

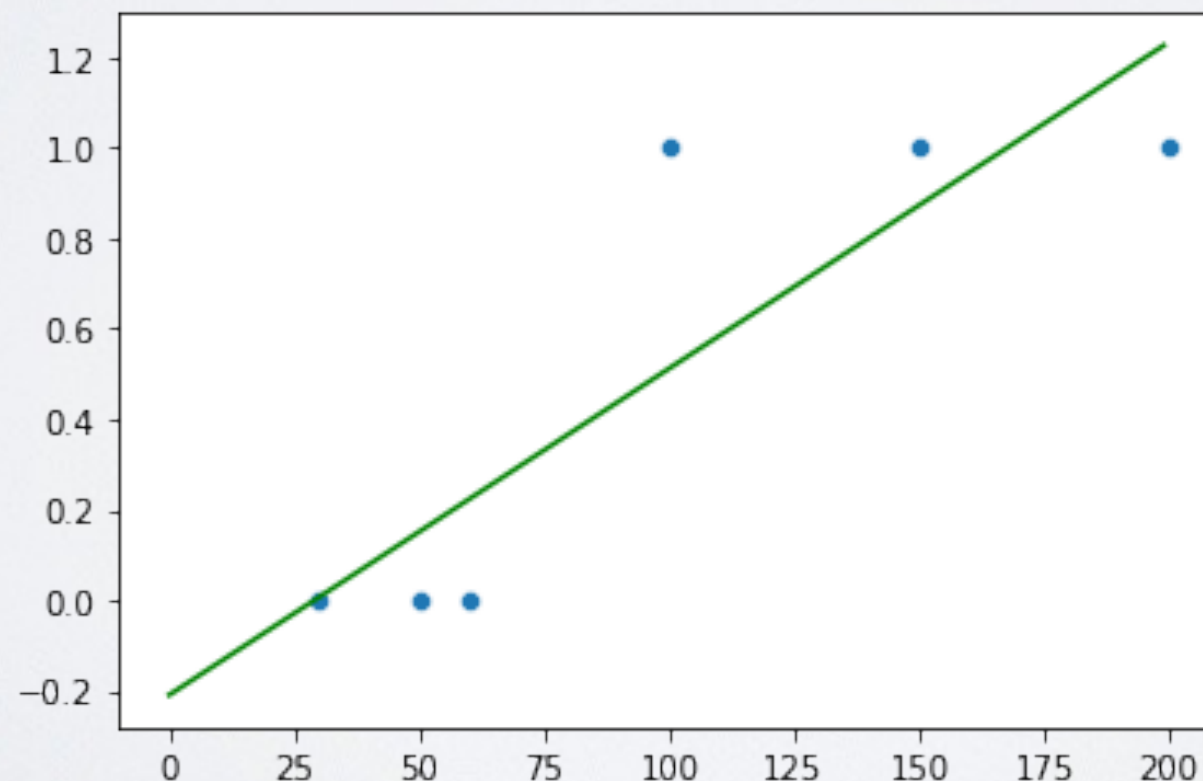
CLASSIFICATION

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

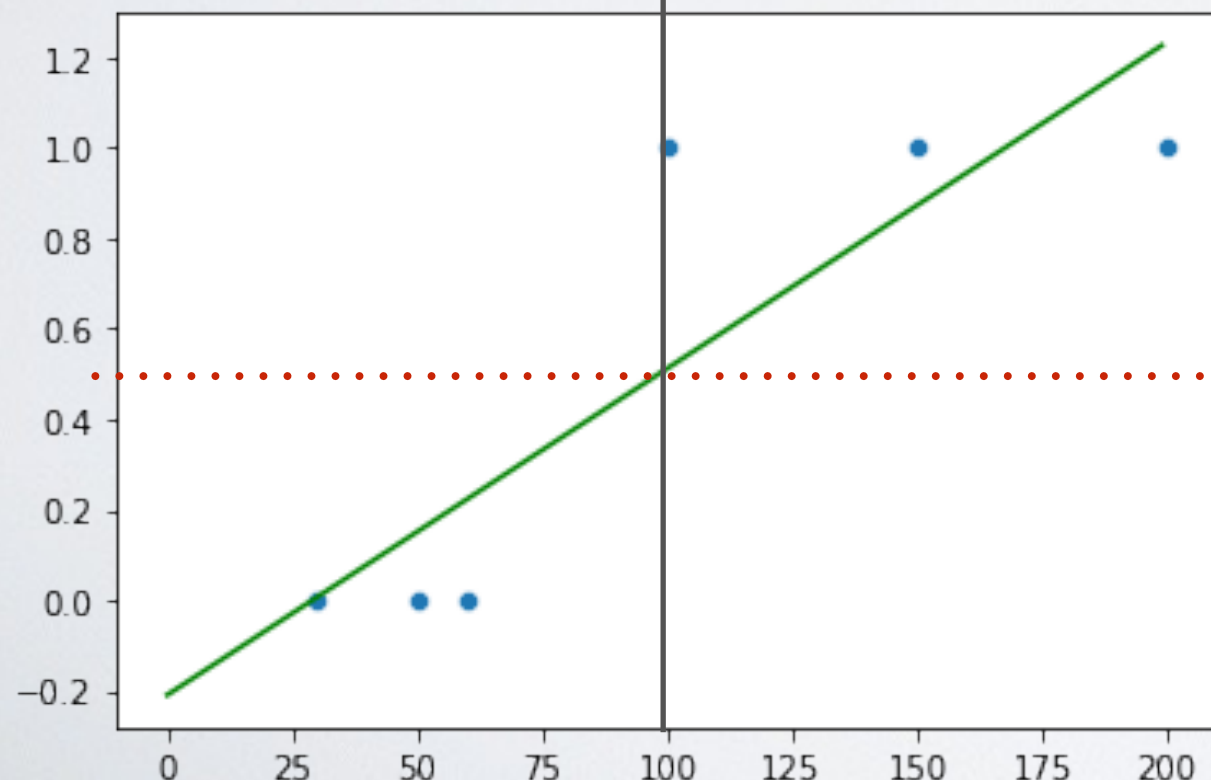
- We can easily adapt linear regression
- Imagine a 1 feature example:
 - We want to classify between apartments and houses
 - Our (unique) feature is surface



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 surface

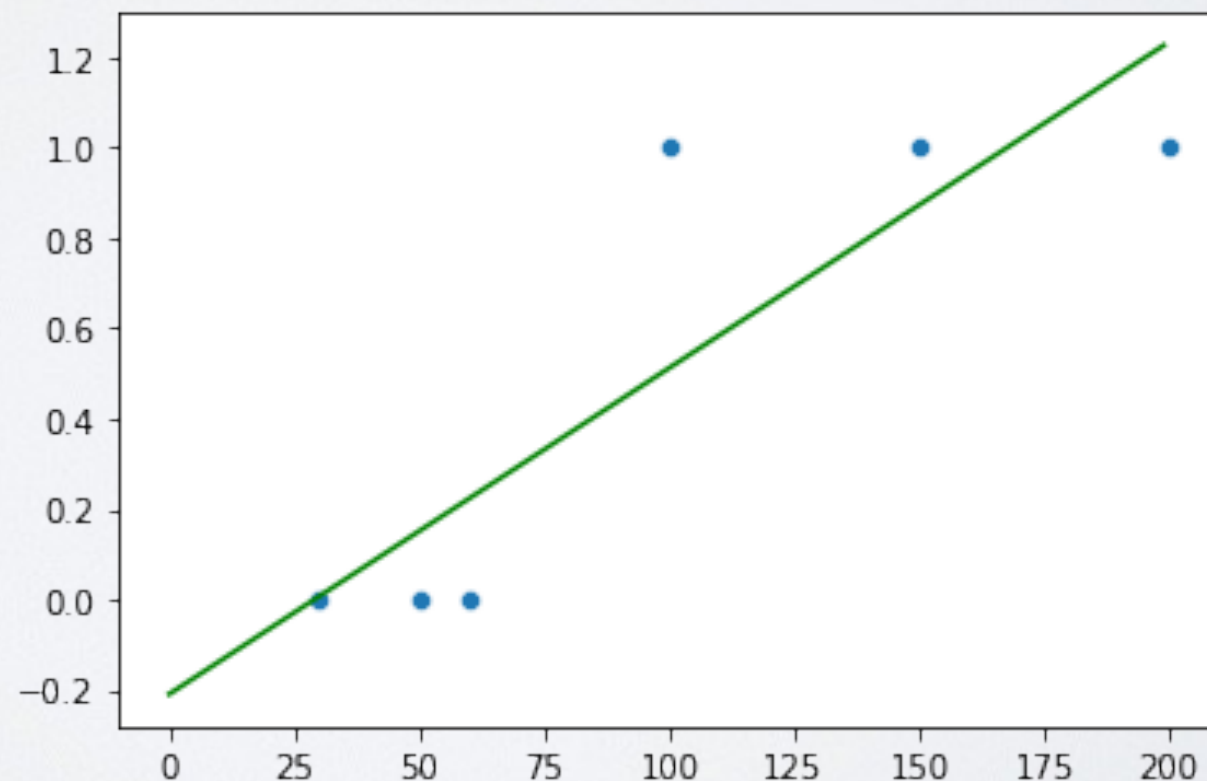
Classified as 0 | Classified as 1



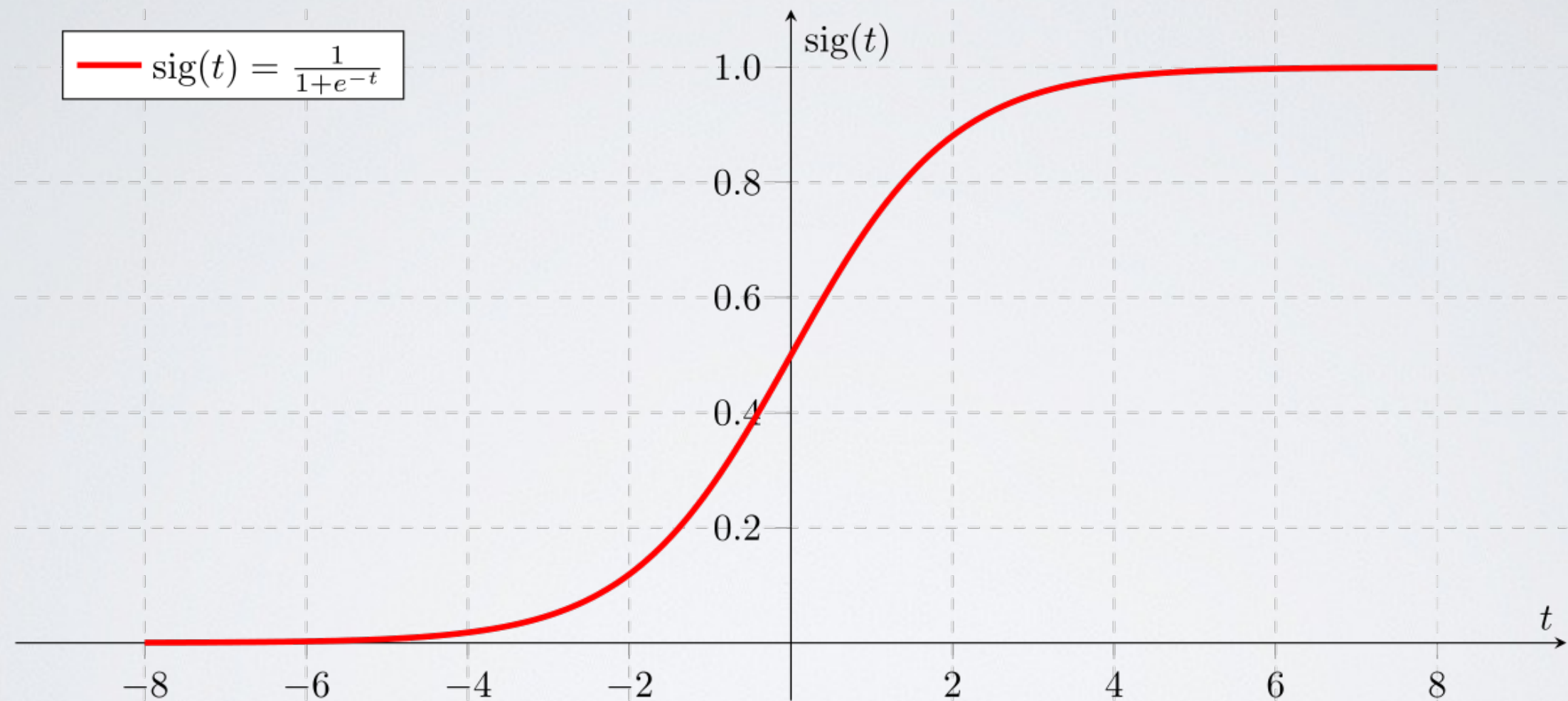
MSE 0.06361520558572538
RMSE 0.2522205494913636
MAE 0.20506852857512292
R2 0.7455391776570985

