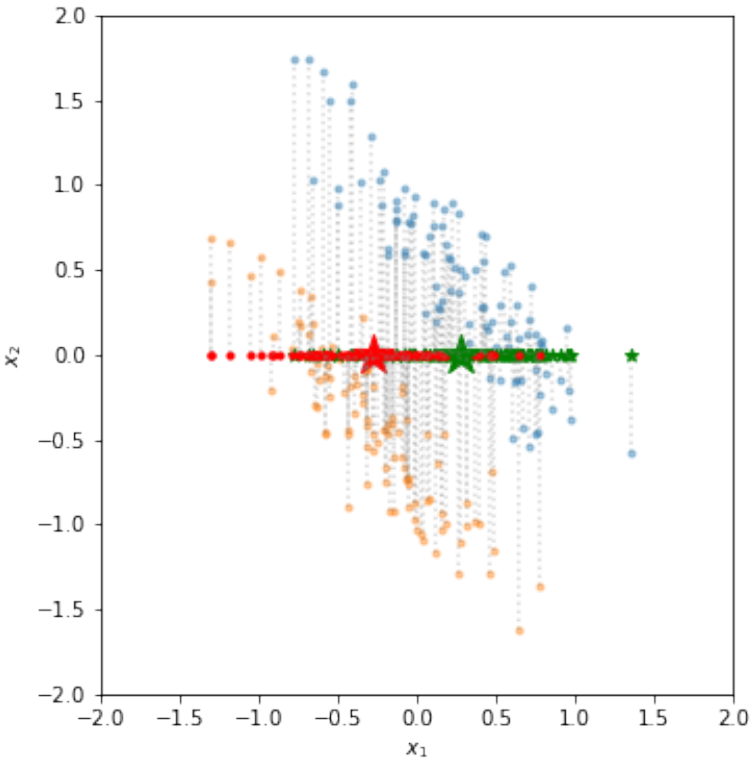


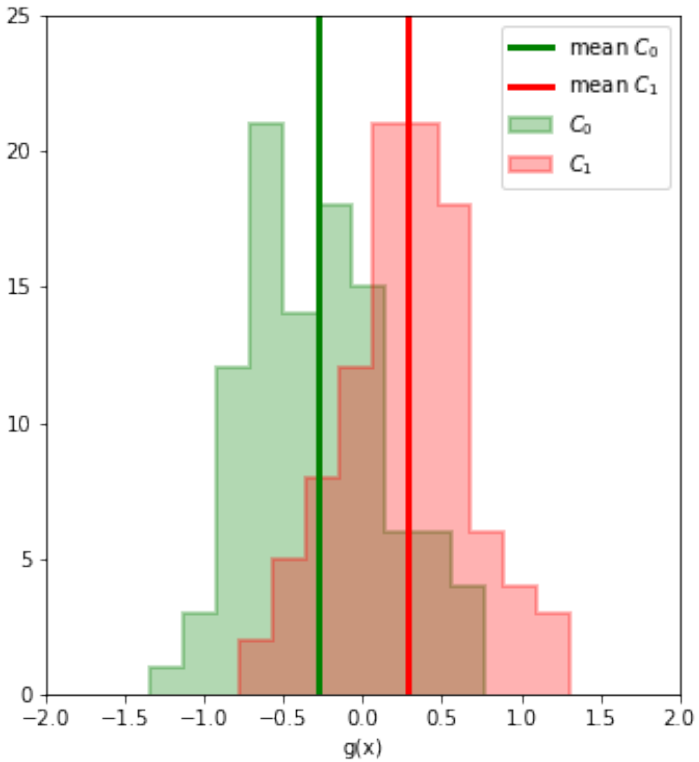
Separability in the output of the discriminant function





Separability in the output of the discriminant function





Maths for the criterion for separability

- **Separate the means** of the scalar datasets that result after applying a linear discriminant function to the training observations of each class.
- And **minimize the variances** of the 1D variables resulting from projecting the original training data with the discriminant function

Separate the means will be a better proxy for class separability if the classes are somewhat **compact** (all observations are concentrated around their corresponding means)

Optimization

$$\max_{\mathbf{w}} J(\mathbf{w}) = \frac{(m_1 - m_{-1})^2}{s_{-1}^2 + s_1^2}$$

$$\text{subject to } \mathbf{w}^\top \mathbf{w} = 1$$

where

$$s_{-1}^2 = \sum_{i \in C_{-1}} (g(\mathbf{x}_i) - m_{-1})^2, \quad s_1^2 = \sum_{i \in C_1} (g(\mathbf{x}_i) - m_1)^2$$

Fisher's Linear Discriminant Optimization

$$\max_{\mathbf{w}} J(\mathbf{w}) = \frac{(m_1 - m_{-1})^2}{s_{-1}^2 + s_1^2} =$$

$$= \frac{\mathbf{w}^\top (\mathbf{m}_1 - \mathbf{m}_{-1})(\mathbf{m}_1 - \mathbf{m}_{-1})^\top \mathbf{w}}{\sum_{i \in C_{-1}} \mathbf{w}^\top (\mathbf{x}_i - \mathbf{m}_{-1})(\mathbf{x}_i - \mathbf{m}_{-1})^\top \mathbf{w} + \sum_{i \in C_1} \mathbf{w}^\top (\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^\top \mathbf{w}}$$

$$= \max_{\mathbf{w}} \frac{\mathbf{w}^\top (\mathbf{m}_1 - \mathbf{m}_{-1})(\mathbf{m}_1 - \mathbf{m}_{-1})^\top \mathbf{w}}{\mathbf{w}^\top \left(\sum_{i \in C_{-1}} (\mathbf{x}_i - \mathbf{m}_{-1})(\mathbf{x}_i - \mathbf{m}_{-1})^\top + \sum_{i \in C_1} (\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^\top \right) \mathbf{w}}$$

where

$$\mathbf{m}_{-1} = \frac{1}{N_{-1}} \sum_{y_i = -1} \mathbf{x}_i, \quad m_{-1} = \mathbf{w}^\top \mathbf{m}_{-1} \qquad \mathbf{m}_1 = \frac{1}{N_1} \sum_{y_i = +1} \mathbf{x}_i, \quad m_1 = \mathbf{w}^\top \mathbf{m}_1$$

