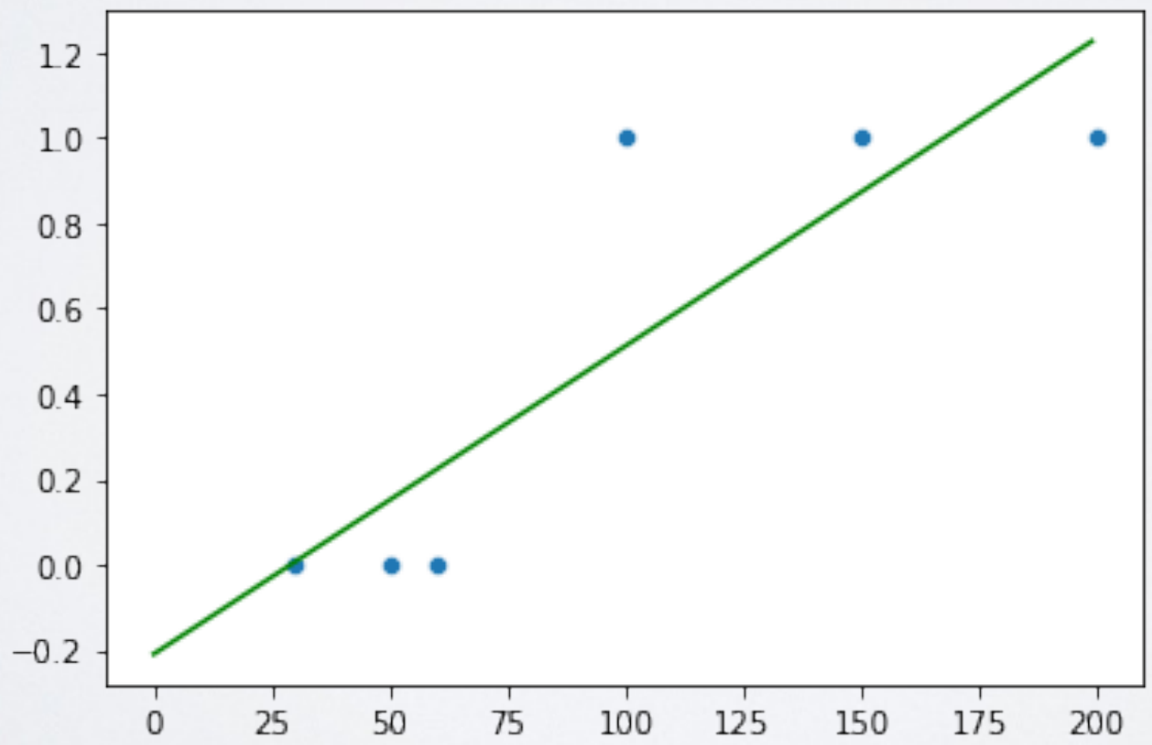
CLASSIFICATION

CLASSIFICATION

- Objective: predict the class of an item
- Methods for regression can be reused with some adaptations
  - Binary Classification is usually simple
  - Multiclass Classification might require more changes
- Evaluation is different

LINEAR CLASSIFICATION

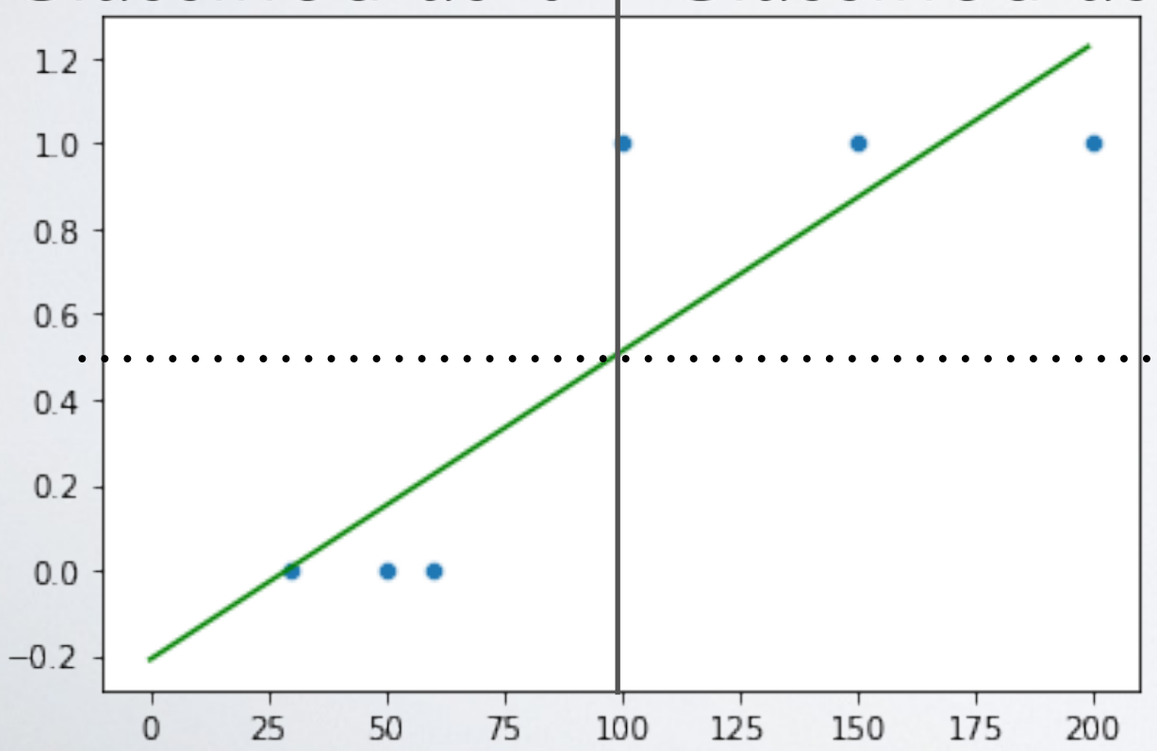
- We can easily adapt linear regression
- Imagine a 1 feature example:
  - We want to classify between apartments and houses
  - Our (unique) feature is dwelling surface



LINEAR CLASSIFICATION

- We can easily adapt linear regression
- Imagine a 1 feature example:
  - We want to classify between apartments and houses
  - Our (unique) feature is dwelling surface

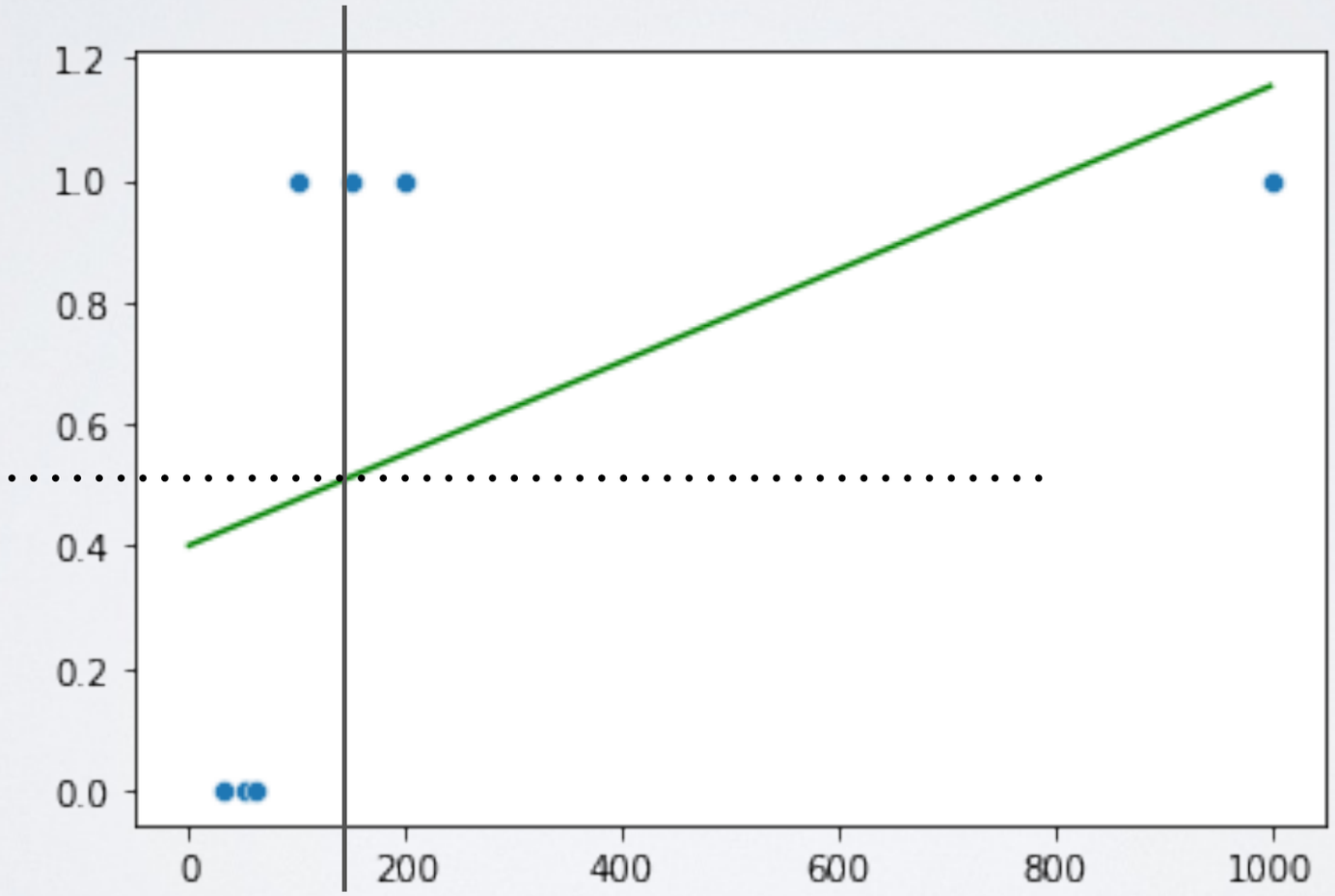
Classified as 0 | Classified as 1




```
MSE 0.0636152055857253 8
RMSE 0.252220549491363 6
MAE 0.205068528575122 92
R2 0.745539177657098 5
```