LINEAR CLASSIFICATION

- Problem: 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 ...)



SIGMOID/LOGISTIC FUNCTION



$$\lim_{t \rightarrow -\infty} \text{sig}(t) = 0$$

$$\lim_{t \rightarrow +\infty} \text{sig}(t) = 1$$

$$\text{sig}(0) = 0.5$$

LOGISTIC REGRESSION

Logistic (Sigmoid) function:

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

Linear regression:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

$$P(y = 1) = \text{Sig}(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)$$

Logistic regression:

$$P(y = 1) = \frac{1}{1 + e^{-\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n}}$$

LOGISTIC REGRESSION

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

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

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

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

LOGISTIC REGRESSION

$$\frac{P(y = 1)}{1 - P(y = 1)} = e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n}$$

Probability to happen / probability not happening
=> odds (FR: cote)

Get a 6 in a dice : odds = 1:5=0.2

Get a 5 or 6: odds = 2:4 =0.5

Get everything but a 1: odds = 5:1=5

From odds to proba: $p = \frac{odds}{1 + odds}$

LOGISTIC REGRESSION

/!\ multiplicative relation between variables

Interpretation as **odds ratios**:

+1 in $x_i \Rightarrow$ odds multiplied by e^{β_i}

$\Rightarrow$ Why odds? To force probabilities remaining under 1 ...

$$\frac{P(y = 1)}{1 - P(y = 1)} = e^{\beta_0} \cdot e^{\beta_1 x_1} \cdot e^{\beta_2 x_2} \cdot \dots \cdot e^{\beta_n x_n}$$

LOGISTIC REGRESSION

- Coefficients reading example
 - Binary variable=Value 0 or 1
 - Coefficient = 0.7. $e^{0.7} = 2.01 \Rightarrow$ Double odds ratios
 - Coefficient = -0.7. $e^{-0.7} = 0.5 \Rightarrow$ Divide odds ratios by 2
 - Probability change is not constant
 - Doubling a medium odd $\Rightarrow$ Large increase in probability
 - Doubling a large/small odd $\Rightarrow$ Small increase in probability
 - Initial proba: 0.9 $\Rightarrow$ odds: 0.9/0.1 $\Rightarrow$ 9
 - Odds X2 $\Rightarrow$ 18 $\Rightarrow$ $p(x) = 18/19=0.94$ (+0.04pts)
 - Initial proba: 0.5 $\Rightarrow$ odds: 0.5/0.5 $\Rightarrow$ 1
 - Odds X2 $\Rightarrow$ 2 $\Rightarrow$ $p(x) = 2/3=0.666$ (+0.16pts)

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: 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.
- 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).

CROSS ENTROPY

Or log-loss

- The usual loss function associated with softmax is the cross-entropy or log loss
 - $q(i)$: estimated probability.
 - $p(i) = 1$ for the true label (one-hot encoding,)
 - $\Rightarrow \log$ of error on true label.

$$H(P, Q) = - \sum_i p(i) \log q(i)$$

CROSS ENTROPY

Or log-loss

- Cross entropy name comes from information theory
 - Number of bits required to encode outcomes from the true distribution $p(y)$, using the predicted distribution $q(y)$ (using optimal coding)

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
- Loss:
 - Gini
 - Entropy

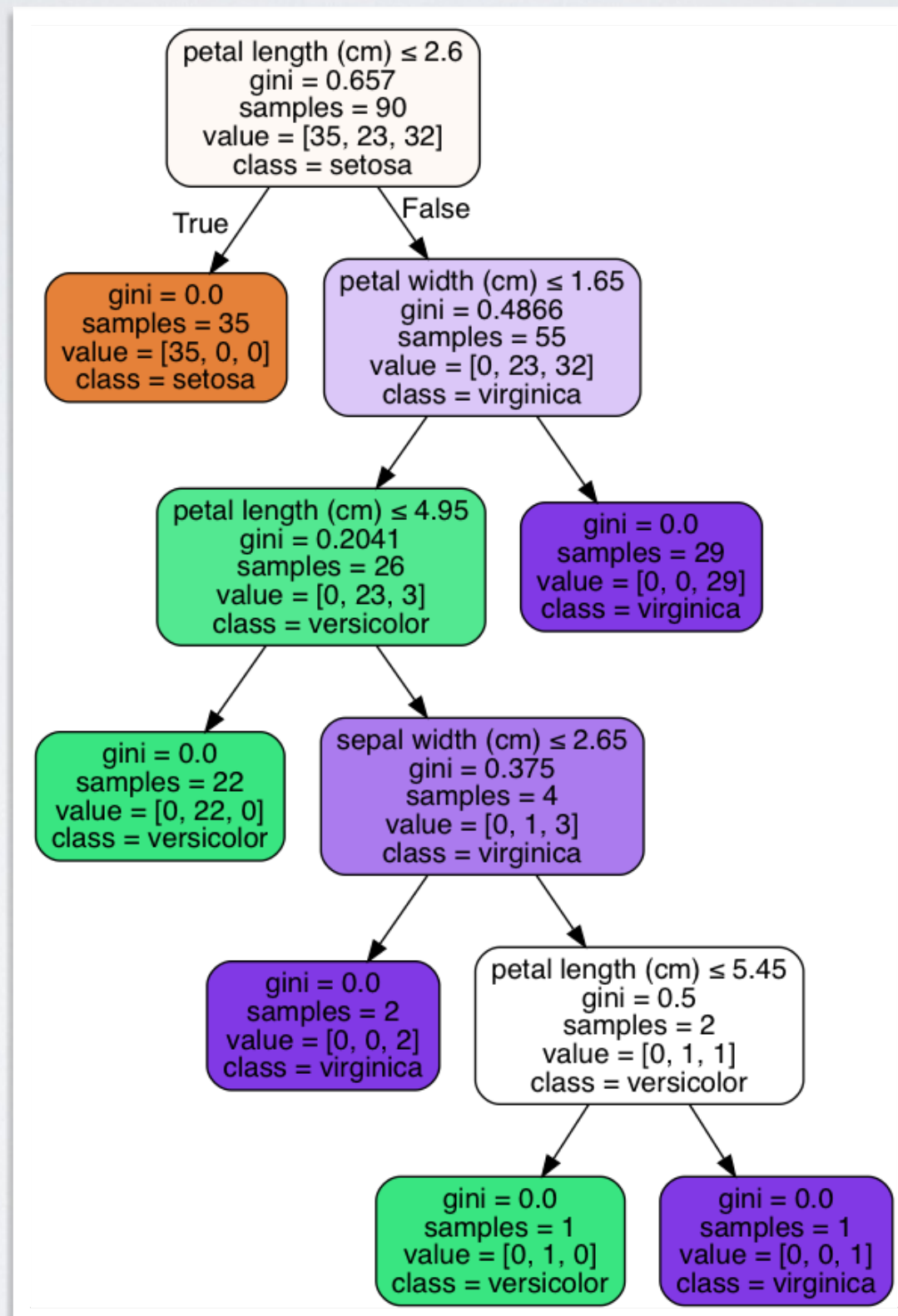
DECISION TREE

- Most common homogeneity/diversity/inequality/purity scores
 - Gini Coefficient: $1 - \sum_j p_j^2$
 - p_i : fraction of items of class i
 - Min: 0: 1 class only
 - Max: 0.5: (2 classes), 0.66(3classes), 0.75 (4classes), 0.875(8classes)
 - Interpretation:
 - If we classify a random item randomly according to class distribution, it is the 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, using optimal coding

DECISION TREE



BINARY CLASSIFICATION EVALUATION

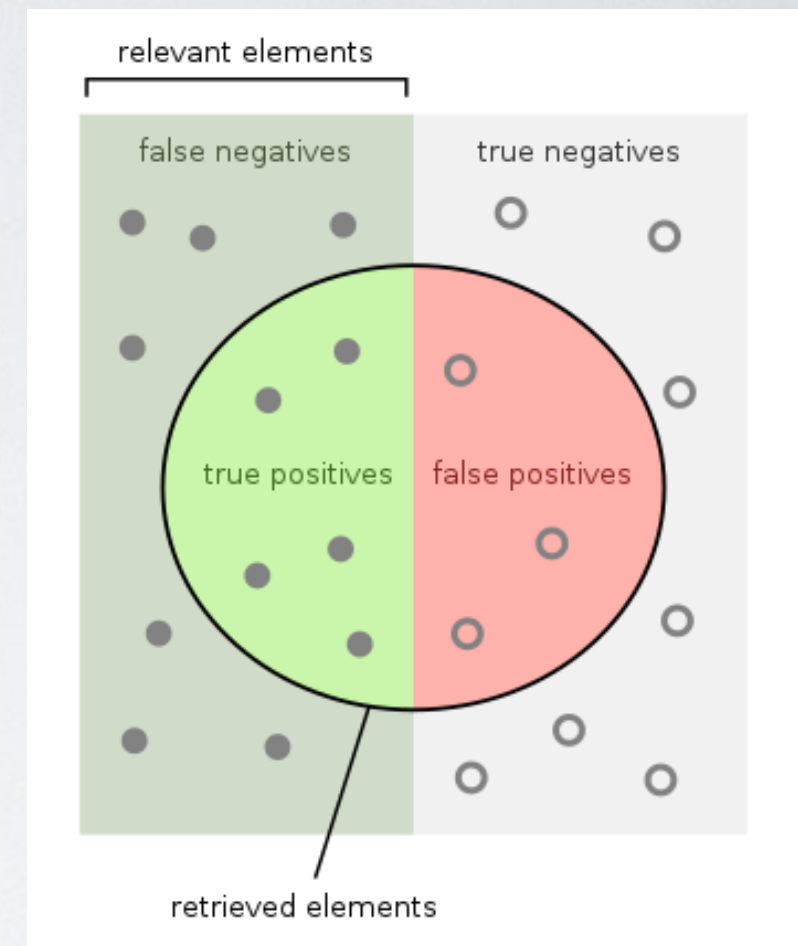
BINARY CLASSIFICATION

- Many scenarios require binary classification
 - Covid/not covid
 - Give a credit/do not give credit
 - Spam/not-spam
 - Postive sentiment/negative sentiment
 - Face on a photo/no face
 - Normal user/bot
 - Etc.