Between class and Intra-class covariance matrices

The numerator can be written as

$$\mathbf{w}^\top(\mathbf{m}_1 - \mathbf{m}_{-1})(\mathbf{m}_1 - \mathbf{m}_{-1})^\top\mathbf{w} = \mathbf{w}^\top S_B\mathbf{w}$$

where $S_B = (\mathbf{m}_1 - \mathbf{m}_{-1})(\mathbf{m}_1 - \mathbf{m}_{-1})^\top$ is the **between class covariance matrix**

The denominator can also be written in a more compact way:

$$\mathbf{w}^\top\left(\sum_{i\in C_{-1}}(\mathbf{x}_i - \mathbf{m}_{-1})(\mathbf{x}_i - \mathbf{m}_{-1})^\top + \sum_{i\in C_1}(\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^\top\right)\mathbf{w} = \mathbf{w}^\top S_W\mathbf{w}$$

where

$$S_W = \sum_{i\in C_{-1}}(\mathbf{x}_i - \mathbf{m}_{-1})(\mathbf{x}_i - \mathbf{m}_{-1})^\top + \sum_{i\in C_1}(\mathbf{x}_i - \mathbf{m}_1)(\mathbf{x}_i - \mathbf{m}_1)^\top$$

is the **intra-class covariance matrix**.

Fisher's Linear Discriminant solution

$$\max_{\mathbf{w}} J(\mathbf{w}) = \frac{\mathbf{w}^\top S_B \mathbf{w}}{\mathbf{w}^\top S_W \mathbf{w}}$$

$$\text{subject to } \mathbf{w}^\top \mathbf{w} = 1$$

Gradientes wrt $\mathbf{w}$ equal to zero

$$\frac{2S_B\mathbf{w}(\mathbf{w}^\top S_W \mathbf{w}) - 2S_W\mathbf{w}(\mathbf{w}^\top S_B \mathbf{w})}{(\mathbf{w}^\top S_W \mathbf{w})^2} = 0$$

$$\Rightarrow S_B\mathbf{w}(\mathbf{w}^\top S_W \mathbf{w}) = S_W\mathbf{w}(\mathbf{w}^\top S_B \mathbf{w}) \Rightarrow S_B\mathbf{w} = \frac{(\mathbf{w}^\top S_B \mathbf{w})}{(\mathbf{w}^\top S_W \mathbf{w})} S_W\mathbf{w}$$

Notice $\frac{(\mathbf{w}^\top S_B \mathbf{w})}{(\mathbf{w}^\top S_W \mathbf{w})}$ is a scalar. Since we are actually seeking the direction of $\mathbf{w}$, not its size, we can replace this scalar by a constant c

$$\Rightarrow S_B\mathbf{w}(\mathbf{w}^\top S_W \mathbf{w}) = S_W\mathbf{w}(\mathbf{w}^\top S_B \mathbf{w}) \Rightarrow S_B\mathbf{w} = cS_W\mathbf{w}$$

Fisher's Linear Discriminant solution

Multiply both sides by S_W^{-1}

$$S_W^{-1} S_B \mathbf{w} = c\mathbf{w}$$

From the definition of S_B:

$$S_B \mathbf{w} = (\mathbf{m}_1 - \mathbf{m}_{-1})(\mathbf{m}_1 - \mathbf{m}_{-1})^\top \mathbf{w} = \beta(\mathbf{m}_1 - \mathbf{m}_{-1})$$

$S_B\mathbf{w}$ goes along the direction given by $(\mathbf{m}_1 - \mathbf{m}_{-1})$ with length β. Since we just want the direction of $\mathbf{w}$, we don't need to compute the exact value of β. Using this result in the main equation:

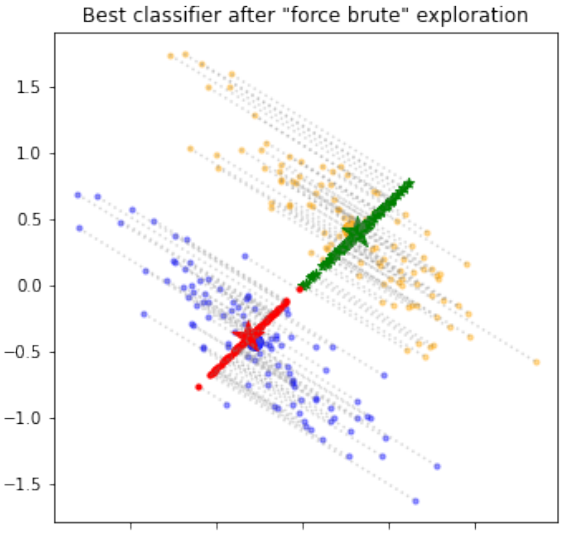
$$\beta' S_W^{-1}(\mathbf{m}_1 - \mathbf{m}_{-1}) = \mathbf{w}$$

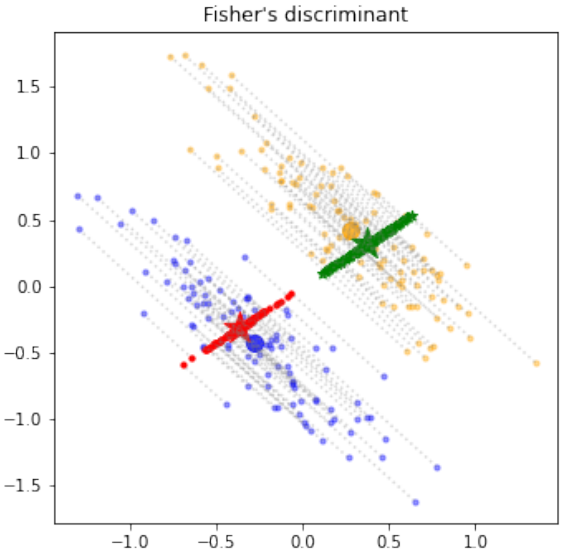
So $\mathbf{w}$ is a unit vector in the direction given by $S_W^{-1}(\mathbf{m}_1 - \mathbf{m}_{-1})$.