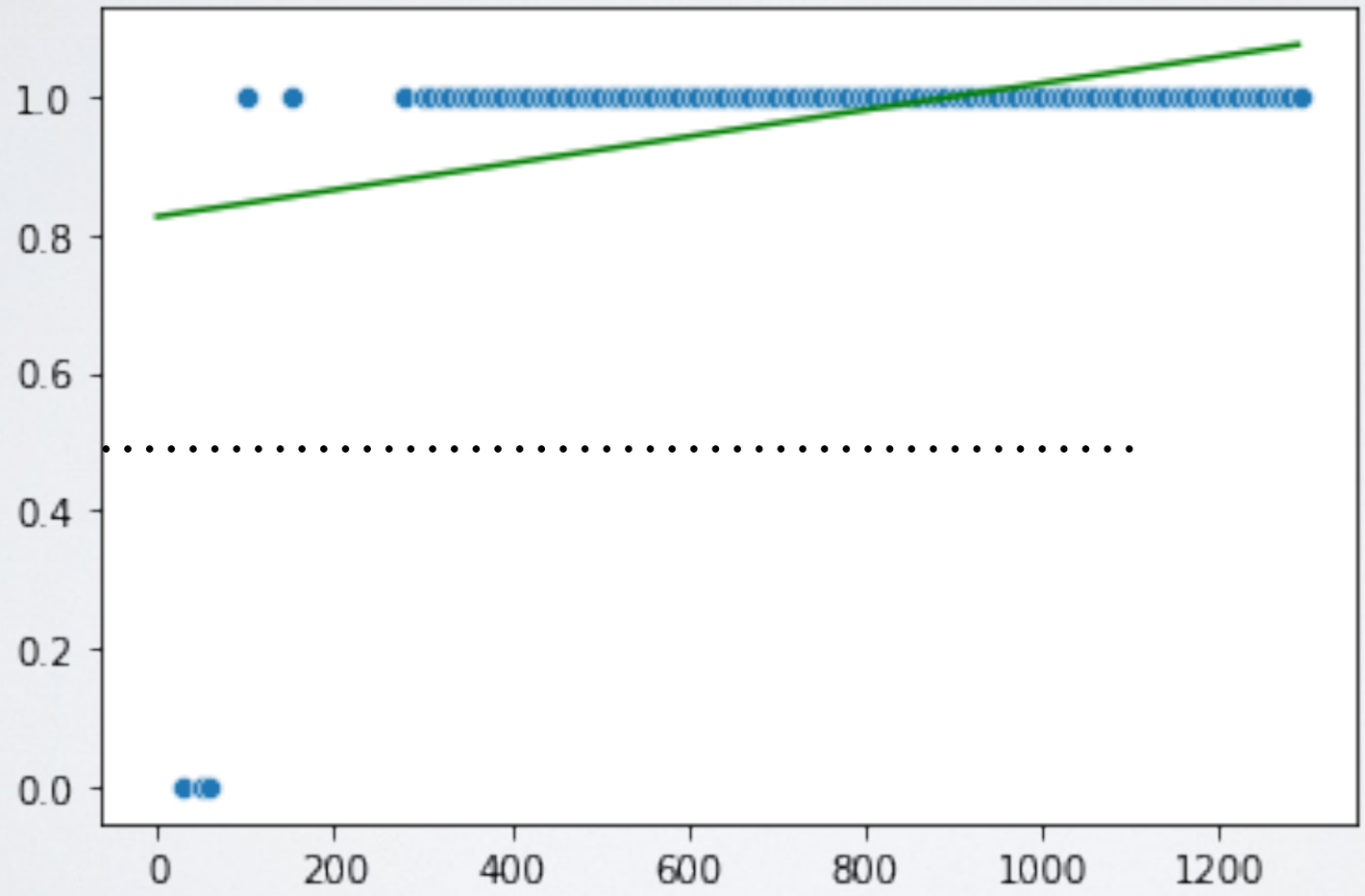
LINEAR CLASSIFICATION

- Weaknesses: Outliers



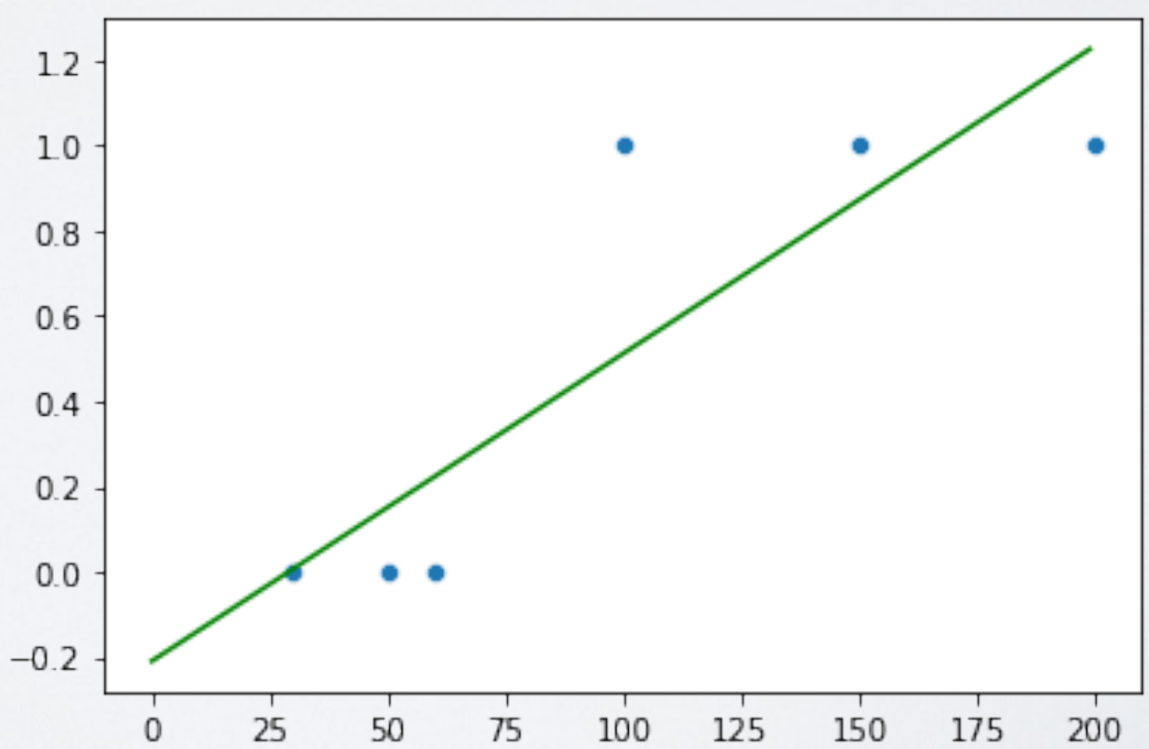
LINEAR CLASSIFICATION

- Weaknesses: Class imbalance

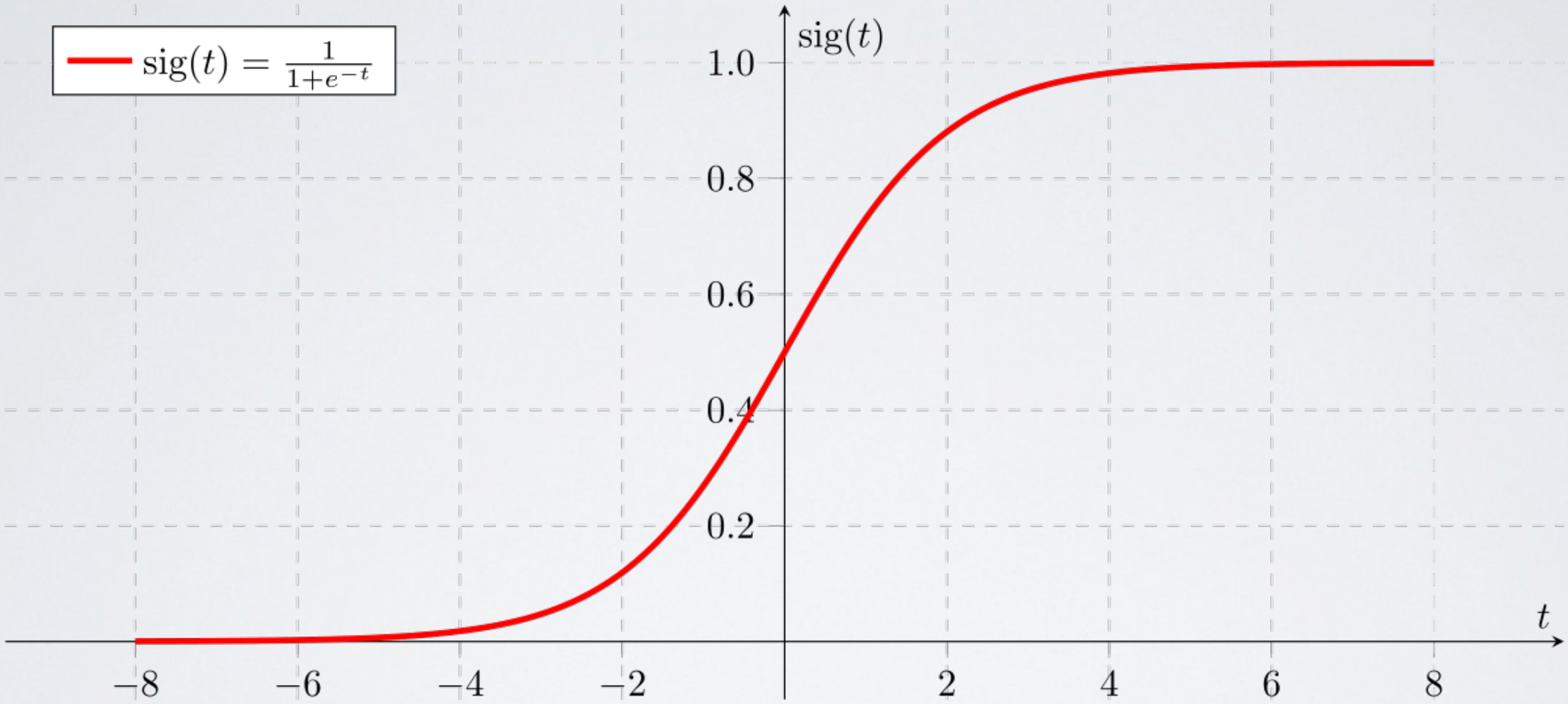


LINEAR CLASSIFICATION

- More generally, inadapted objective:
  - The relation is not linear
  - We minimize a cost function (MSE) which is not meaningful:
    - Some predictions go *beyond* possible values (prediction less than 0 or more than 1 adding error



SIGMOID FUNCTION



$$\lim_{t \to -\infty} sig(t) = 0 \qquad \lim_{t \to +\infty} sig(t) = 1 \qquad sig(0) = 0.5$$

LOGISTIC REGRESSION

Logisitic (Sigmoid) function:

$$Sig(x) = \frac{1}{1 + e^{-x}}$$

Linear regression:

$$\hat{y} = \beta_0 + \beta_1 x_i + \beta_2 x_2 + \ldots + \beta_n x_n$$

Logistic Regression:

$$P(y = 1) = Sig(\beta_0 + \beta_1 x_i + \beta_2 x_2 + \ldots + \beta_n x_n)$$

$$P(y = 1) = \frac{1}{1 + e^{-\beta_0 + \beta_1 x_i + \beta_2 x_2 + \ldots + \beta_n x_n}}$$

LOGISTIC REGRESSION

$$P(y=1) = \frac{1}{1+e^{-\beta_0+\beta_1 x_i+\beta_2 x_2+\ldots+\beta_n x_n}}$$

$$\frac{1}{P(y=1)} = 1+e^{-\beta_0+\beta_1 x_i+\beta_2 x_2+\ldots+\beta_n x_n}$$

$$\frac{1-P(y=1)}{P(y=1)} = e^{-\beta_0+\beta_1 x_i+\beta_2 x_2+\ldots+\beta_n x_n}$$

$$\frac{P(y=1)}{1-P(y=1)} = e^{\beta_0+\beta_1 x_i+\beta_2 x_2+\ldots+\beta_n x_n}$$