CLASSIFICATION: EVALUATION

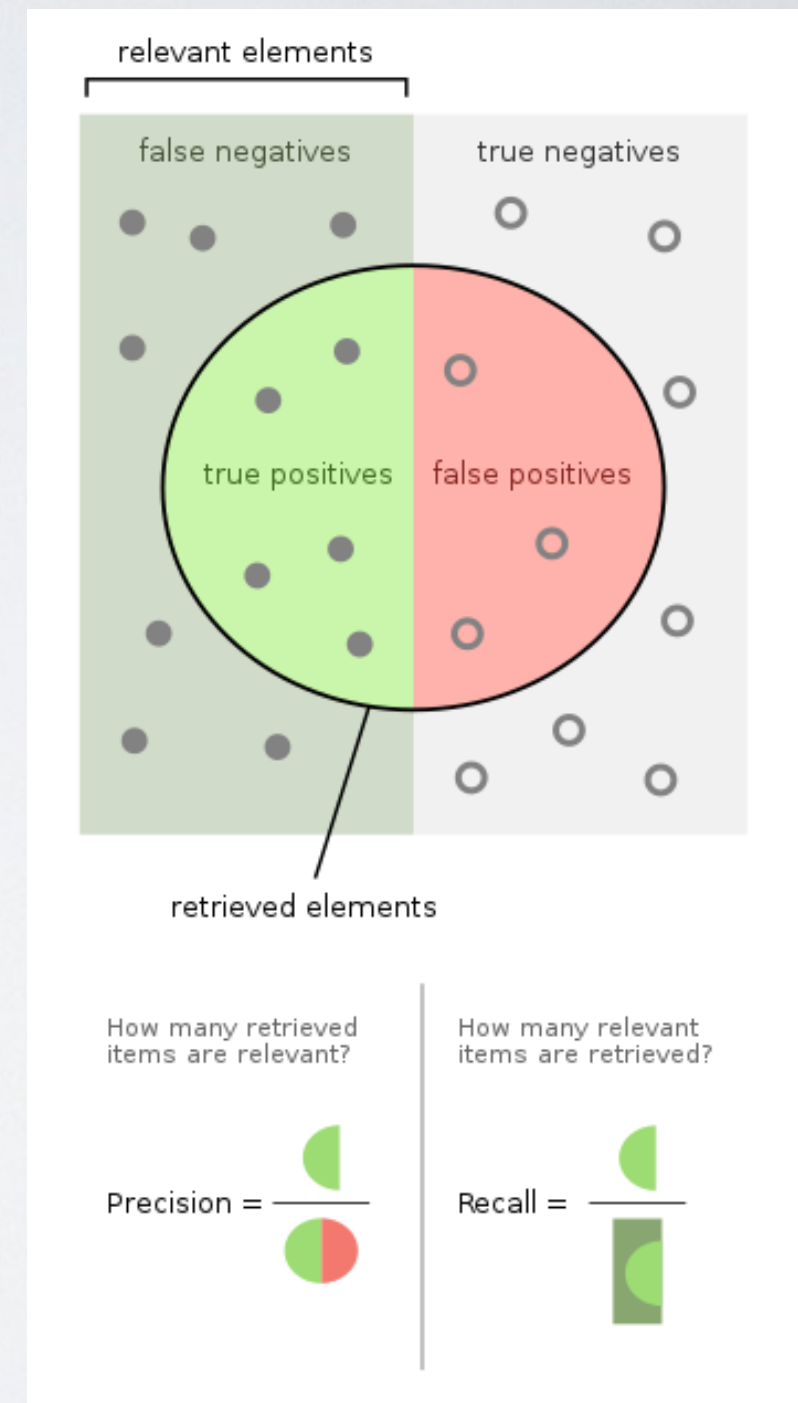
		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

/!\ 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 identify correctly
- Non-symmetric
 - Precision White $\neq$ Precision Black.
 - $\Rightarrow$ Easy to optimize one of the two, never trust one of them alone!



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: $F_1 = 2 \frac{\textit{precision} * \textit{recall}}{\textit{precision} + \textit{recall}}$
 - 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)
 - => F1 = 0.04

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, 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

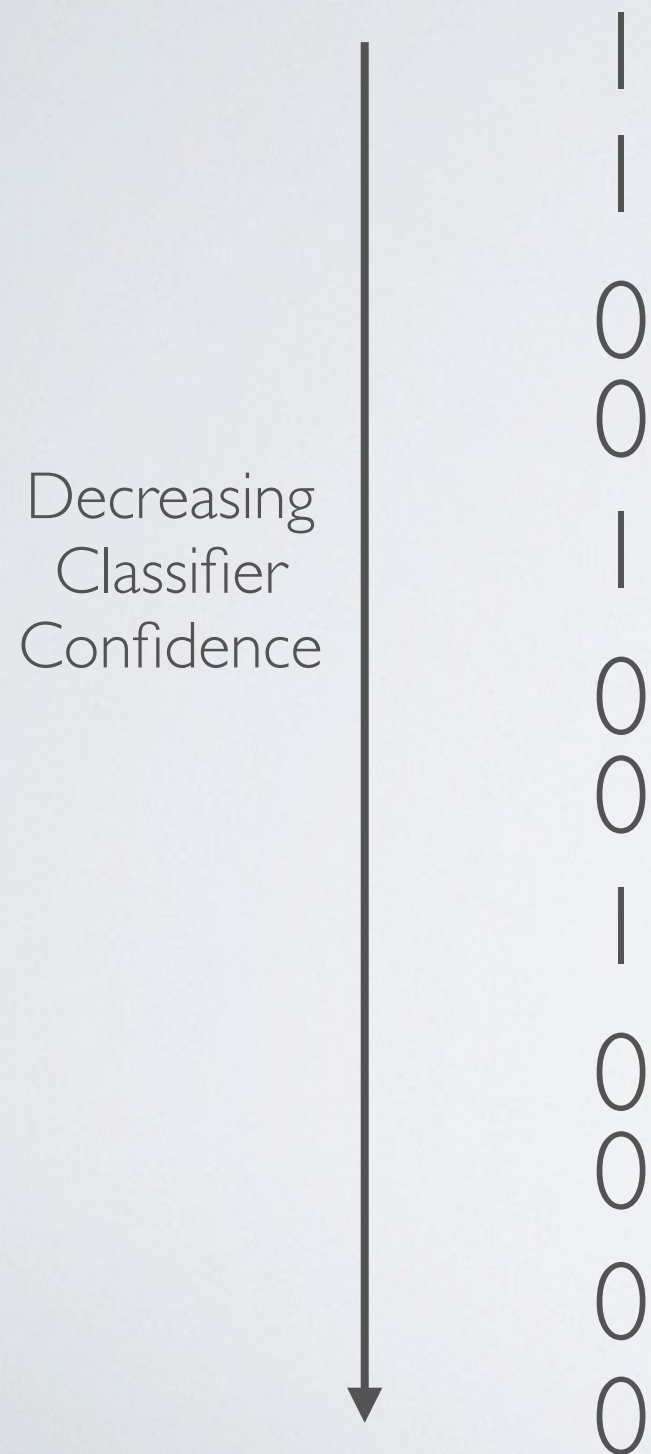
- If we don't know the exact k-value, but we know we care more about the first ones: Average Precision@k