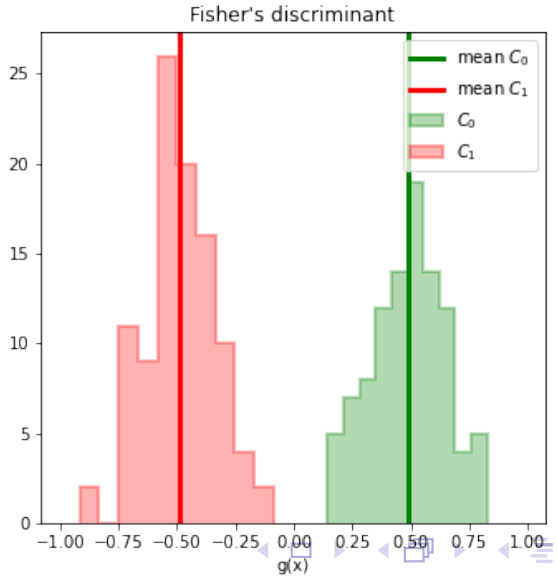
Fisher's discriminant

Vector $\mathbf{w}$ defines the direction of the **Fisher's discriminant**. This classifier must be completed with a threshold w_0 to compare the output of the discriminant function and make the decision

Fisher's discriminant vs. Brute Force







Index

Separating each mean from the overall mean

Separate m_{-1} from m_1 is equivalent to **simultaneously separate m_{-1} and m_1 from m**, where m is the **mean of the whole training set**.

$$(m_1 - m)^2 + (m_0 - m)^2 = m_1^2 - 2m_1 m + m^2 + m_0^2 - 2m_0 m + m^2$$
$$= m_1^2 + m_0^2 + 2m^2 - 2m_0 m - 2m_1 m$$

If we assume equiprobable classes, the overall mean is

$$m = \frac{1}{2}(m_1 + m_0)$$

Using this result in the main equation

$$(m_1 - m)^2 + (m_0 - m)^2 =$$

$$m_1^2 + m_0^2 + \frac{1}{2}(m_1^2 + m_0^2 + 2m_1 m_0) - m_0(m_1 + m_0) - m_1(m_1 + m_0) =$$

$$m_1^2 + m_0^2 + \frac{1}{2}(m_1^2 + m_0^2 + 2m_1 m_0) - 2m_0 m_1 - m_0^2 - m_1^2 =$$

$$= \frac{1}{2}m_1^2 + \frac{1}{2}m_0^2 - m_1 m_0 = \frac{1}{2}(m_1 - m_0)^2$$

Linear separability in more than 2 dimensions

- The Fisher's discriminant in a binary problem can be interpreted as a mapping of the input data into a 1 dimensional space (scalar variable) and then separate the means of the resulting classes in 1D.
- If we have $C > 2$ classes, we would need $C - 1$ dimensions to linearly separate the mean of each class from the overall mean
- Therefore a problem with $C > 2$ classes needs the observations to live in a space of at least $d = C - 1$ dimensions, in order to be able to map these observations into the $C - 1$ subspace in which the linear discrimination can take place
- **The number of linearly independent columns of the dataset imposes a limit in the number of classes that can be separated with a Multiclass Linear Discriminant**

Mapping into $C-1$ dimensions

Fisher's discriminant in a case with $C > 2$ classes must map the input data into a subspace of dimensionality $C-1$.
This mapping is carried out with $C-1$ unit vectors $\{\mathbf{w}_q\}_{q=1}^{C-1}$ that we need to find. These $\mathbf{w}_q$ are the rows of a $(C-1) \times d$ matrix W

$$\mathbf{z}_i = W\mathbf{x}_i = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \vdots \\ \mathbf{w}_{C-1}^\top \end{bmatrix} \mathbf{x}_i$$

Therefore, the means of the overall dataset and of each mapped class are

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{z}_i \in \mathbb{R}^q$$

$$\boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{y_i=1} \mathbf{z}_i \in \mathbb{R}^q, \quad \boldsymbol{\mu}_2 = \frac{1}{N_2}\sum_{y_i=2} \mathbf{z}_i \in \mathbb{R}^q, \quad \ldots \quad \boldsymbol{\mu}_C = \frac{1}{N_C}\sum_{y_i=M} \mathbf{z}_i \in \mathbb{R}^q$$

where N_c is the number of observations of class C_c in the training set

Optimization Problem in the mapped space

- Between-class covariance

$$s_B = \sum_{c=1}^{C} N_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^\top$$

- Intra-class covariance

$$s_W = s_1 + s_2 + \cdots + s_C = \sum_{c=1}^{C} \sum_{y_i=c} (\mathbf{z}_i - \boldsymbol{\mu}_c)(\mathbf{z}_i - \boldsymbol{\mu}_c)^\top$$

- Functional

$$J(W) = \max_W \text{Trace}\{s_W^{-1} s_B\}$$

Between-class matrix in the input space

$$s_B = \sum_{c=1}^{C} N_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^\top$$

$$= \sum_{c=1}^{C} N_c W(\mathbf{m}_c - \mathbf{m})(\mathbf{m}_c - \mathbf{m})^\top W^\top$$

$$= W\left(\sum_{c=1}^{C} N_c(\mathbf{m}_c - \mathbf{m})(\mathbf{m}_c - \mathbf{m})^\top\right) W^\top =$$

$$W S_B W^\top$$

where

$$S_B = \sum_{c=1}^{C} N_c(\mathbf{m}_c - \mathbf{m})(\mathbf{m}_c - \mathbf{m})^\top$$