LOGISTIC REGRESSION

$$ln(\frac{P(y=1)}{1-P(y=1)}) = \beta_0 + \beta_1 x_i + \beta_2 x_2 + \ldots + \beta_n x_n$$

$$\frac{P(y=1)}{1-P(y=1)} = e^{\beta_0+\beta_1 x_i+\beta_2 x_2+\ldots+\beta_n x_n}$$

$$\frac{P(y=1)}{1-P(y=1)} = e^{\beta_0} e^{\beta_1 x_i} e^{\beta_2 x_2} (\ldots) e^{\beta_n x_n}$$

LOGISTIC REGRESSION

$$ln(\frac{P(y=1)}{1-P(y=1)}) = \beta_0 + \beta_1 x_i + \beta_2 x_2 + \ldots + \beta_n x_n$$

Problem to solve similar to a linear regression.
We minimize the error between true $y \in \{0,1\}$
and estimated probability of being 1

LOGISTIC REGRESSION

/!\ log transform of the target variable => multiplicative relation between variables

Interpretation as **odd ratios:**

$+1$in x_i =>prediction multiplied by e^{β_i}

$$\frac{P(y=1)}{1-P(y=1)} = e^{\beta_0} e^{\beta_1 x_i} e^{\beta_2 x_2} (\ldots) e^{\beta_n x_n}$$

https://christophm.github.io/interpretable-ml-book/logistic.html

MULTICLASS LOGISTIC REGRESSION

- In many cases, we have more than 2 classes
  - e.g.: {house, apartment, office, industrial}. {cat,dog,horse,…}
  - Categories are unordered=> conversion to numeric would be catastrophic
- Simple solution (often used): one VS all
  - Train a logistic classifier on one class VS all other classes.
  - Pick the class with the largest confidence
    - e.g.: house: 20%. Apartment: 30%. Office: 70%. Industrial: 80%=>Industrial.
  - Rather a heuristic than principled method.
- Alternative approach: softmax regression

SOFTMAX

- Softmax is a generalization of Logistic/Sigmoid to Multiclass
  - Takes several outputs with arbitrary values $\in (-\infty, +\infty)$
  - Convert into a set of (positive) probabilities summing to 1.
- $$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$
  - $\mathbf{z}$: vector of real numbers
  - Exponential convert Real into $(0, +\infty)$
  - Division by the sum normalizes (sum of values =1).

SOFTMAX

- Define the cost function to minimize as:
- $$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{e^{f(x^{(i)})}}{\sum_{j=1}^{K} e^{f(x^{(i)})}}\right]$$
  - 1{x}=1 if x:True, 0 if x:False
  - Sum for each observation and each class of the error, defined as {0,1}-proba of the class
- =>No analytical solution, optimization, e.g., gradient descent

CLASSIFICATION WITH DECISION TREE

DECISION TREE

- Trees can be easily adapted to the classification task
  - It is even more natural than for regression

- The principle is to divide observations in term of **class homogeneity**
  - We want items in the same branch/leaf to belong to the same class

DECISION TREE

- Most common homogeneity/diversity/inequality/purity scores
  - p_i: fraction of items of class i
  - Gini Coefficient: $1 - \sum_j p_j^2$
  - Entropy: $-\sum_j p_j \cdot log_2 p_j$

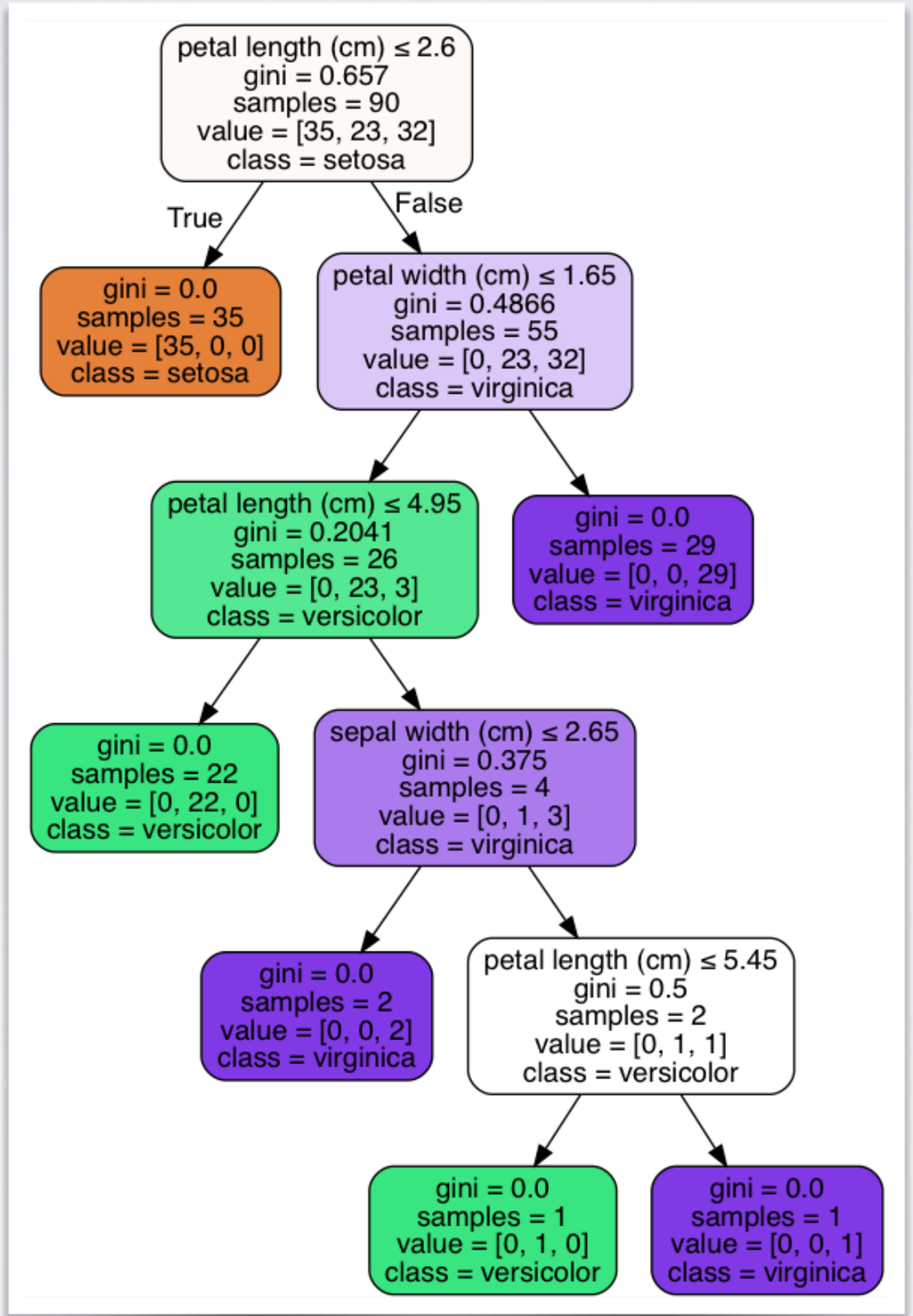
DECISION TREE

- Most common homogeneity/diversity/inequality/purity scores
  - p_i: fraction of items of class i
  - Gini Coefficient: $1 - \sum_j p_j^2$
    - Min: 0: 1 class only
    - Max: 0.5: (2 classes), 0.66(3classes), 0.75 (4classes), 0.875(8classes)
  - Interpretation:
    - If we classify by taking an element at random, probability to be wrong.

DECISION TREE

- Most common homogeneity/diversity/inequality/purity scores
  - p_i: fraction of items of class i
  - Entropy: $-\sum_j p_j \cdot log_2 p_j$
    - Min: 0: 1 class only
    - Max: 1 (2 classes), 1.584(3 classes), 2 (4 classes), 3 (8 classes), etc.
  - Interpretation: average # of bits required to encode the information of the class of each item

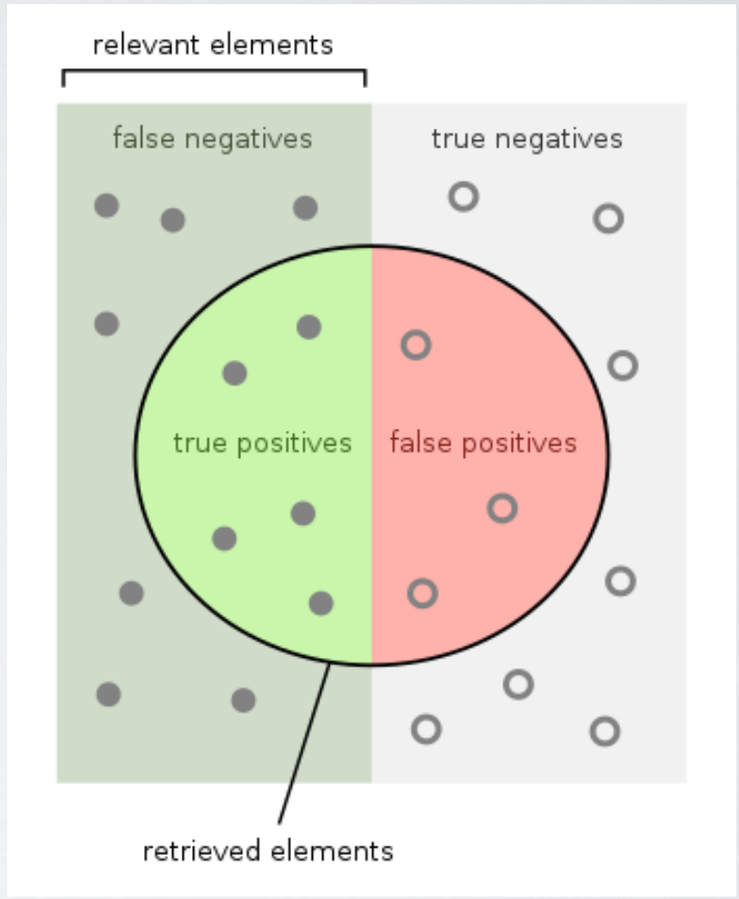
DECISION TREE



CLASSIFICATION EVALUATION

CLASSIFICATION: EVALUATION

		Actual	
		Positive	Negative
Predicted	Positive	**True Positive**	**False Positive**
	Negative	**False Negative**	**True Negative**