- Compute the precision for each value of k, weighted by the gain in recall

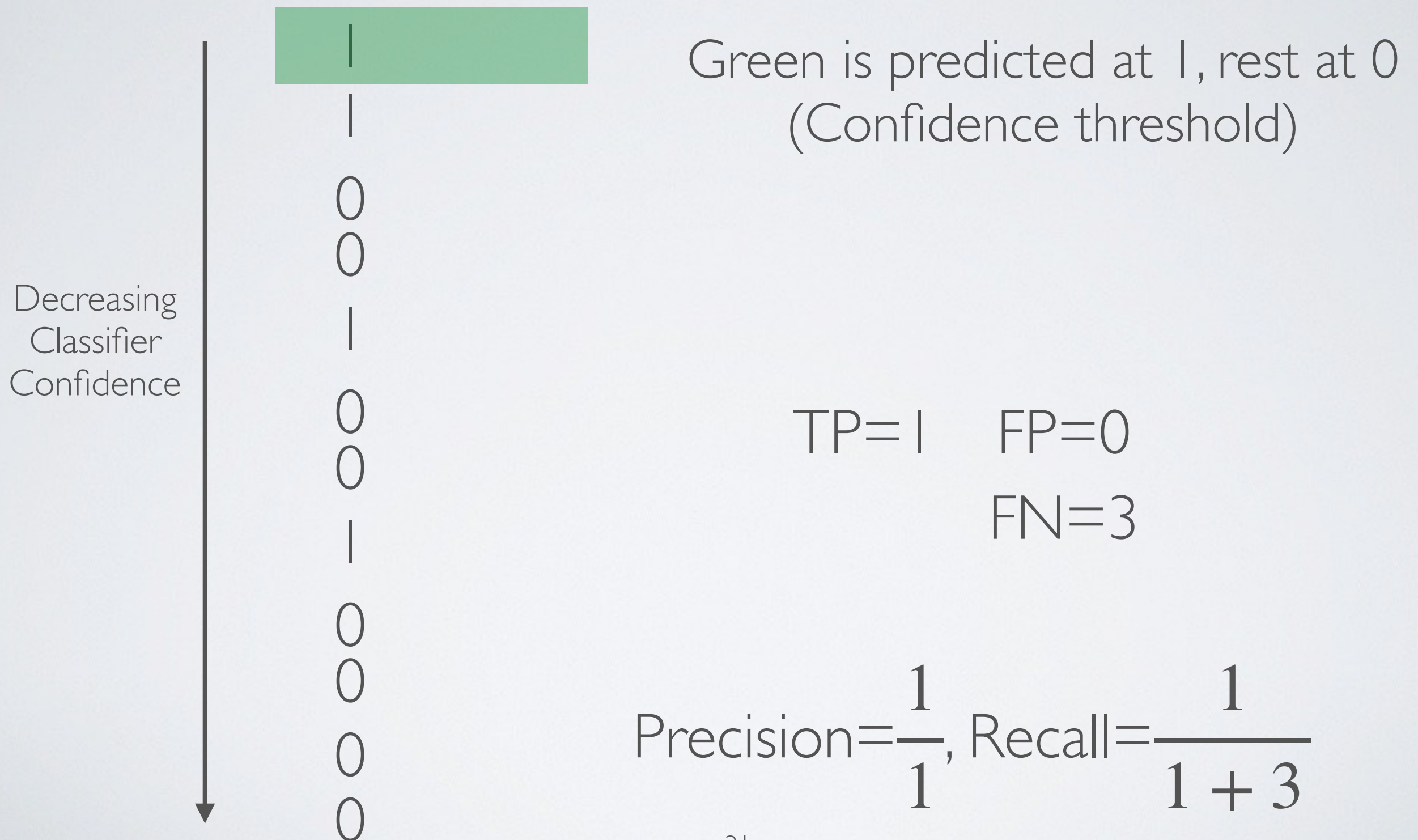
- $$\sum_i^n (R_i - R_{i-1}) P_i$$

- It can also be understood as the area under the Precision/Recall Curve

AVERAGE PRECISION



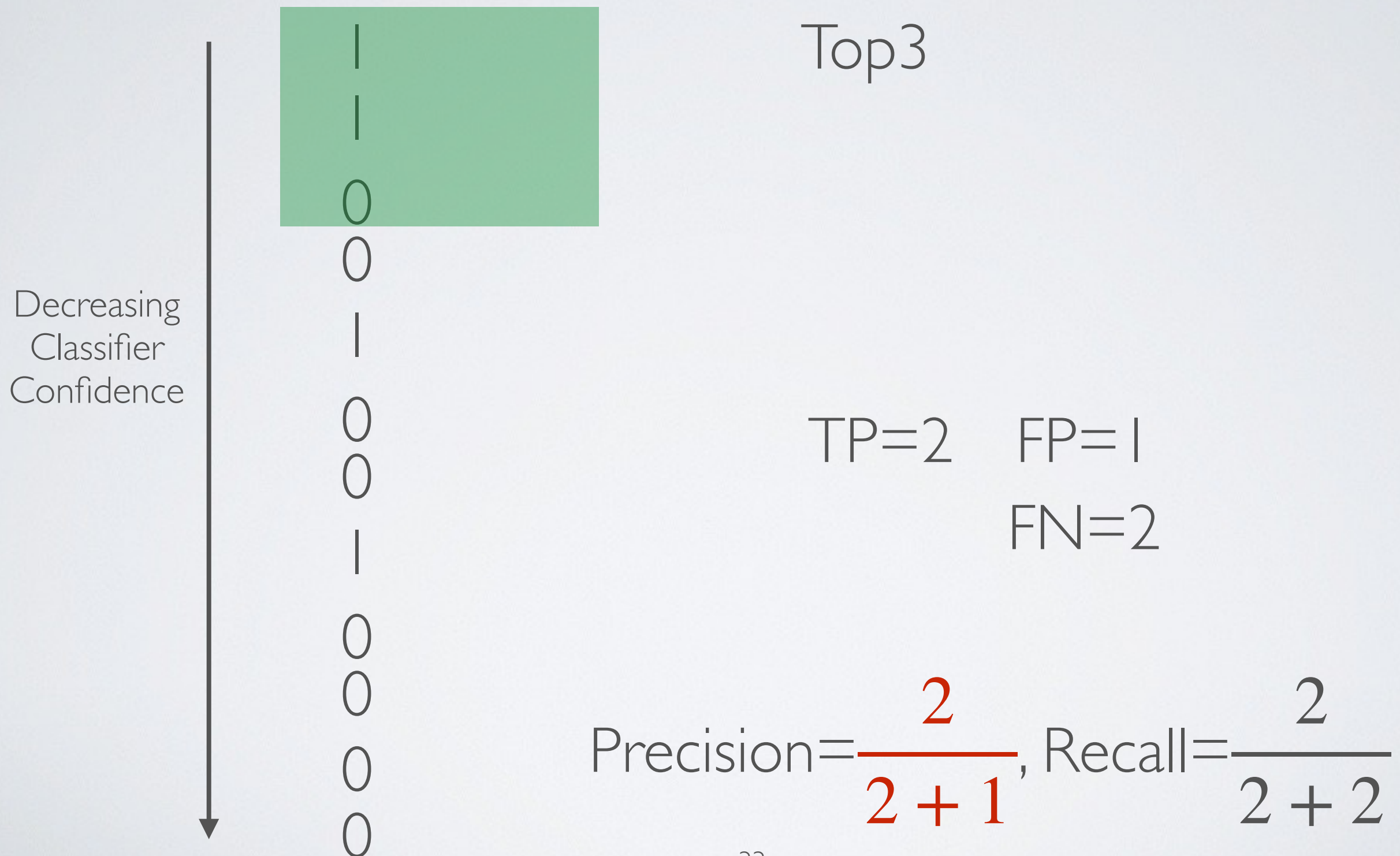
AVERAGE PRECISION



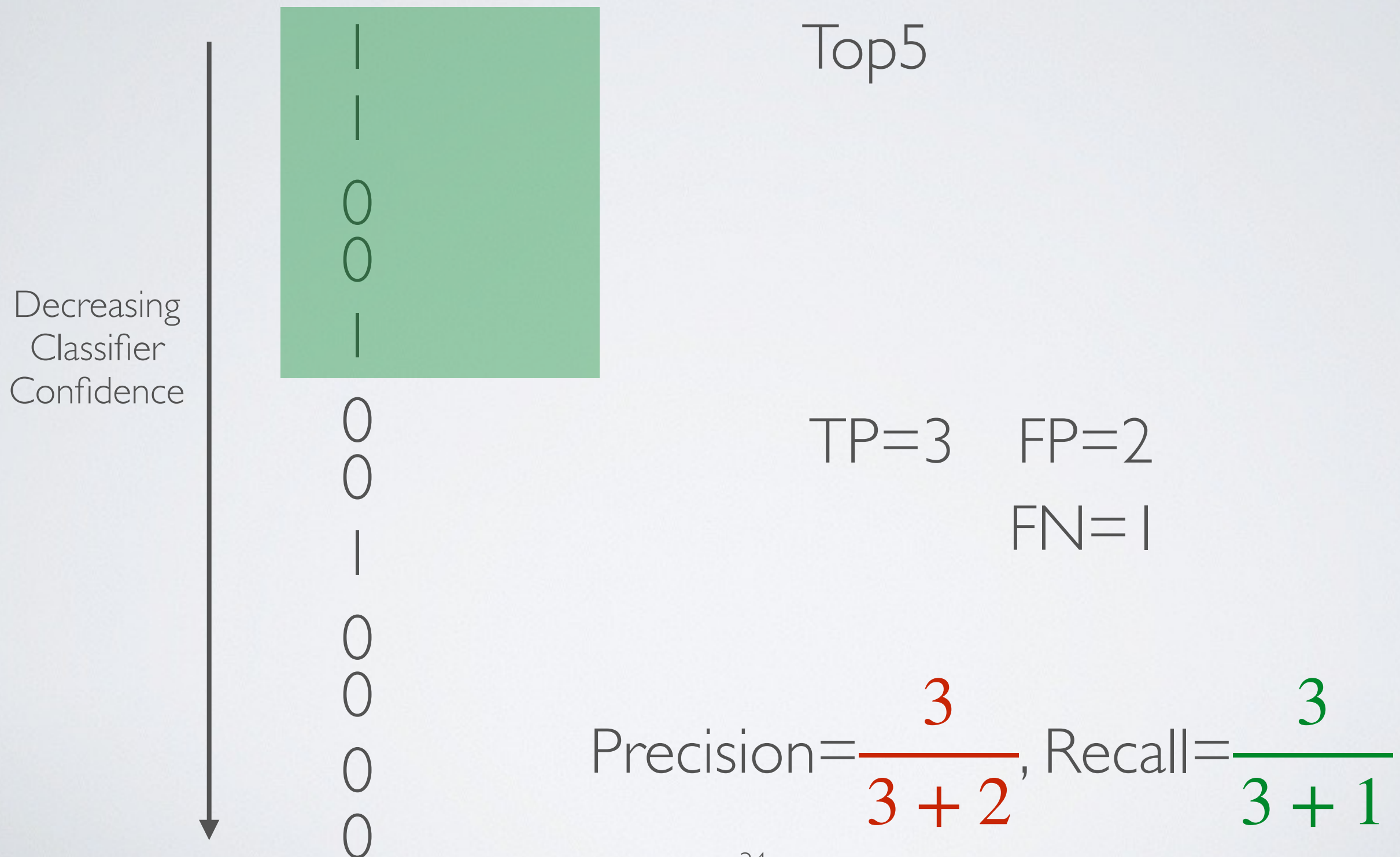
AVERAGE PRECISION



AVERAGE PRECISION



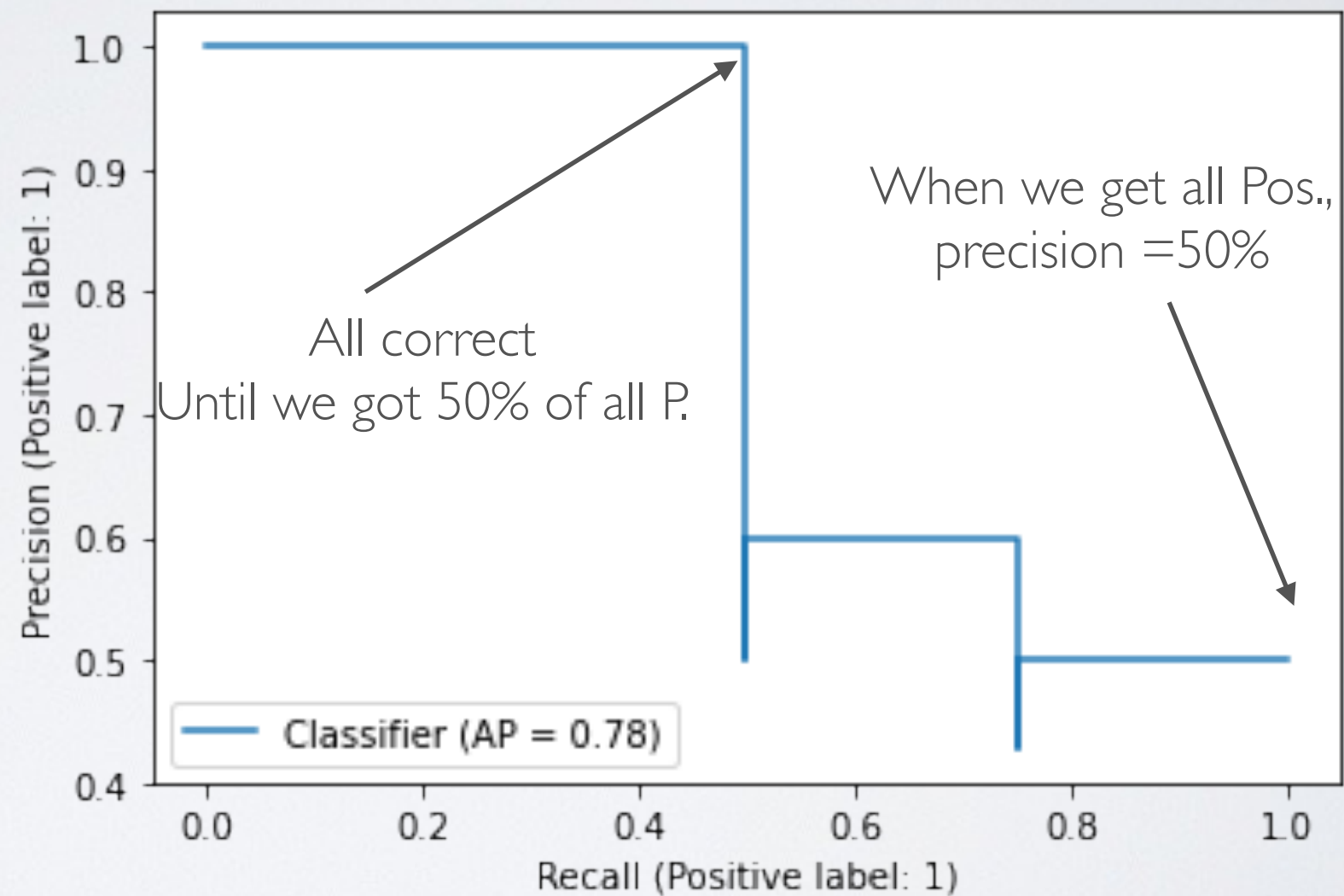
AVERAGE PRECISION



AVERAGE PRECISION

Decreasing
Classifier
Confidence

1
0
0
0
0
0
0
0
0
0
0



AVERAGE PRECISION

- 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 $\Rightarrow$ value of 0.9
 - If 10% of class 1, random order $\Rightarrow$ 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