Intra-class matrix in the input space

$$s_W = \sum_{c=1}^{C} \sum_{y_i=c} (\mathbf{z}_i - \boldsymbol{\mu}_c)(\mathbf{z}_i - \boldsymbol{\mu}_c)^\top$$

$$= \sum_{c=1}^{C} \sum_{y_i=c} W(\mathbf{x}_i - \mathbf{m}_c)(\mathbf{x}_i - \mathbf{m}_c)^\top W^\top$$

$$= W \left(\sum_{c=1}^{C} \sum_{y_i=c} (\mathbf{x}_i - \mathbf{m}_c)(\mathbf{x}_i - \mathbf{m}_c)^\top \right) W^\top =$$

$$= W S_W W^\top$$

where

$$S_W = \sum_{c=1}^{C} \sum_{y_i=c} (\mathbf{x}_i - \mathbf{m}_c)(\mathbf{x}_i - \mathbf{m}_c)^\top$$

Optimization Problem in input space

$$J(W) = \max_W \text{Trace}\{s_W^{-1} s_B\}$$

Introducing the definitions of the between-class and intra-class covariances in terms of the input space data:

$$s_B = W S_B W^\top$$

$$s_W = W S_W W^\top$$

yields

$$J(W) = \max_W \text{Trace}\{(W S_W W^\top)^{-1} W S_B W^\top\}$$

This optimization turns out into computing the **eigenvalues and eigenvectors** of matrix $S_W^{-1} S_B$. Precisely W is formed with the $C-1$ **eigenvectors** of $S_W^{-1} S_B$ with **largest eigenvalues**
Since S_B has a rank less or equal than $C-1$, $S_W^{-1} S_B$ will have maximum $C-1$ different eigenvalues.

Index

Index

Approximate the posterior with a linear function

The critical pdf in a binary classification problem is the posterior of the class $P(y = 1|\mathbf{x})$. We can use this posterior to build the following discriminant function

Discriminant Function based on the posterior

$$g(\mathbf{x}) = P(y = +1|\mathbf{x}) \underset{D=0}{\overset{D=1}{\gtrless}} \frac{1}{2}$$

Logistic Regression proposes to **approximate the posterior** with the composition of a **linear regressor** and a **sigmoid**

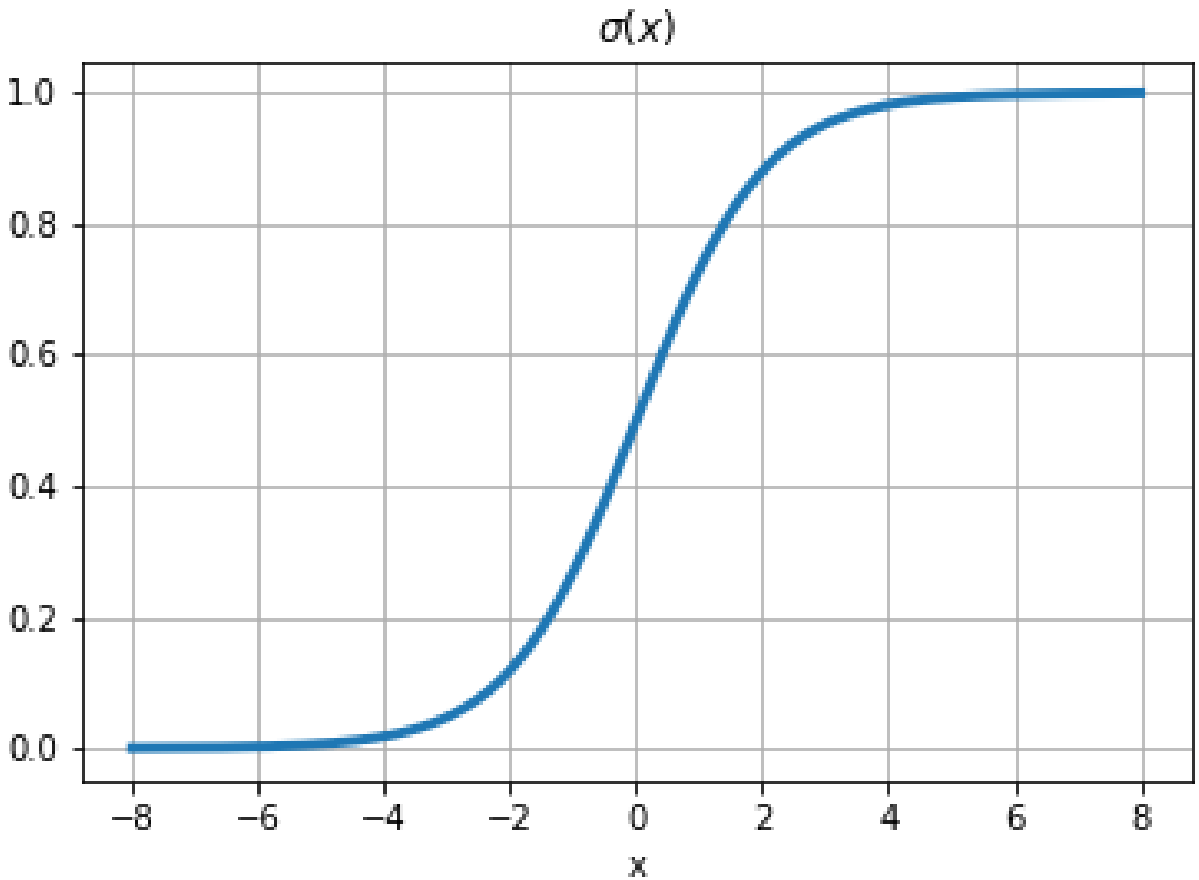
Logistic Regression

$$g(\mathbf{x}) = P(y = +1|\mathbf{x}) = \sigma(w_0 + \mathbf{w}^\top \mathbf{x}) = \sigma(\mathbf{w}_e^\top \mathbf{x}_e)$$