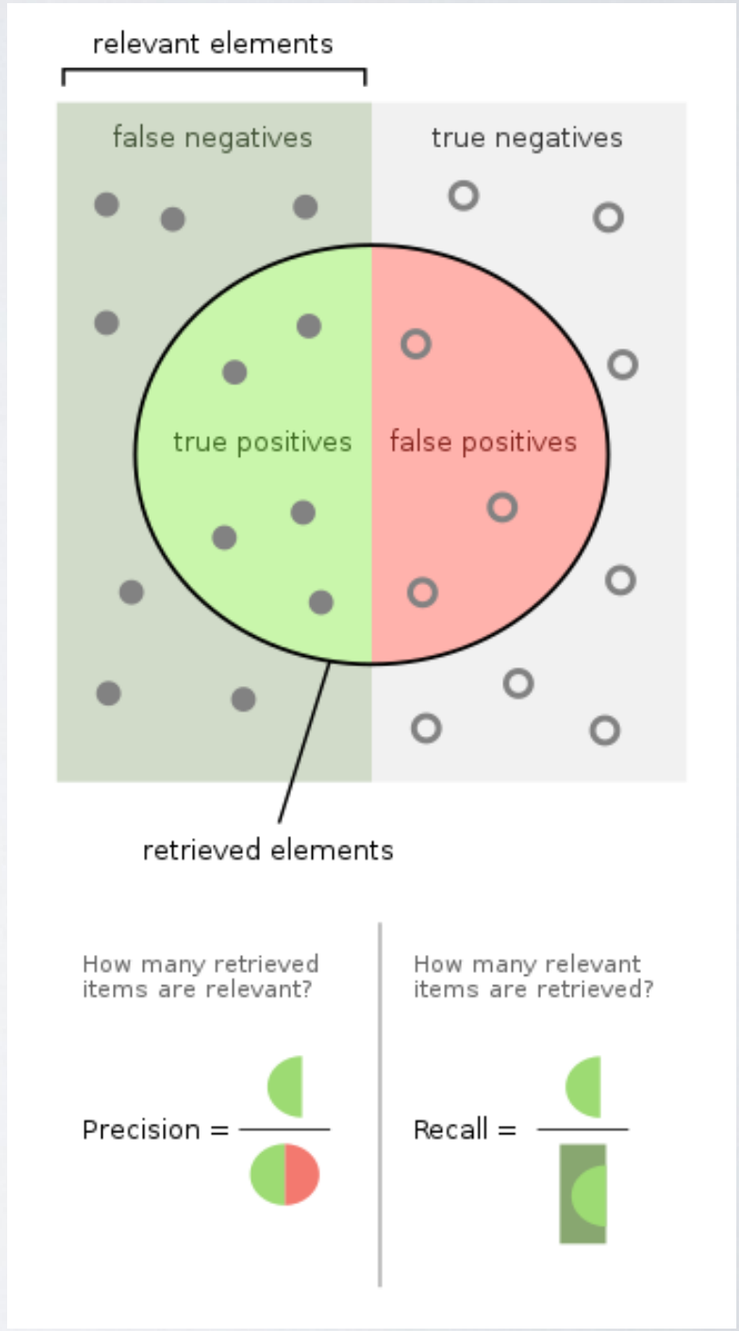
/!\ Positive=1, not 0.
Results change according to which class
Is 1.

CLASSIFICATION: EVALUATION

- Precision=$\frac{TP}{TP+FP}$
  - Among those predicted as True, fraction of really True

- Recall=$\frac{TP}{TP+FN}$
  - Among those really true, what fraction did we identity correctly

- Non-symmetric
  - Precision success != Precision failure.



ACCURACY

- Accuracy: $\frac{TP + TN}{P + N}$

- Fraction of correct prediction, among all predictions
  - Simple to interpret, symmetric

- Main drawback: class imbalance
  - Test whole city, 1 000 people, for Covid
    - 95% don't have covid, i.e., 50 people have covid, 950 don't have it
  - Our test (ML algorithm) is pretty good: TP: 45 - FN: 5 - TN: 900 -FP: 50
    - Accuracy= (45+900)/1 000=0.945
  - Dumb classifier: Always answer: not covid
    - Accuracy: (0+950)/1 000 = 0.95

F1 SCORE

- F1 score: $F_1 = 2\frac{precision * recall}{precision + recall}$
  - Harmonic mean between precision and recall
    - Harmonic mean more adapted for rates.
    - Gives more importance to the lower value
    - Not symmetric

- Scores for the covid predictor:
  - Precision=45/95=0.47
  - Recall = 45/50=0.9
  - F1=0.65

- Score for the naive predictor impossible to compute…
  - You need at least some TP !
  - Assuming 1 "free" TP (Precision=1, Recall=1/50)

RANKING-BASED EVALUATION SCORES

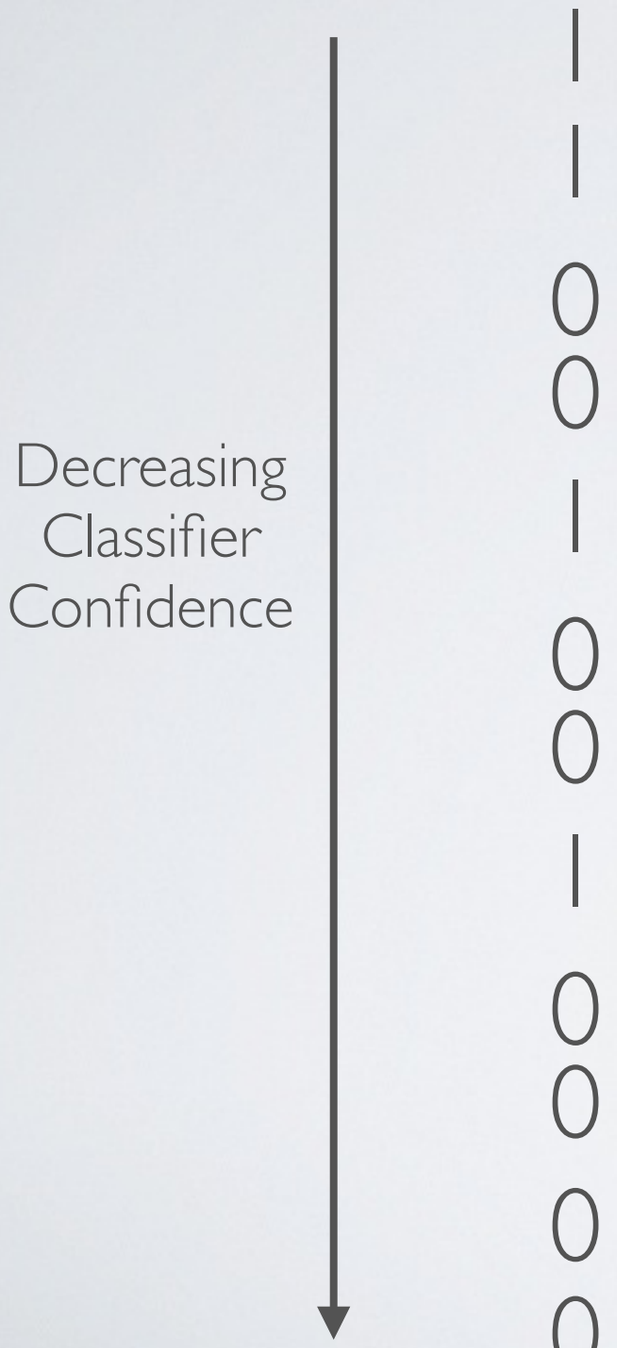
RANKING-BASED SCORES

- Most classification methods assign a probability, or score, to their prediction.
- If our objective is not really to answer a yes/no question, but rather to find some positives, we can use ranking-based approaches
  - Typical example: recommendation. Will user X buy product Z?
    - We are not really interested in having a correct classification(impossible problem), but of ranking correctly items.

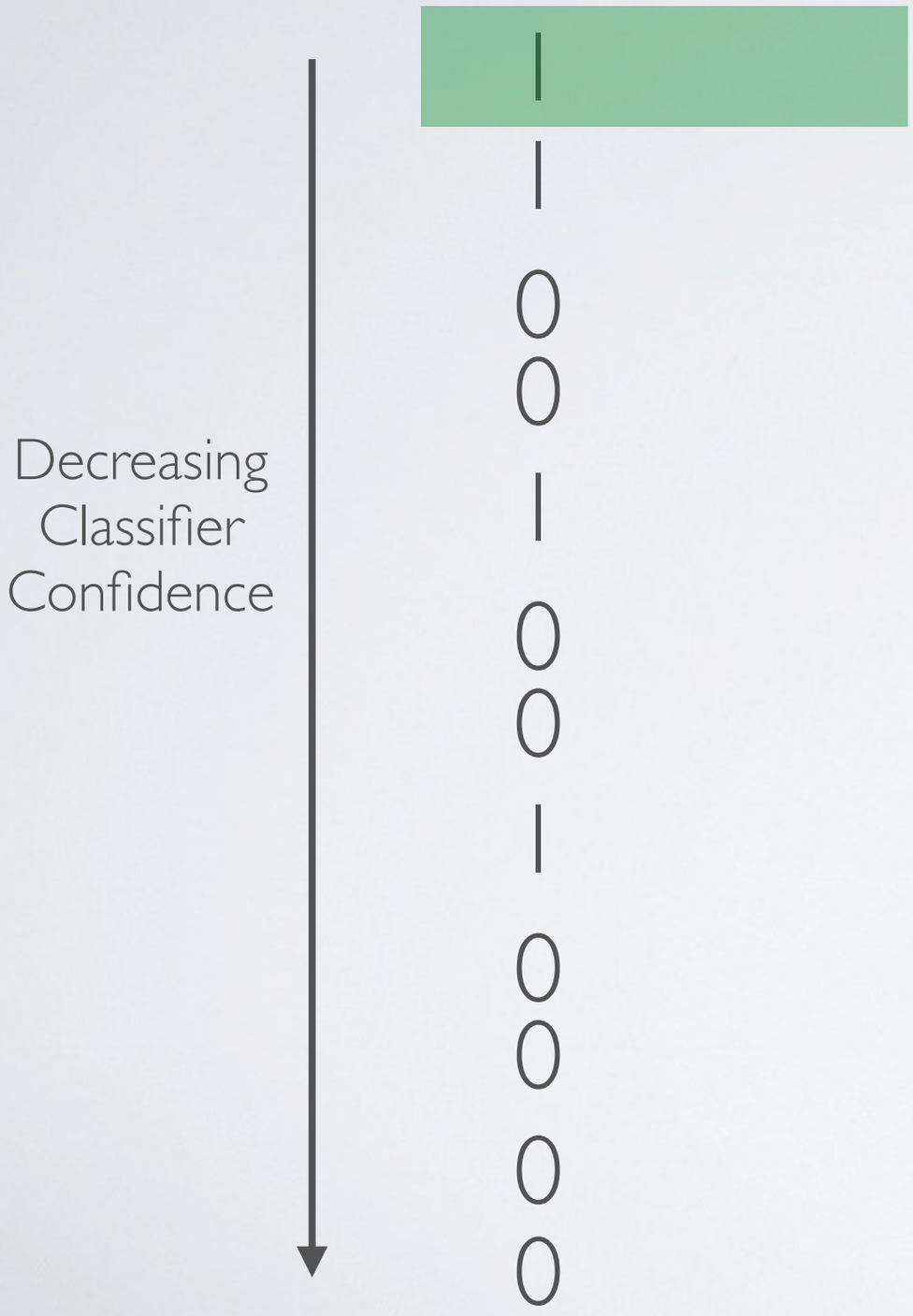
PRECISION@K

- If we know that we will do exactly k recommendations, compute the precision among the k highest scores: Precision@k
  - Typically, search engine-like evaluation
- If we don't know the exact k-value, but we know we care more about the first ones: Average Precision@k
  - Compute the precision for each value of k, weighted by the gain in recall
    - $\sum_{i}^{n} (R_i - R_{i-1}) P_i$
    - It can also be understood as the area under the Precision/Recall Curve