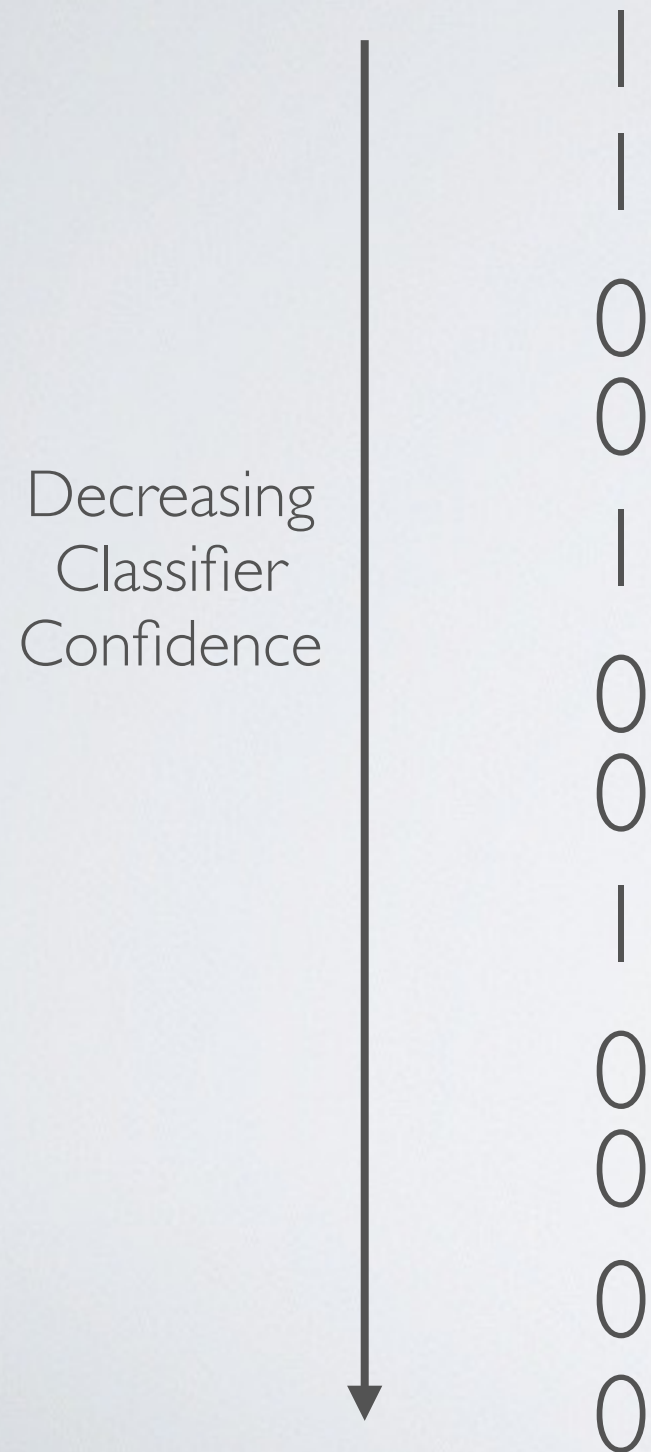
- True positives rate (recall): $TPR = \frac{TP}{TP + FN} = Recall$

- Among all really positives, those we labelled correctly

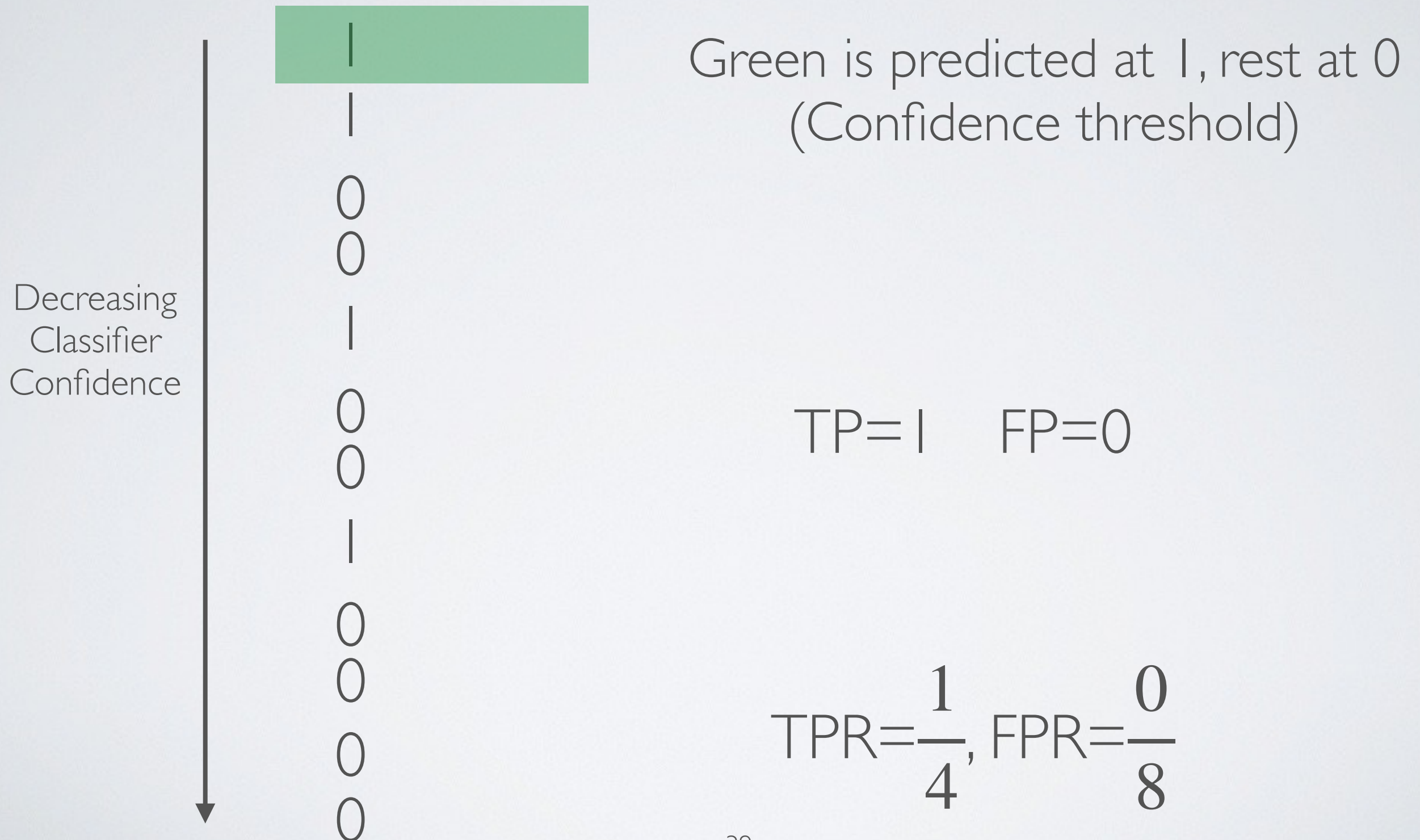
- False positives rate : $FPR = \frac{FP}{FP + TN}$

- Among all really negatives, fraction we mislabelled.