Sigmoid functions

Sigmoid

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



Derivatives of sigmoids

$$\frac{d}{dx}\sigma(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)\frac{e^{-x}}{1+e^{-x}} = \sigma(x)(1-\sigma(x))$$

$$\frac{d}{dx}\log\sigma(x) = \frac{1}{\sigma(x)}\frac{d}{dx}\sigma(x) = (1-\sigma(x))$$

$$\frac{d}{dx}\log(1-\sigma(x)) = \frac{1}{1-\sigma(x)}\frac{d}{dx}(-\sigma(x)) = \frac{-\sigma(x)(1-\sigma(x))}{1-\sigma(x)} - \sigma(x)$$

Joint posterior of the training set

In the logistic regression framework we will use the following notation to relate classes and hypothesis:

$$y_i = \begin{cases} +1 & \text{if } \mathbf{x}_i \in H_1 \\ 0 & \text{if } \mathbf{x}_i \in H_0 \end{cases}$$

Now we introduce a vector $\mathbf{y}$ including all the true targets of the training set

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

The joint posterior of $\mathbf{y}$ will be

$$P(\mathbf{y}|\mathbf{w}) = \prod_{i=1}^{N} P(y_i|\mathbf{w}) = \prod_{i=1}^{N} \left(\frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x})} \right)^{y_i} \left(1 - \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x})} \right)^{1-y_i}$$

Take logs in the joint posterior

$$J(\mathbf{w}) = \log P(\mathbf{y}|\mathbf{w}) = \sum_{i=1}^{N} y_i \log(\sigma(\mathbf{x}_i, \mathbf{w})) + (1-y_i) \log\left(1 - \sigma(\mathbf{x}_i, \mathbf{w})\right)$$

Index

Newton's Method

2nd order Taylor's expansion

$$J(\mathbf{w}) \approx J(\mathbf{w}_0) + (\mathbf{w} - \mathbf{w}_0)\nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_0} + \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^\top H(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0)$$

In the minimum the gradient will be zero: $\nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}^*} = \mathbf{0}$.
The gradient of the second order expansion is:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_0} + H(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0)$$

Therefore if the minimum sits on $\mathbf{w} = \mathbf{w}^*$

$$\nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}^*} = \nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_0} + H(\mathbf{w}_0)(\mathbf{w}^* - \mathbf{w}_0) \Rightarrow$$

$$\mathbf{w}^* = \mathbf{w}_0 - H(\mathbf{w}_0)^{-1}\nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_0}$$

Newton's Method in the logistic regression case

- Gradient

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \nabla_{\mathbf{w}} \left(\sum_{i=1}^{N} y_i \log(\sigma(\mathbf{x}_i, \mathbf{w})) + (1 - y_i) \log\left(1 - \sigma(\mathbf{x}_i, \mathbf{w})\right) \right)$$

$$= \sum_{i=1}^{N} \mathbf{x}_i (y_i - \sigma(\mathbf{x}_i, \mathbf{w})) = X^\top (\mathbf{y} - \boldsymbol{\sigma})$$

- Hessian

$$H(\mathbf{w}_0) = \nabla_{\mathbf{w}} \nabla_{\mathbf{w}} J(\mathbf{w}) = \nabla_{\mathbf{w}} \left(- \sum_{i=1}^{N} \sigma(\mathbf{x}_i, \mathbf{w}) \mathbf{x}_i \right)$$

$$= - \sum_{i=1}^{N} \nabla_{\mathbf{w}} \sigma(\mathbf{x}_i, \mathbf{w}) \mathbf{x}_i = - \sum_{i=1}^{N} \sigma(\mathbf{x}_i, \mathbf{w})(1 - \sigma(\mathbf{x}_i, \mathbf{w})) \mathbf{x}_i \mathbf{x}_i^\top = X^\top R X$$

where R is a diagonal matrix with elements $R_{ii} = \sigma(\mathbf{x}_i, \mathbf{w})(1 - \sigma(\mathbf{x}_i, \mathbf{w}))$

Newton's Method in the logistic regression case

- Recursive main equation, we can't solve in one step as $\sigma()$ depends on $\mathbf{w}$

$$\mathbf{w}^* = \mathbf{w}_0 - H(\mathbf{w}_0)^{-1} \nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}=\mathbf{w}_0}$$

$$\Rightarrow \mathbf{w}^* = \mathbf{w}_{\text{old}} - (X^\top R X)^{-1} X^\top (\mathbf{y} - \boldsymbol{\sigma})$$

$$= (X^\top R X)^{-1} \left(X^\top R X \mathbf{w}_{\text{old}} - X^\top (\mathbf{y} - \boldsymbol{\sigma}) \right)$$

$$= (X^\top R X)^{-1} X^\top \left(R X \mathbf{w}_{\text{old}} - (\mathbf{y} - \boldsymbol{\sigma}) \right)$$

$$= (X^\top R X)^{-1} X^\top R \left(X \mathbf{w}_{\text{old}} - R^{-1} (\mathbf{y} - \boldsymbol{\sigma}) \right)$$

$$= (X^\top R X)^{-1} X^\top R \mathbf{z}$$

where

$$\mathbf{z} = \left(X \mathbf{w}_{\text{old}} - R^{-1} (\mathbf{y} - \boldsymbol{\sigma}) \right)$$

Recursive solution: Iterative Recursive Weighted Least Squares

Due to the interdependence between $\sigma()$ and $\mathbf{w}$, we need to implement the following recursion

1. Random initial guess for $\mathbf{w}_{\text{old}}$
2. Compute $\boldsymbol{\sigma}$ using $\mathbf{w}_{\text{old}}$
3. Compute R using $\boldsymbol{\sigma}$
4. Compute $\mathbf{z} = \left(X\mathbf{w}_{\text{old}} - R^{-1}(\mathbf{y} - \boldsymbol{\sigma})\right)$
5. Compute $\mathbf{w}^* = (X^\top RX)^{-1}X^\top R\mathbf{z}$
6. Make $\mathbf{w}_{\text{old}} = \mathbf{w}^*$ and go to step 2 until convergence

Since the Hessian is positive semidefinite the convergence to a global optimum is guaranteed.