AVERAGE PRECISION



AVERAGE PRECISION



Green is predicted at 1, rest at 0
(Confidence threshold)

TP=1 FP=0

FN=3

$$\text{Precision}=\frac{1}{1},\ \text{Recall}=\frac{1}{1+3}$$

AVERAGE PRECISION

Top2

TP=2 FP=0

FN=2

$$\text{Precision}=\frac{2}{2},\ \text{Recall}=\frac{2}{2+2}$$

AVERAGE PRECISION

Top3

TP=2 FP=1

FN=2

$$\text{Precision}=\frac{2}{2+1},\ \text{Recall}=\frac{2}{2+2}$$

AVERAGE PRECISION

Top5

TP=3 FP=2

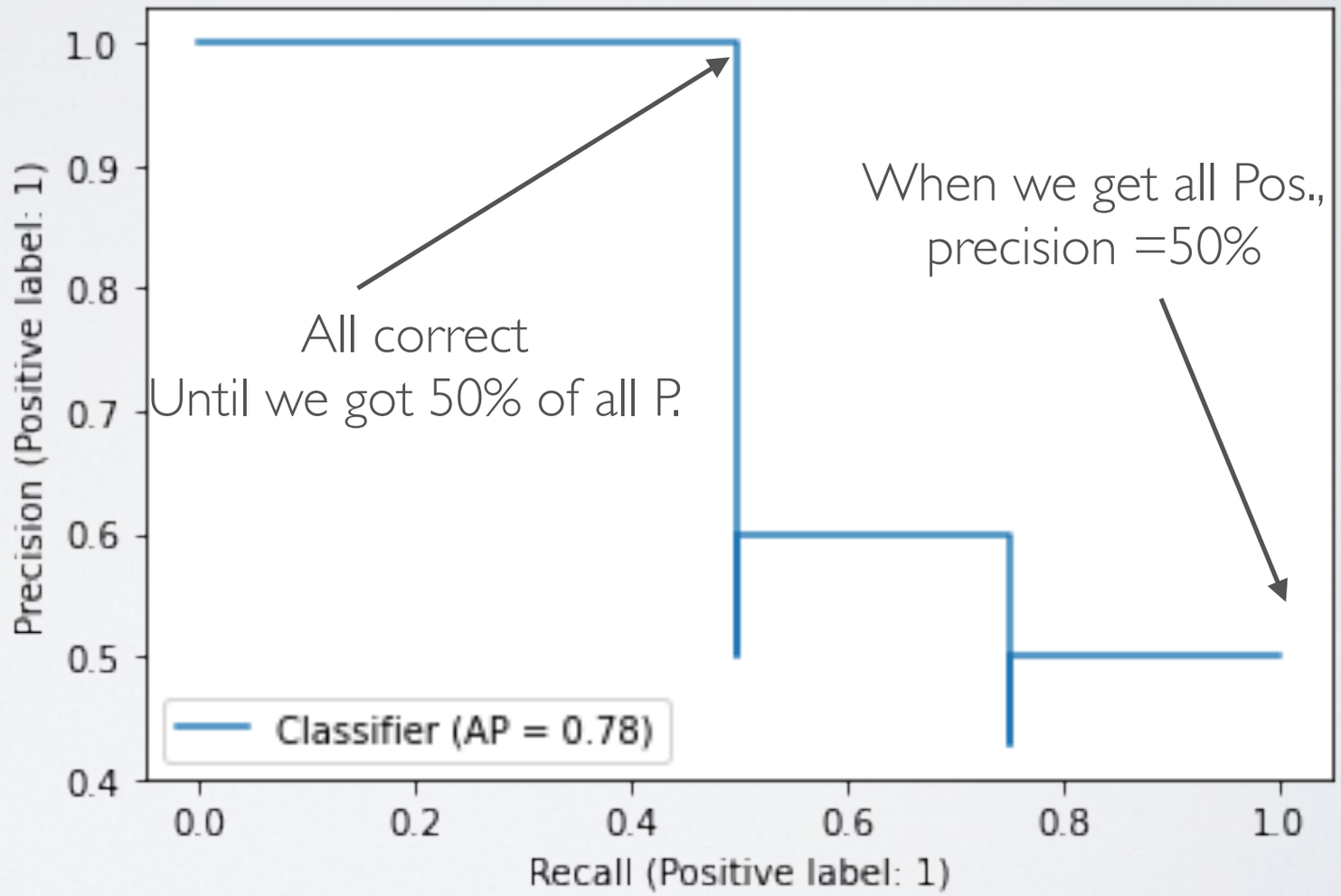
FN=1

$$\text{Precision}=\frac{3}{3+2},\ \text{Recall}=\frac{3}{3+1}$$

AVERAGE PRECISION

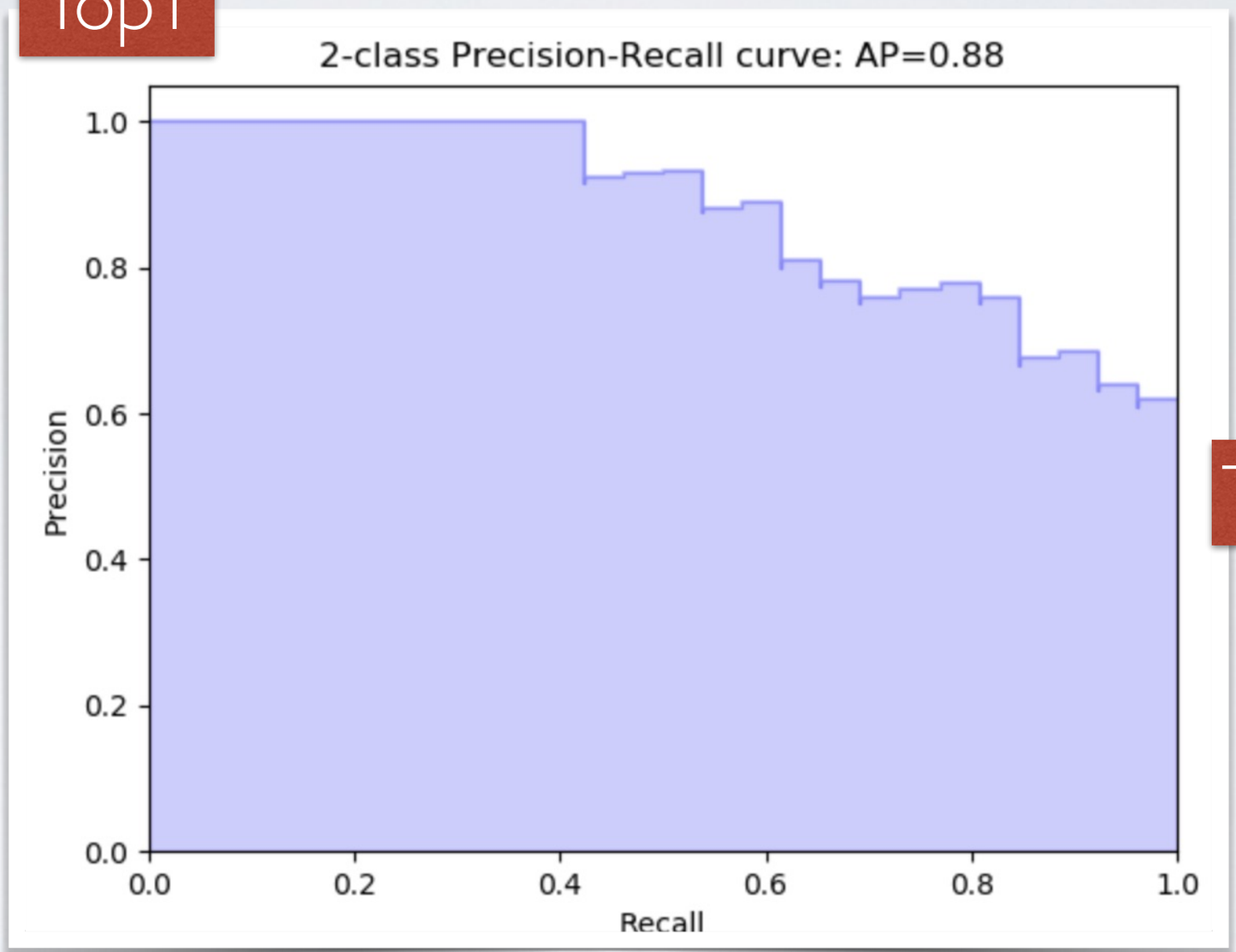
Decreasing Classifier Confidence

0 0 1 0 0 1 0 0 0 0



AVERAGE PRECISION

Top 1



Top n

AVERAGE PRECISION

- Interpretation:
  - 1: all class 1 ranked first

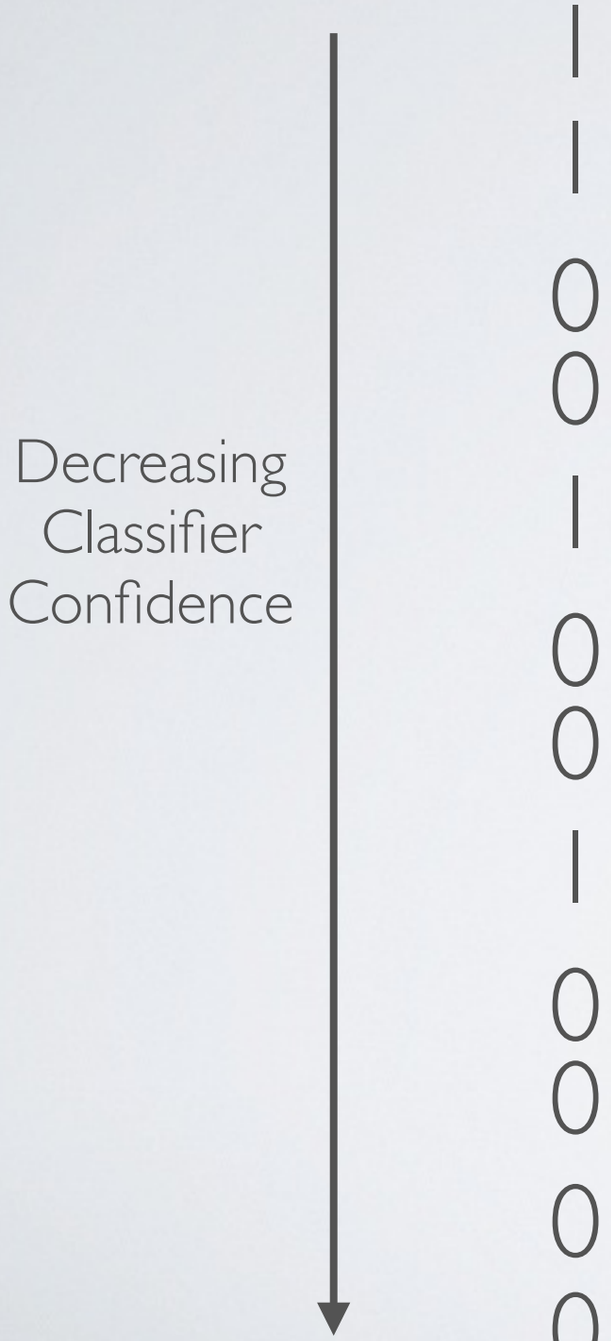
- Pros:
  - No need to arbitrarily decide k