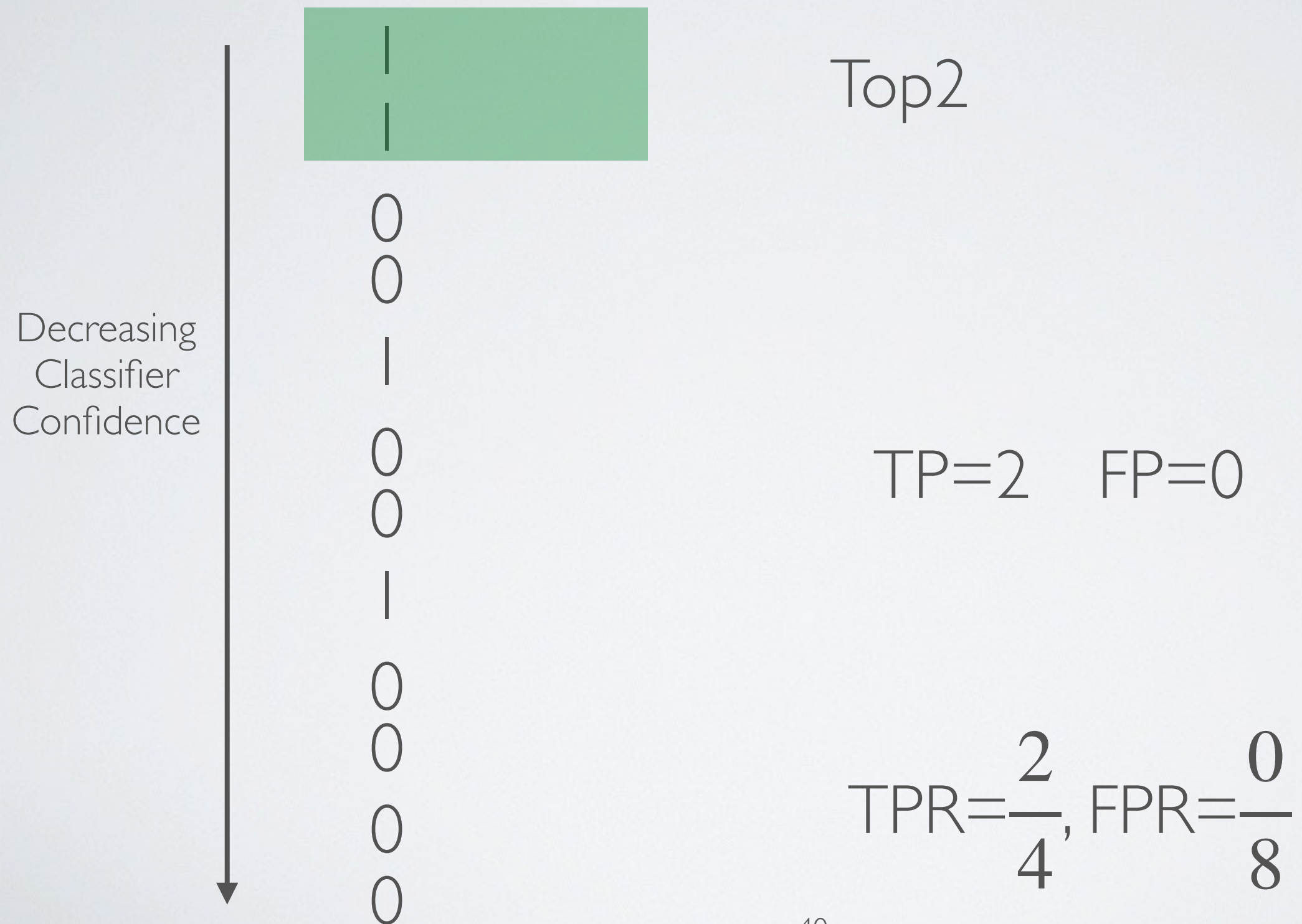
AUC



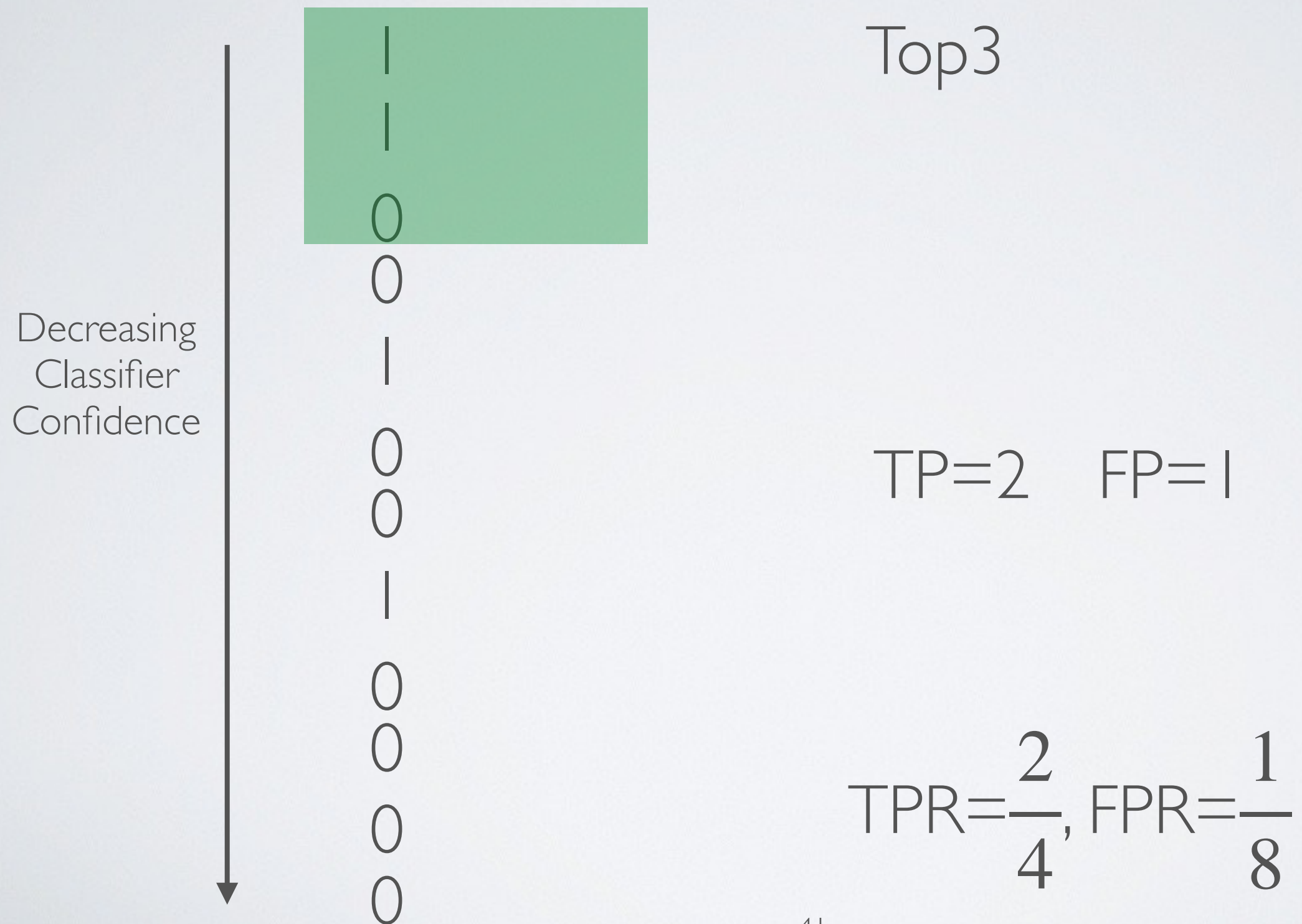
AUC



AUC



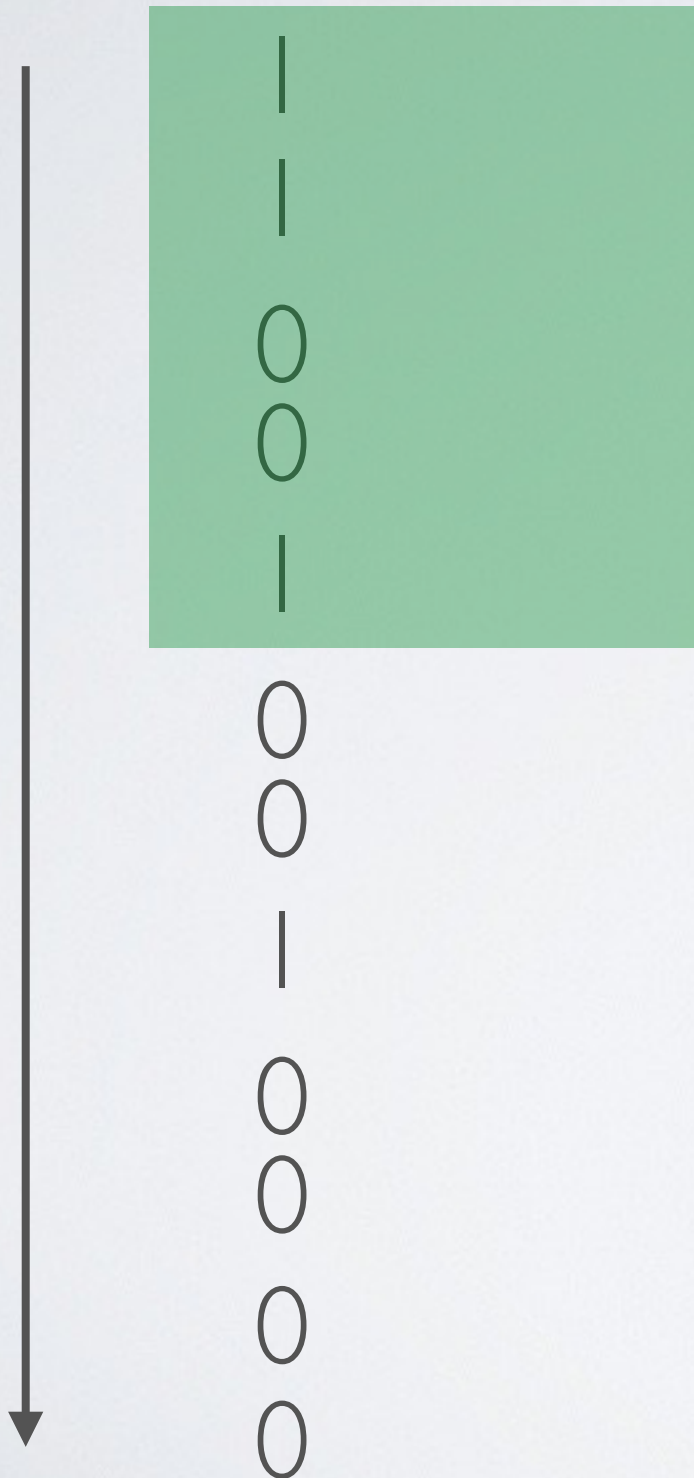
AUC



AUC

Top5

Decreasing
Classifier
Confidence



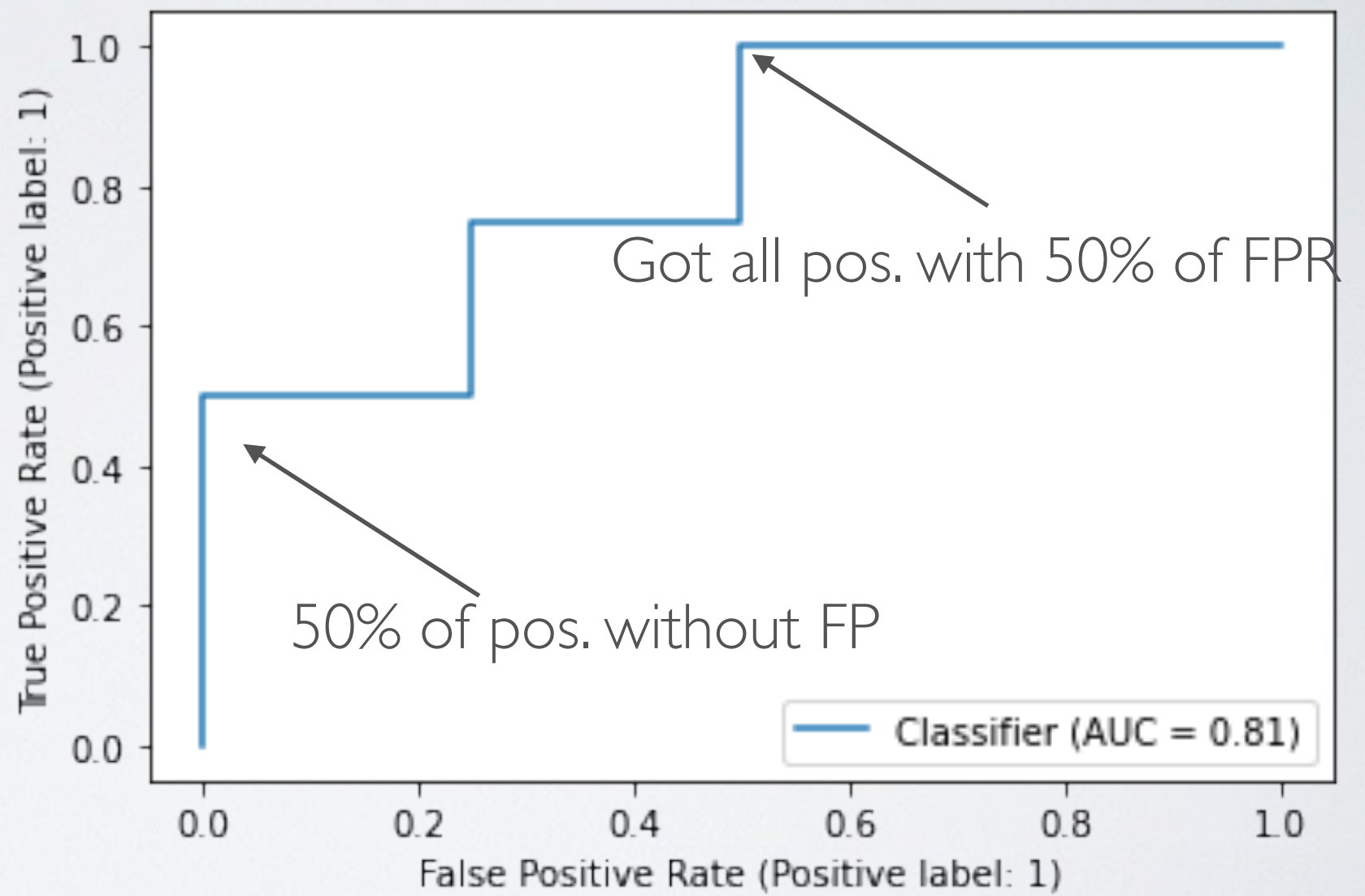
TP=3 FP=2

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

AUC

Decreasing
Classifier
Confidence

1
0
0
0
0
0
0
0
0
0



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
 - Not helpful if you care about the first few elements