Index

Index

Cross Validation

- Method for estimating the **generalization capability** of any machine learning model
- Simulates the process of evaluating the model with a **separate set** of data not used for training.

Algorithm:

1. Split the training set in N subsets called folds
2. Loop for $n = 1, \ldots, N$
   1. Fit the model with a training set formed by the union of all the folds but n (notice your training set size is that of $N - 1$ folds)
   2. Evaluate the model with subset n and store the *score*
3. Use the **average** of th N scores as estimation to the score that you will obtain when you train the model with all the training data and use a real test set in the evaluation.

Grid Search Cross Validation

Commonly used method to obtain **good values for the hyperparmeters** of a model

- k in kNN
- Maximum number of leaf nodes or depth of the tree in decision trees
- Number of trees in a random forest

Algorithm:

1. Define a range of values you want to explore for each hyperparameter to be tuned
2. Construct a **grid** that spans all the possible combinations of hyperparameters. If for instance you have to explore 3 parameters a, b and c with ranges M_a, M_b and M_c the size of the gird will be $M_a \times M_b \times M_c$
3. Estimate the quality of each combination of hyperparameters running a **cross-validation** on each node of the grid
4. return the values for the hyperparameters that form the node of the grid that achieved the best cross validation score.

Index

True positive, False negative...

In a binary classification the outcome of the classifier can be also categorized into 4 groups:

True Positive Those observations whose true target is the positive class and the predicted class is also the positive one. TP: number of True Positives

False Positive Those observations whose true target is the negative class but the predicted class is the positive one. FP: number of False Positives

True Negative Those observations whose true target is the negative class and the predicted class is also the negative one. TN: number of True Negatives

False Negative Those observations whose true target is the positive class but the predicted class is the negative one. FN: number of False Negatives