- Cons:
  - Results still depend on the fraction of real positive in the test set:
    - The more positive, the easier it is to have a good score
    - Imagine 90% of class 1 : random order => value of 0.9
    - If 10% of class 1, random order => value of 0.1

AUC - AUROC

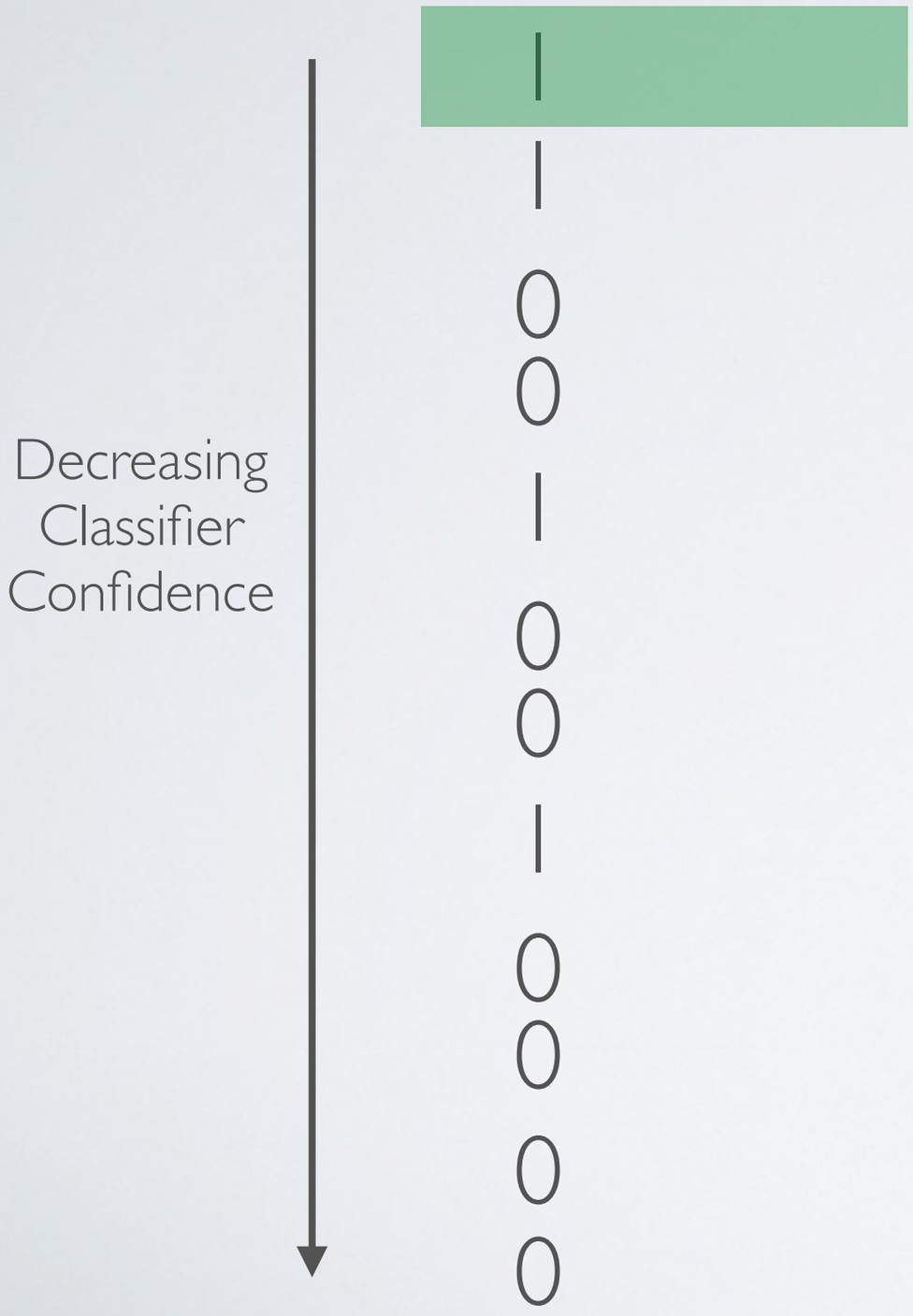
- AUC: Area Under the Curve. Short name for AUROC (Area under the Receiver Operating Characteristic Curve)
- Similar idea than AP, but analyzing the relationship between
  - True positives rate (recall): $TPR = \frac{TP}{TP + FN} = Recall$
    - Among all really positives, those we labelled correctly
  - False positives rate :$FPR = \frac{FP}{FP + TN}$
    - Among all really negatives, fraction we mislabelled.

AUC



1
1
0
0
1
0
0
1
0
0
0
0

AUC



Green is predicted at 1, rest at 0
(Confidence threshold)

TP=1 FP=0

$$TPR=\frac{1}{4}, FPR=\frac{0}{8}$$

AUC

Top2

TP=2 FP=0

$$TPR=\frac{2}{4}, FPR=\frac{0}{8}$$

AUC

Top3

TP=2 FP=1

$$TPR=\frac{2}{4}, FPR=\frac{1}{8}$$

AUC

Top5

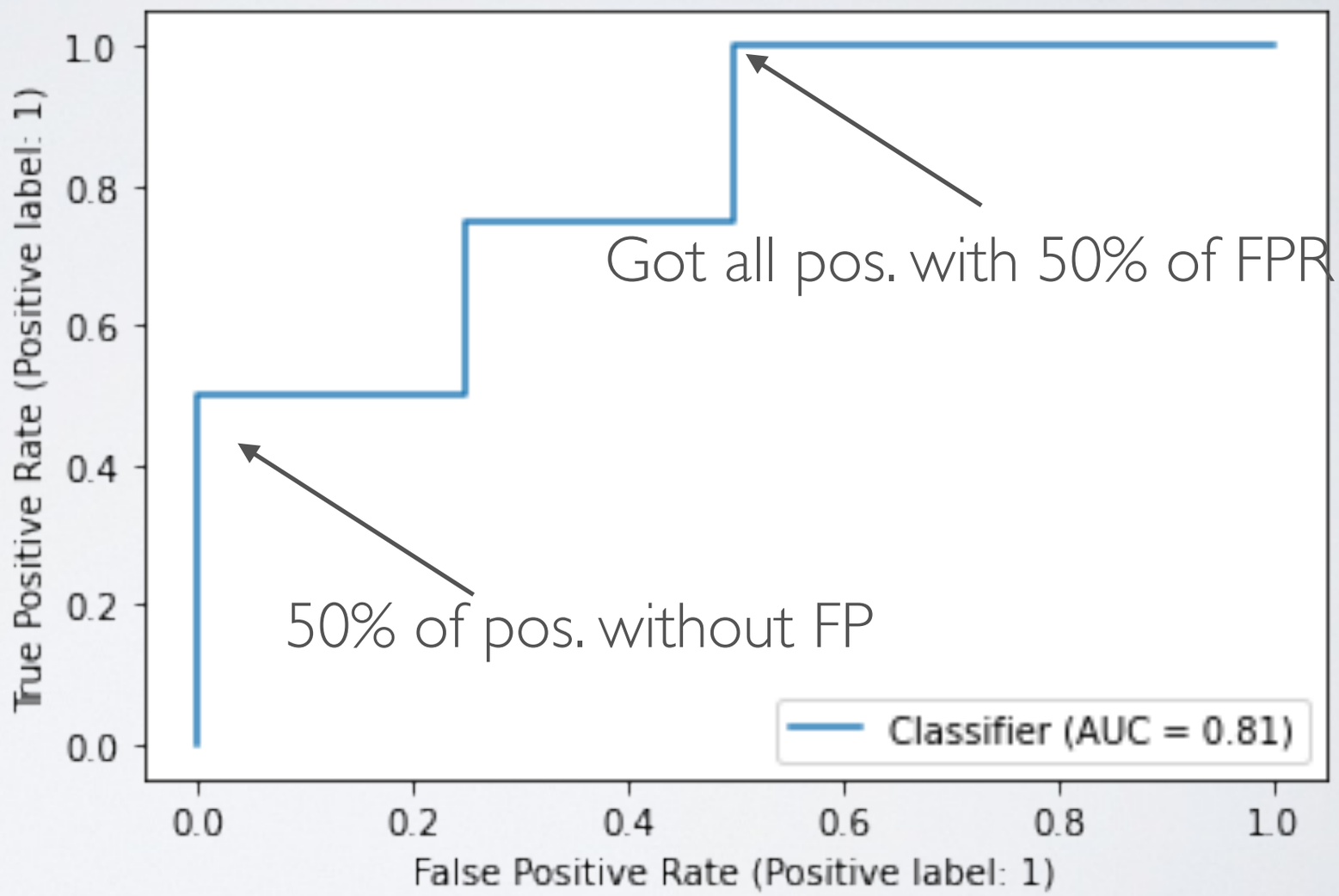
TP=3 FP=2

$$TPR=\frac{3}{4}, FPR=\frac{2}{8}$$

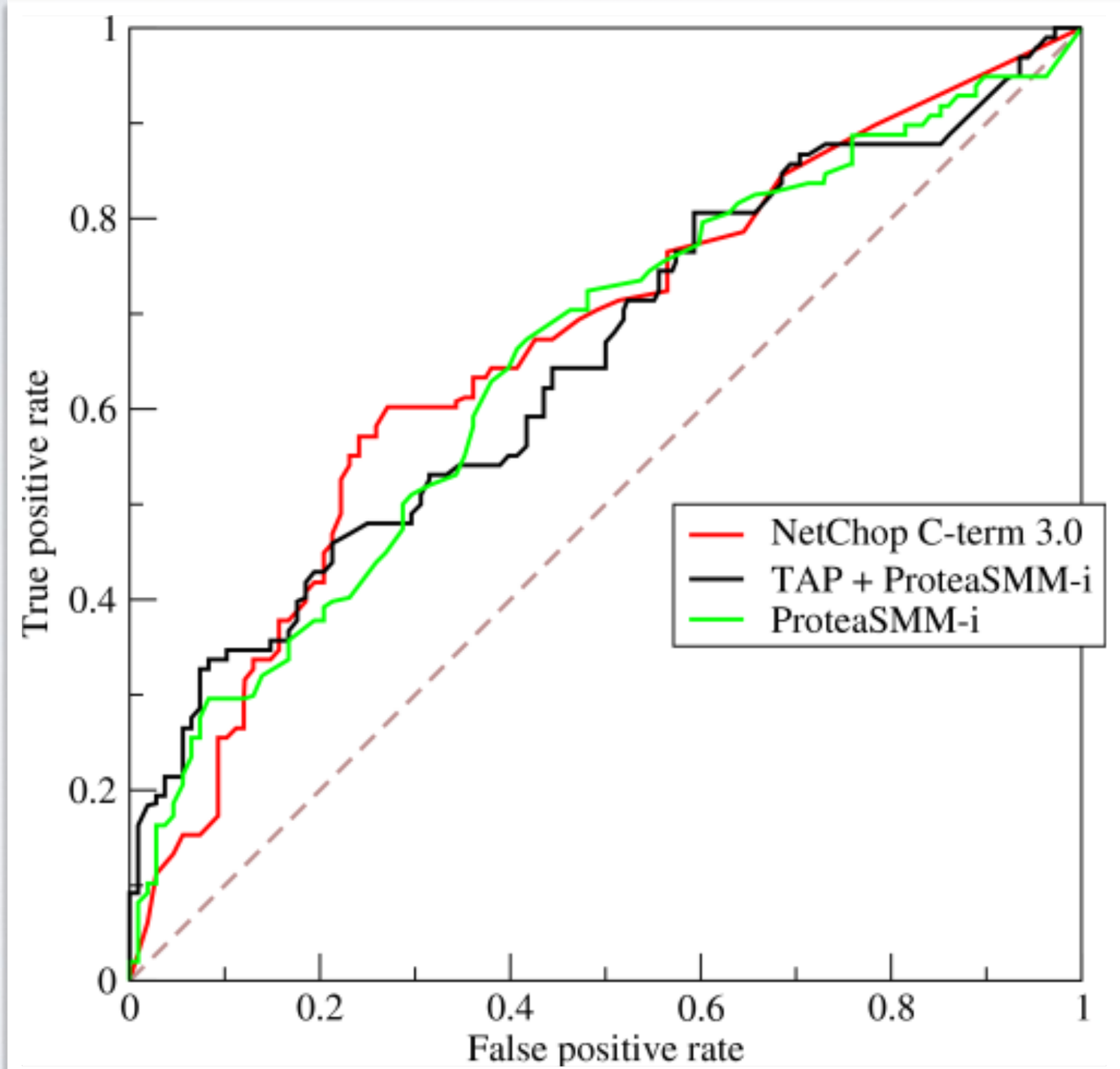
AUC

Decreasing Classifier Confidence

1
1
0
0
1
0
0
1
0
0
0
0



AUC - AUROC


True positive rate
False positive rate
0
0.2
0.4
0.6
0.8
1
NetChop C-term 3.0
TAP + ProteaSMM-i
ProteaSMM-i

AUC - AUROC

- Probabilistic interpretation:
  - If we pick a random positive example and a random negative example, probability that the positive one has a higher score

- Pros:
  - Independent on the fraction of positive examples, i.e., an unbalanced test set can be used
    - If at random we got 30% of all positives, we have also 30% of all negatives

- Cons:
  - Often high values, (>0.95), thus small (relative) improvements

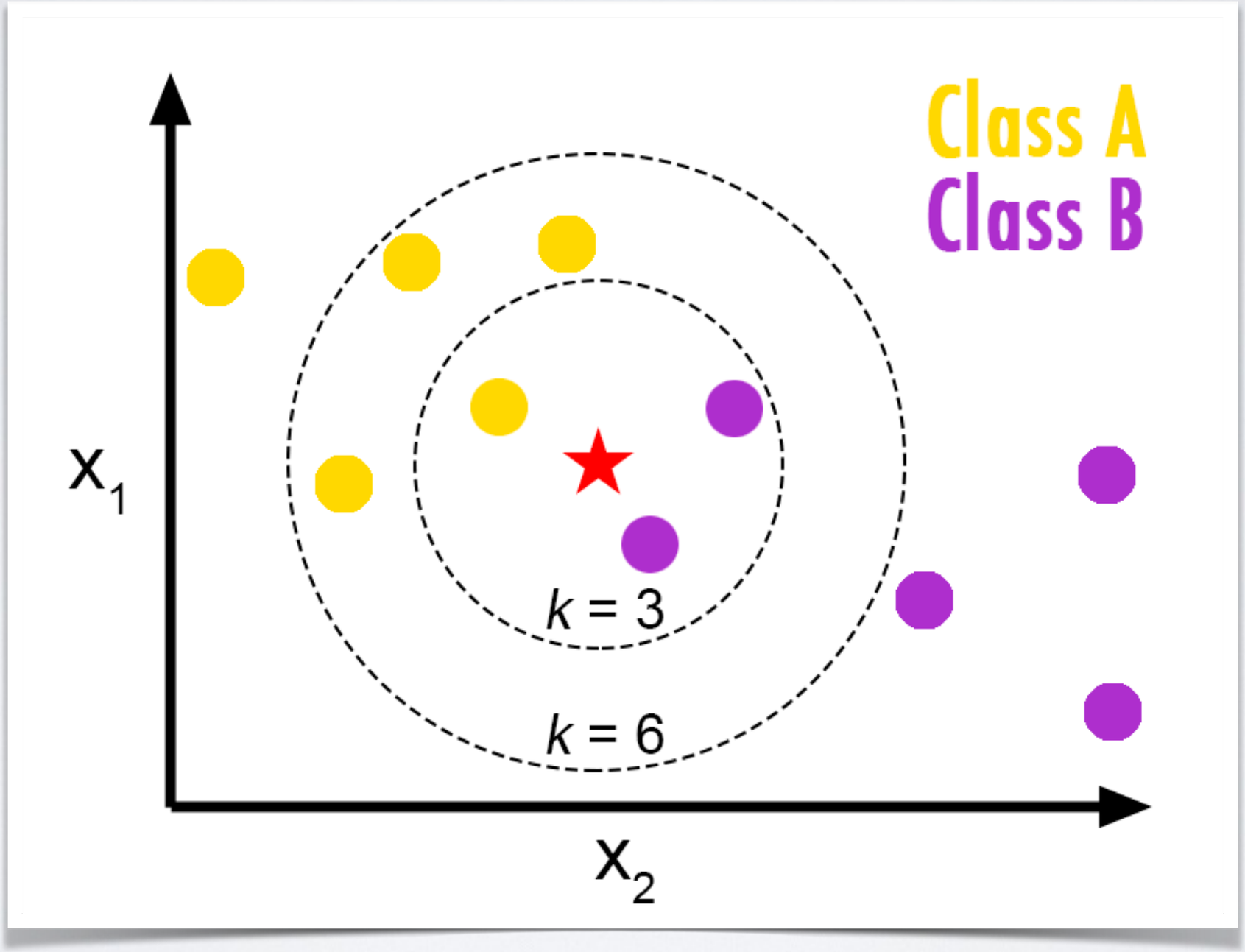
KNN

K nearest neighbors

K-NN

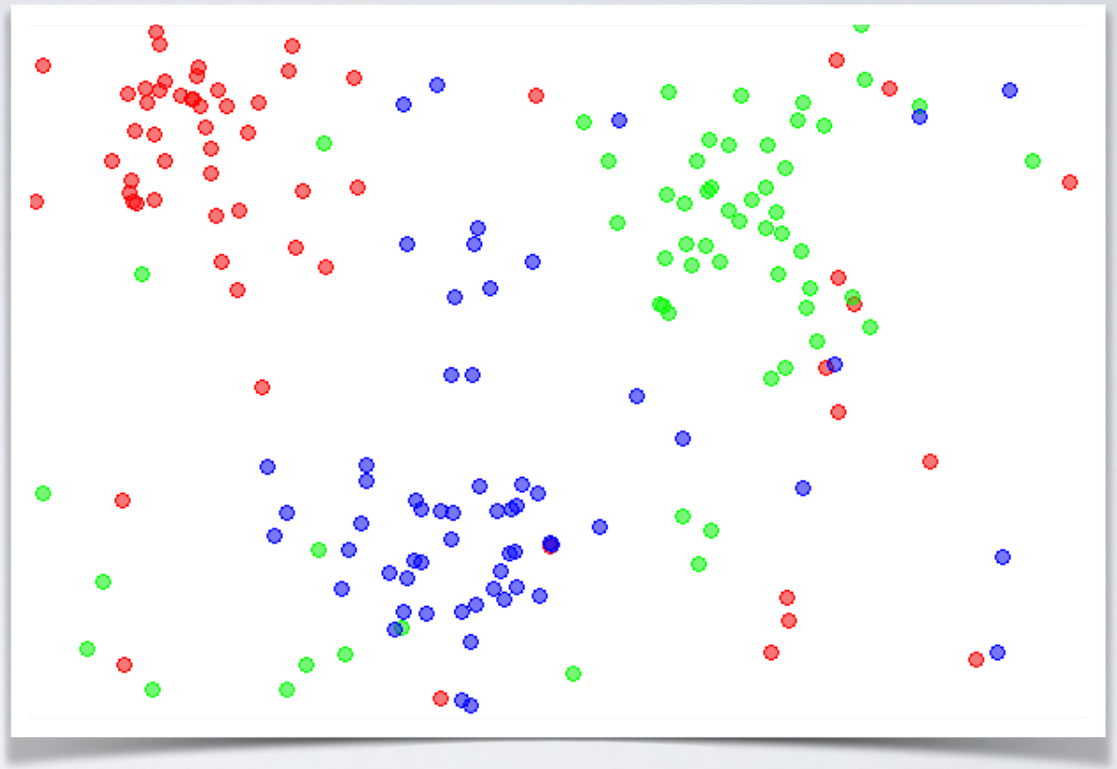
- Extremely simple approach, yet very powerfull in certain cases
- Principle: to classify (or regress) a new observation, we search for the closest one(s) in the training set, and assign the same class/value average.
  - K is obviously a parameter