		Predicted class	
		Negative	Positive
True class	Negative	TN	FP
	Positive	FN	TP

Relation with false alarms and detections

- Probability of False Alarm

$$P_{\text{FA}} \approx \frac{FP}{FP + TN}$$

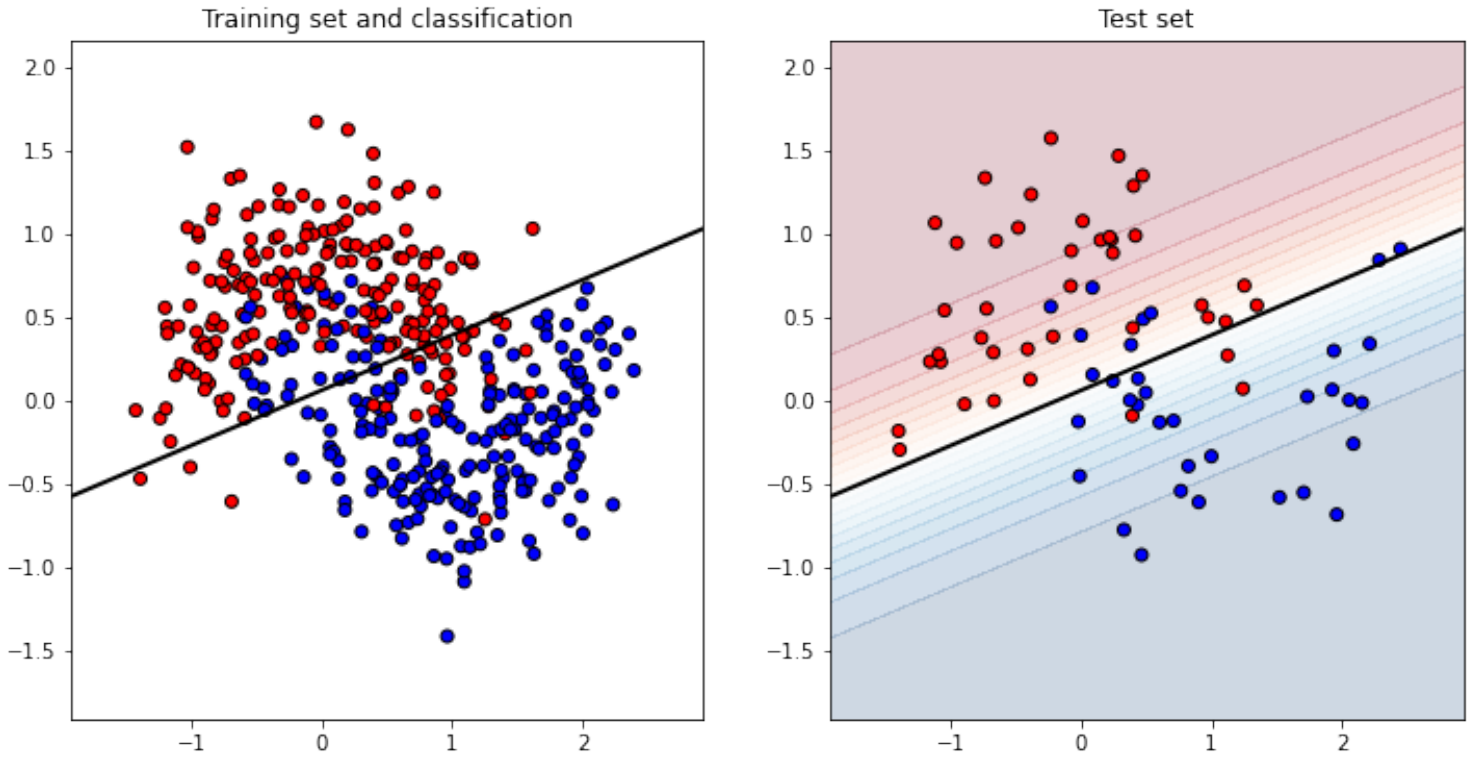
- Probability of Missing Targets

$$P_{\text{M}} \approx \frac{FN}{TP + FN}$$

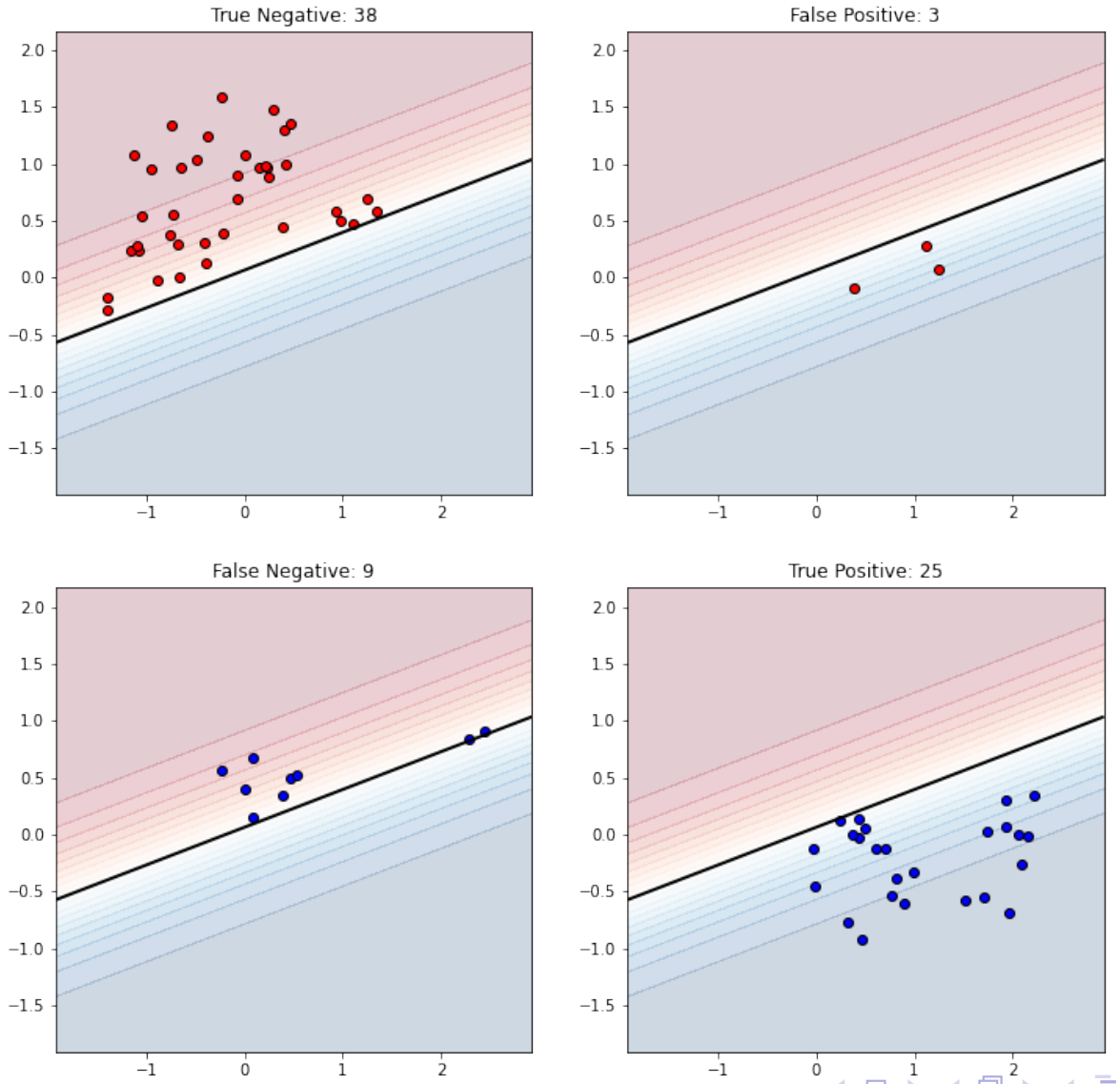
- Probability of Detection

$$P_{\text{D}} \approx \frac{TP}{TP + FN}$$

Example


Training set and classification
Test set
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1
0
1
2

Example


True Negative: 38
False Positive: 3
False Negative: 9
True Positive: 25

Confusion matrix

- Binary classification

$$\begin{bmatrix} TN & FP \\ FN & TP \end{bmatrix}$$

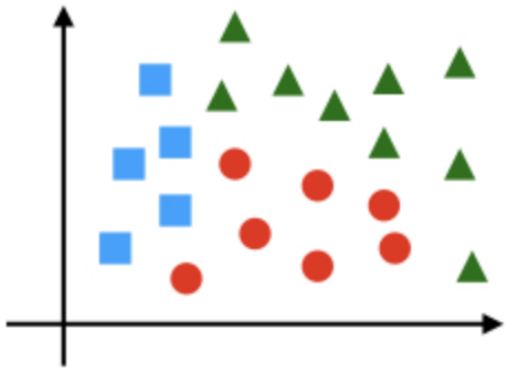
$$\underset{\text{unnormalized}}{\begin{bmatrix} 38 & 3 \\ 9 & 25 \end{bmatrix}} \quad \underset{\text{normalize='true'}}{\begin{bmatrix} 0.93 & 0.07 \\ 0.26 & 0.74 \end{bmatrix}} \quad \underset{\text{normalize='pred'}}{\begin{bmatrix} 0.81 & 0.11 \\ 0.19 & 0.89 \end{bmatrix}} \quad \underset{\text{normalize = 'all'}}{\begin{bmatrix} 0.51 & 0.04 \\ 0.12 & 0.33 \end{bmatrix}}$$

- Multiclass classification
  Element $M[i, j]$ contains the number of observations whose true class is C_i but were put into class C_j by the classifier

  Diagonal elements indicate the hits of the classifier, while off-diagonal elements indicate classification errors.