K-NN

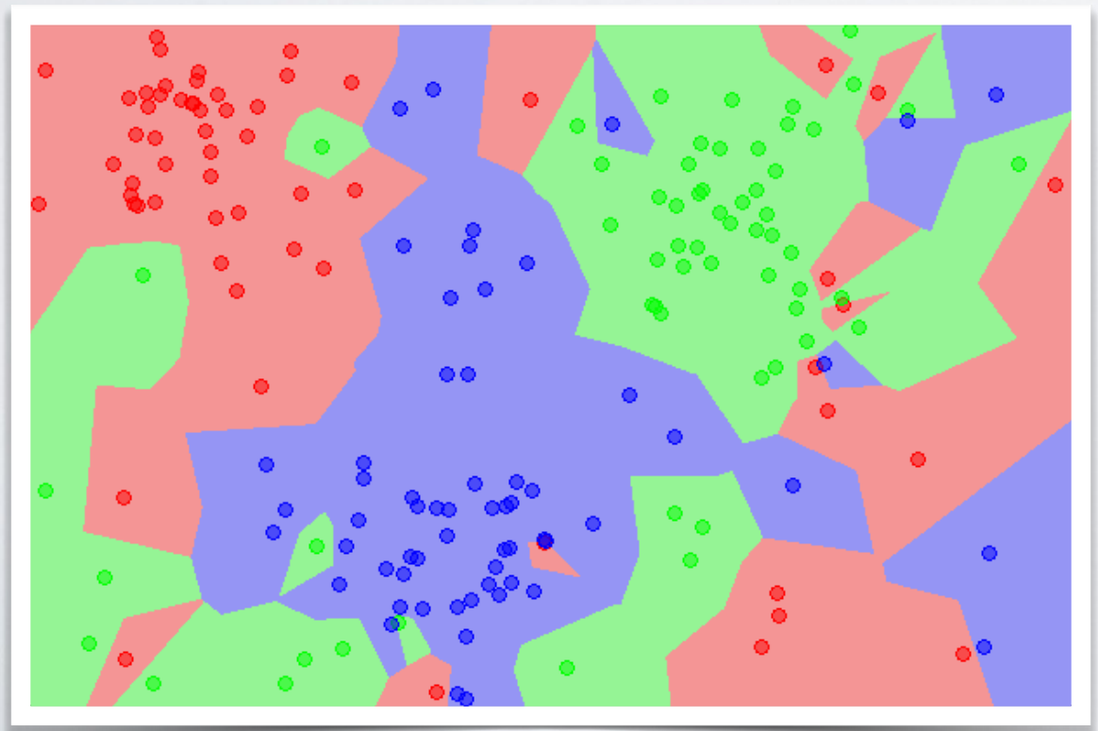


https://helloacm.com/a-short-introduction-to-k-nearest-neighbors-algorithm/

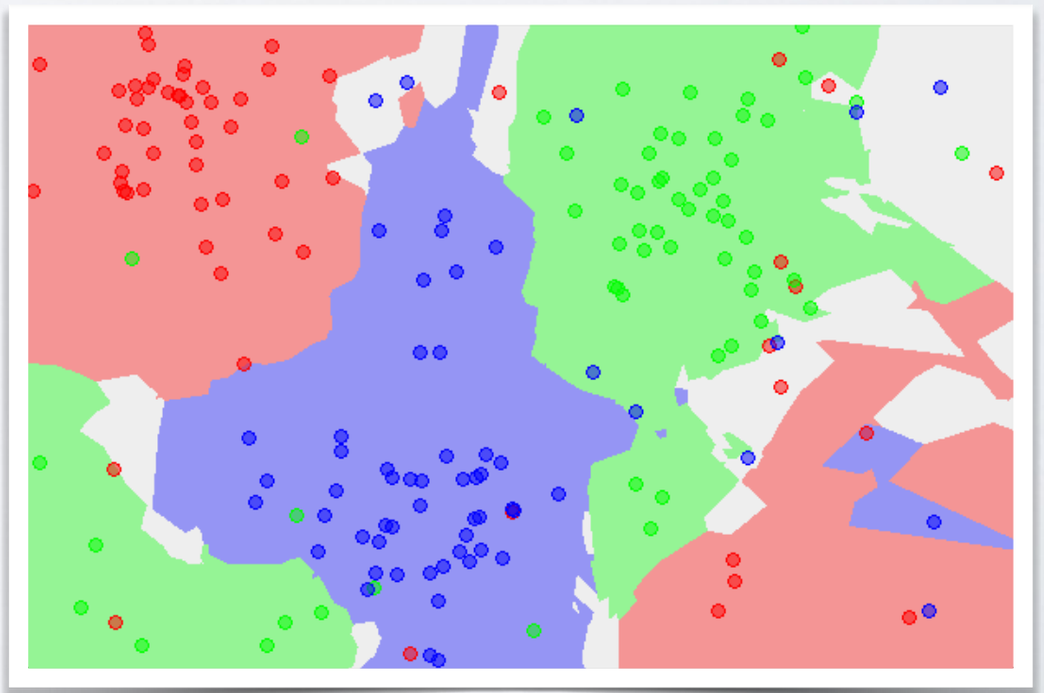
Dataset (2D, 3 classes)



1-NN



5-NN



K-NN

- Strength
  - Extremely efficient with large training set and good covering of the feature space
    - Shown to outperforms more advanced methods in many applications
  - Few parameters, simple to understand
  - No training time (possible precomputation)

- Weaknesses
  - Finding neighbors is done at evaluation time, which can be a problem with large datasets
    - Solutions: K-D tree, Ball tree… but keep dataset in memory. Hashing…
  - Curse of dimensionality=>dimensionality reduction first.
  - Choice of a proper distance

NAIVE BAYES

NAIVE BAYES

- As KNN, a simple yet powerful approach for classification
- Imagine you want to classify fruits/vegatables
  - You have features: color, height, sweetness, weight…
  - Make the naive assumptions that variables are uncorrelated
  - You want to know if product X,{red, 10cm,sweet,200g} is of class: Tomato. Compute independently:
    - If a product is a tomato, probability to be red?=> among tomatoes, fraction of red
    - If a product is a tomato, probability to be 10cm?=> among tomatoes, fraction of 10cm
    - …
  - Combine all independent predictions as a product of probabilities.

NAIVE BAYES

- General case: $\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,.$
  - K are possible classes to predict
  - $p(C_k)$ probability to observe class k
  - $p(x_i \mid C_k)$: Probability to have property x if we have class k

NAIVE BAYES

- Why Bayes?
  - Solution comes from Bayes theorem

- We want to find: $p(C_k \mid \mathbf{x})$
  - $\mathbf{x}$: vector of observed features for an item

- Bayes theorem: $p(C_k \mid \mathbf{x}) = \dfrac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$
  - posterior $= \dfrac{\text{prior} \times \text{likelihood}}{\text{evidence}}$

NAIVE BAYES

- $p(x_1, \ldots, x_n \mid C_k) = \prod_{i=1}^{n} p(x_i \mid C_k)$

- In $p(C_k \mid \mathbf{x}) = \dfrac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$
  - $p(x)$ is a constant and can be ignored

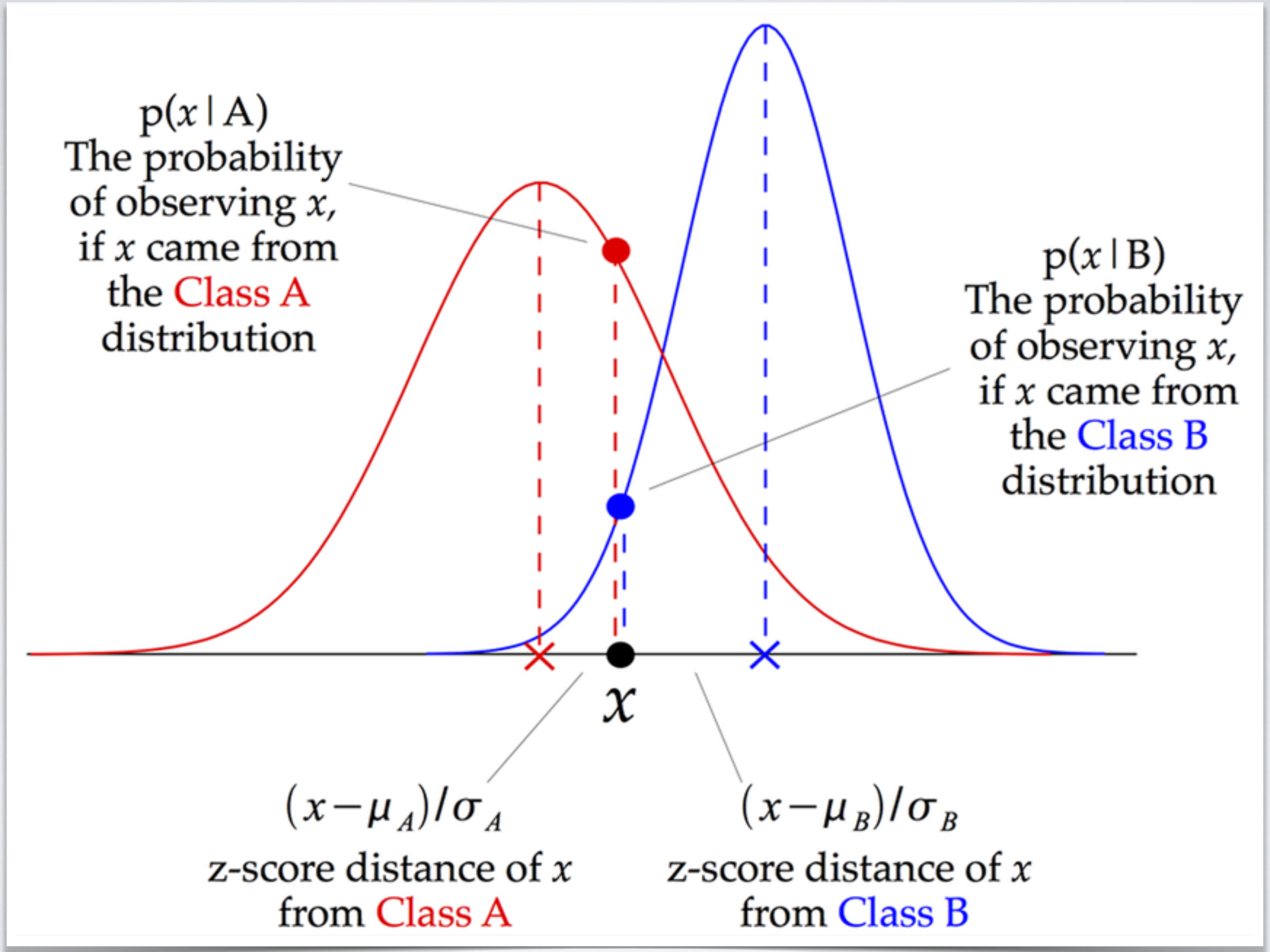
- Thus => $\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,.$

NAIVE BAYES

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,.$$

- How to compute $p(x_i \mid C_k)$ with continuous features?

- Make an assumption about the variable distribution
  - Typically, normal distribution.
  - $$p(x = v \mid C_k) = p(x = v \mid N(\mu_k, \sigma_k^2)) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$


p(x | A)
The probability of observing x, if x came from the Class A distribution
p(x | B)
The probability of observing x, if x came from the Class B distribution
x
(x − μ_A)/σ_A
z-score distance of x from Class A
(x − μ_B)/σ_B
z-score distance of x from Class B

ML ADVANCED

ML: OLD-STYLE VS NEW GENERATION

- Old style machine learning:
  - 1)Define a set of intuitive rules to solve a problem
    - Not necessarily with mathematical justification (e.g., decision tree, KNN, DBSCAN…)
  - 2) Well defined mathematical objective without realistic assumption
    - Linear regression…

- More recent trend:
  - Design methods with 1)clearly defined objectives and 2)good properties:
    - Expressive enough to go beyond underfit and allow non-linearity
    - Integrate protections agains overfit

MACHINE LEARNING: GENERAL FORMALIZATION

MACHINE LEARNING

- Start by defining the objective:
- Loss functions
  - MSE, L2 loss : $L(y_i, \hat{y}_i) = \sum_{i}^{N} (y_i - \hat{y}_i)^2$
  - MAE, L1 loss $L(y_i, \hat{y}_i) = \sum_{i}^{N} |y_i - \hat{y}_i|$
  - K-means loss : $L(S, \mathbf{x}) = \sum_{j=1}^{k} \sum_{\mathbf{x} \in S_j} (\mathbf{x} - m_j)^2$
  - Softmax logistic Regression Loss : $L(y_i, \hat{y}_i) = \sum_{i=1}^{N} y_i \log(\hat{y}_i)$
  - …

MACHINE LEARNING

- Once we have define our loss function, the ML task to solve can simply be expressed as minimizing it over some parameters:
  - E.g., for Linear Regression:
    - Detailed way: $\arg\min_{\beta_0,\beta_1\ldots,\beta_n} \sum_i^n (y^i - (\beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \ldots + \beta_n x_n^{(i)}))^2$
    - Simplified way: $\arg\min_{\beta} \sum_i^n (y^i - f(\beta, x))^2$
    - Generic way: $\arg\min_{\theta} L_2(f(\theta, x))$