Index

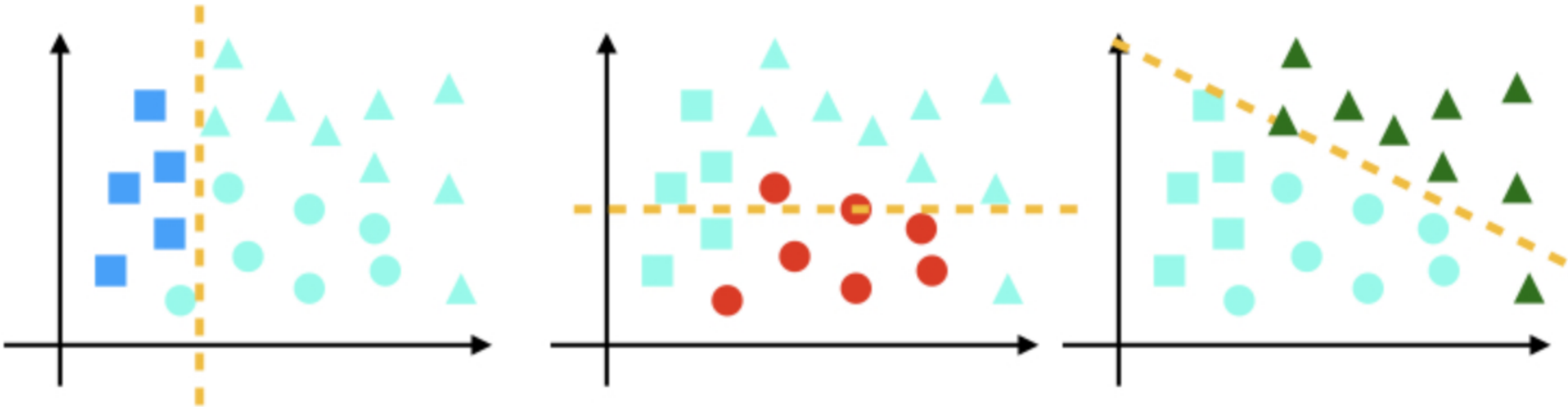
Linear classification in multiclass problems



The linear discriminant function is natural in binary classification. There are two common approaches to **tackle multiclass problems with binary classifiers**

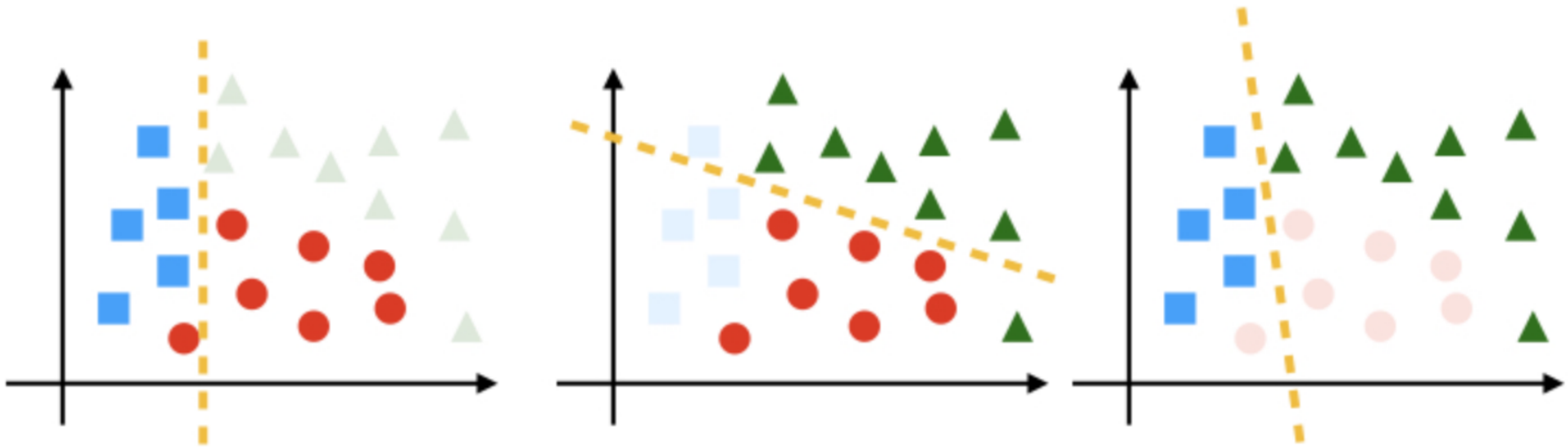
- **One vs All**: In a case with $C > 2$ classes, train C linear classifiers, each focused in separating one of the classes (positive class of the c classifier) from the rest (the training observations that belong to classes different from c form the negative class)
- **One vs One**: n a case with $C > 2$ classes, train $C(C-1)/2$ linear classifiers, one per every pair of classes.

One vs All Classification



Predict each test observation with the C classifiers and assign it to the class whose corresponding classifer achieves the larger value of the discriminant function.

One vs One Classification



Predict each test observation with all the classifiers and assign it to the class that achieves a larger number of votes among all the classifiers.

To solve ties we could also look at the values of the discriminant functions.

Index

Discriminant functions with nonlinear elements

An immediate way to introduce non-linear elements in the discriminant function is to **extend the input matrix with columns that incorporate non-linear functions of the original variables**

These new columns act in the discriminant function as if we had added more variables. The overall discriminant is a linear combination of all the columns. Since these new columns are non-linear functions of the original variables, it turns out that the discriminant function that results from the application of one of the linear methods discussed in this lecture in the extended data is a non-linear function of the original variables.

Example of nonlinear discriminant functions