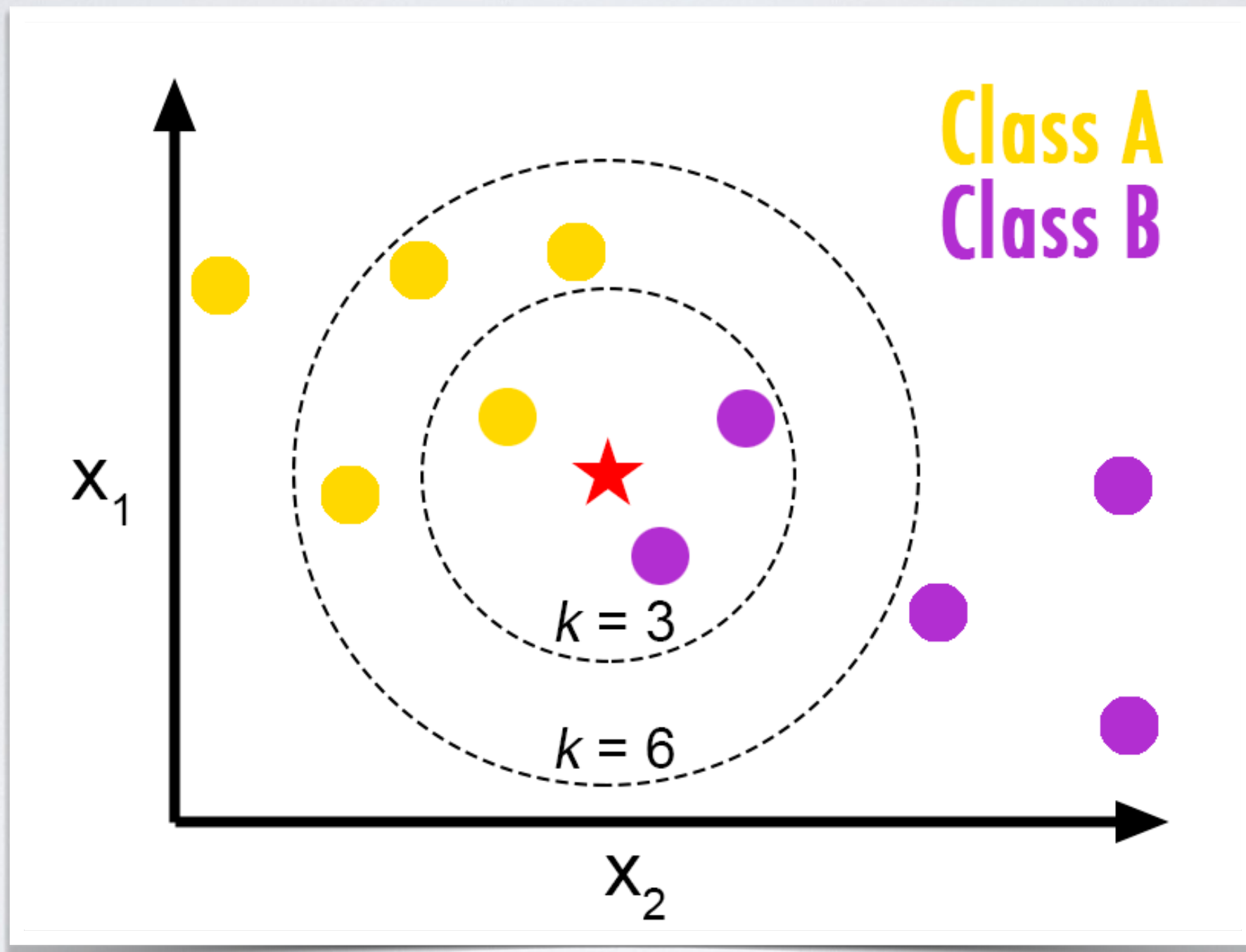
KNN

K nearest neighbors

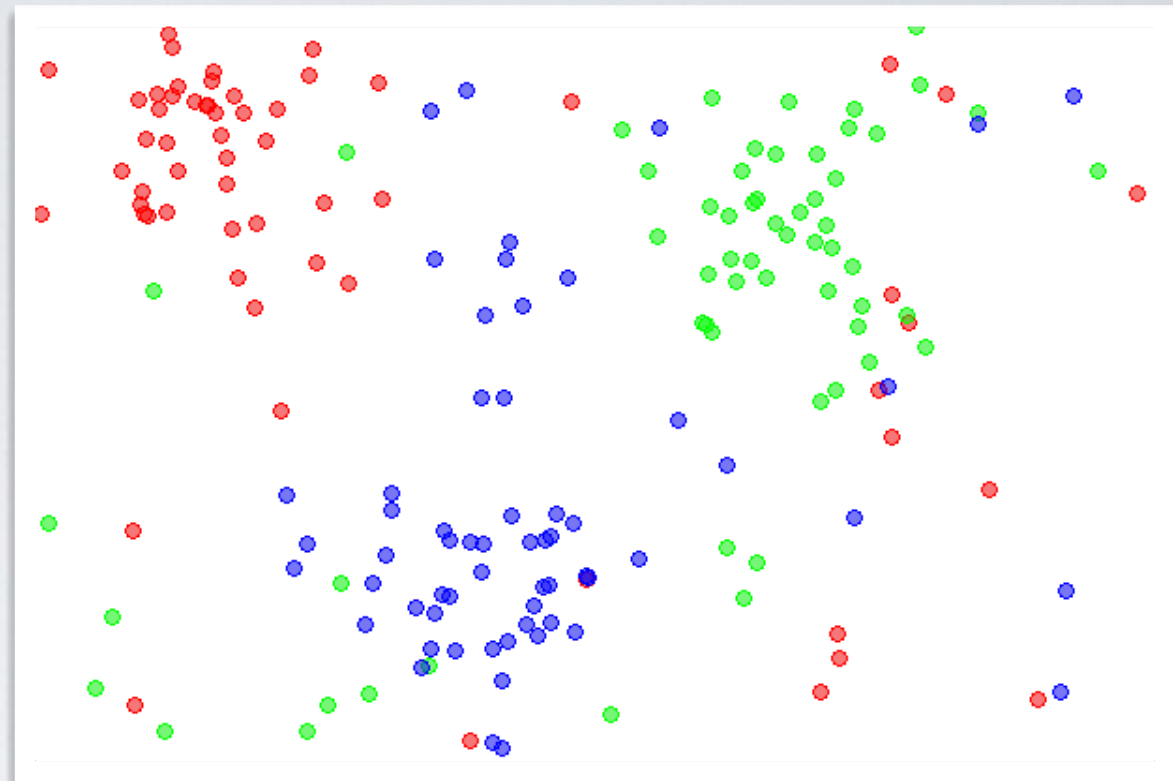
K-NN

- 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

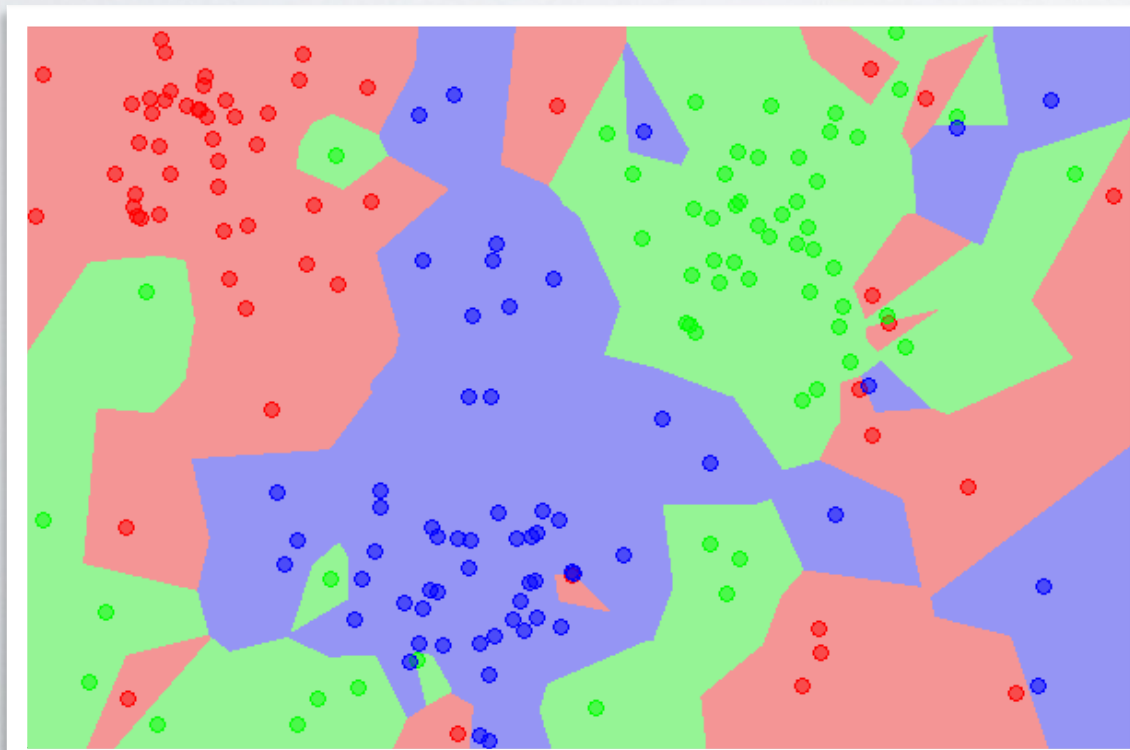
K-NN



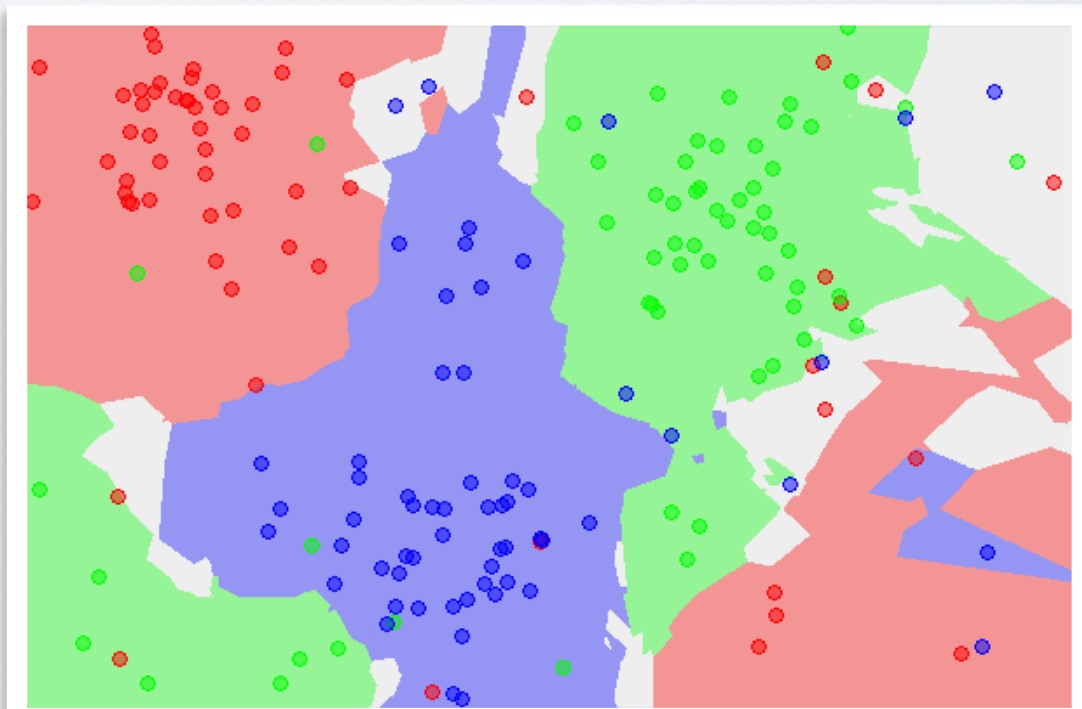
Dataset (2D, 3 classes)



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.
- No generalization to cases outside of the training set space

ML ADVANCED

REGULARIZATION

REGULARIZATION

- We have seen that a drawback of ML methods is that they can overfit
- When the ML objective can be clearly expressed, there is a generic way to limit overfitting: regularization
 - Two types of regularization:
 - L1 or Lasso regularization
 - L2 or Ridge regularization

L2 REGULARIZATION

- L2 or Ridge Regularization (for linear regression)

- $$\ell(b, w) = \frac{1}{N} \sum_i^N (y_i - (b + \sum_j^p (w_j x_{ij})))^2 + \lambda \sum_j^p w_j^2$$

- $$\ell(b, w) = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2 + \lambda \sum_j^p w_j^2$$

- Notation:
$$\sum_j^p w_j^2 = \|w\|_2^2$$

L2 REGULARIZATION

- Expressed as a general principle

$$\ell(b, w) = \frac{1}{N} \sum_i^N f(y_i, \hat{y}_i, b, w) + \lambda \sum_j^p w_j^2$$

- Some parameters are regularized, and some others might not be (intercept...)

- Intuition: we force coefficients to be small.

- If $\lambda=0$, normal regression
- If $\lambda \rightarrow \infty$, all coefficients tends towards 0
- /!\ The magnitude of coefficients depends on the magnitude of variables!
 - Important to normalize the variables, else you will constraint more the variables of lower amplitude

L1 REGULARIZATION

- L1 or Lasso Regularization

- Lasso: Least Absolute Shrinkage and Selection Operator

- $$\ell(b, w) = \frac{1}{N} \sum_i^N (y_i - (b + \sum_j^p (w_j x_{ij})))^2 + \lambda \sum_j^p |w_j|$$

- $$\ell(b, w) = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2 + \lambda \sum_j^p |w_j|$$

- Notation:
$$\sum_j^p |w_j| = \|w\|_1$$

REGULARIZATION

- Similar methods, different results:
 - L1 regularization tends to force some values to be 0
 - L2 regularization tends not to attribute 0
- L1 regularization thus performs **variable selection**
 - Variables for which the coefficient is 0 can be discarded

ELASTIC NET

- Best of both worlds :)

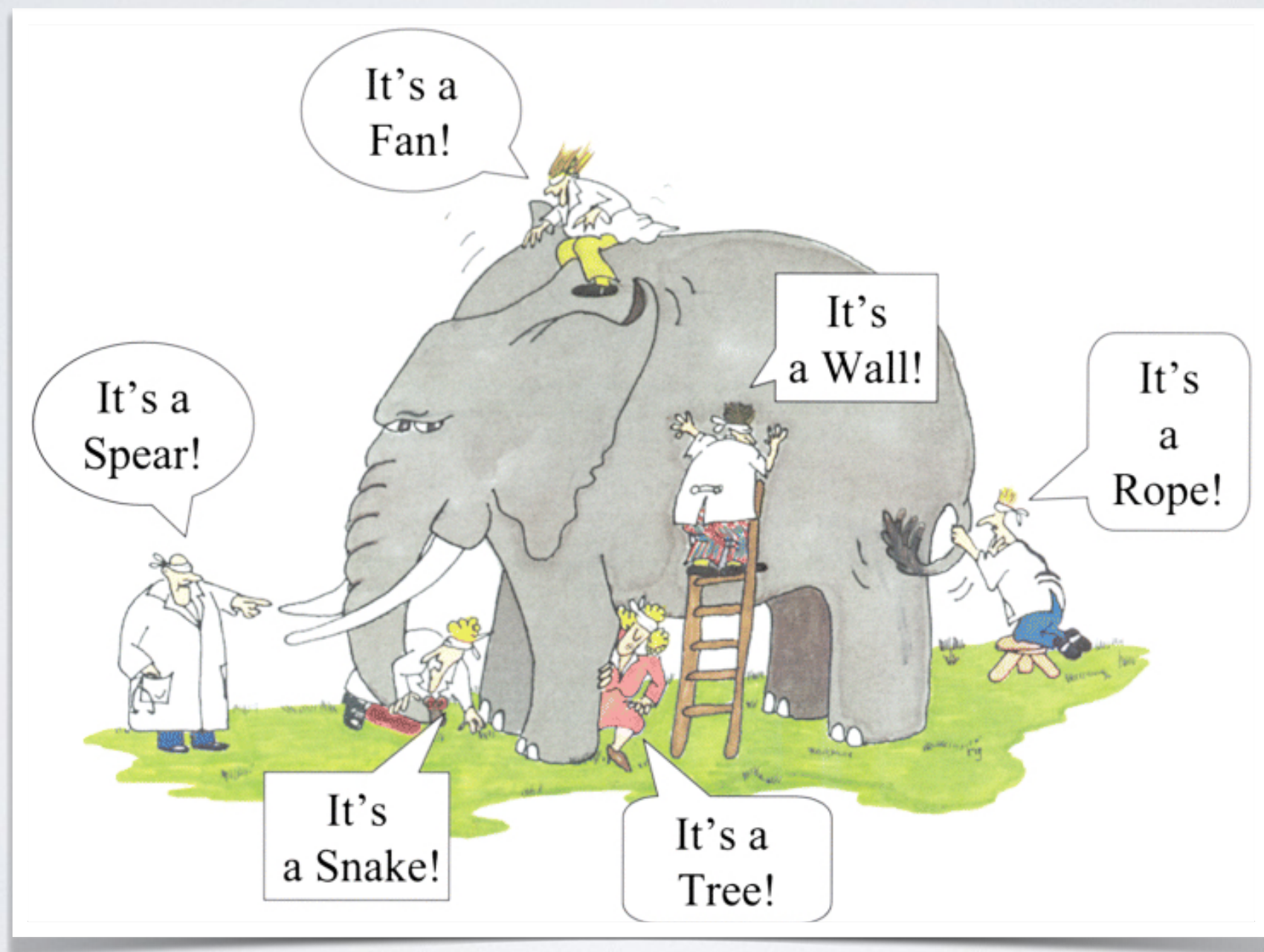
$$\bullet \ell(b, w) = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i) + \lambda_1 \sum_j^p |w_j| + \lambda_2 \sum_j^p w_j^2$$

ENSEMBLE LEARNING

ENSEMBLE LEARNING

- Ensemble learning is a general principle:
 - All models have strengths and weaknesses
 - e.g., linear models struggle with non-linearities but are good at extrapolation
 - Decision trees are good at capturing non-linearities, but struggle with extrapolation
 - Could we combine the strengths of various models?
 - Direct application: **Stacking**
 - Using multiple times the same model: **Bagging**
 - Training models specifically to solve other weaknesses: **Boosting**

ENSEMBLE LEARNING



STACKING

- In the simplest approach, various models (different approaches, same approach with different parameters) are trained on the same dataset
- Their predictions are then combined:
 - Regression: averaging. Average values of the classifiers (possibly weighted)
 - Classification:
 - Voting: class with the most vote
 - Soft / Averaging: average of probabilities yielded by the classifier
- Weaknesses:
 - What if several models make the same mistake? (Correlation of errors...)
 - What if we merge good models and poor models?

STACKING

- A possible solution to stacking is to use a meta-model:
 - The prediction made by each individual model is considered as a feature for the meta-model
 - The meta-model is trained as any ML model with the original target, but using sub-models outputs as features.
- Any model can be used as meta-model
- Famous for winning the \$1M prize of the 2009 Netflix prize.
 - 100+ individual predictors

BAGGING

- Bagging is an ensemble methods, but differ from stacking in two main ways:
 - The various individual predictors are made of the same algorithm
 - Each algorithm is trained on a subset of the original data
 - Different subsets on all variables
 - And/Or trained only on some variables
 - => Various strategies exist.
- Advantages over stacking:
 - All models are comparable, less chances to average “good” and “bad” models
 - Can be understood as “lower the Variance”, i.e., prevent overfit.

BAGGING: RANDOM FOREST

- Random forest is the most famous bagging algorithm
 - It is based on decision trees
 - A direct application of bagging
- Trees are good candidates for bagging because overfit is their main problem
 - What is similar between trees will stay, and when they disagree, taking the average of all the errors should get close to the right answer.
 - Similar to “Wisdom of the crowds”

RANDOM FOREST

- Set
 - Parameters of individual trees (not too simple, not too large...)
 - Averaging function
 - Nb. of trees
- What is specific is the subsample strategy
 - What is key is to avoid correlation between trees, i.e., train on different data
 - Subsample observations: With replacement. Sample n at random among n items
 - Variants: m among n . Or without replacement: random samples, or “folds” (each observation used in a single tree, but requires lot of data)...
 - Specific to trees: subsample of variables at each node: to chose the best split, restrain to a random fraction of variables.
 - Impose diversity in the trees

BOOSTING

BOOSTING

- Again, a general principle
- We train various models in sequence
 - First, train a normal model
 - Usually, this model will be tuned to be relatively simple, and thus underfit=>Weak learners
 - Then, extract the errors of the model (incorrect classes/residuals).
 - Train a second model, focusing on predicting the errors missed by the first model
 - Update the main model and recompute the errors
 - Repeat until we cannot improve anymore
- Final prediction is the sum of all weak learners (not average: each method *corrects, complements* previous ones)

$$F_T(x) = \sum_{t=1}^T f_t(x)$$

XGBOOST

XGBOOST

- Very popular method among those not using neural networks
- Used in winning solutions in countless ML challenges
 - And at Google, Amazon, Uber...
- Both :
 - A method described in a scientific paper
 - A library developed and improved by a community

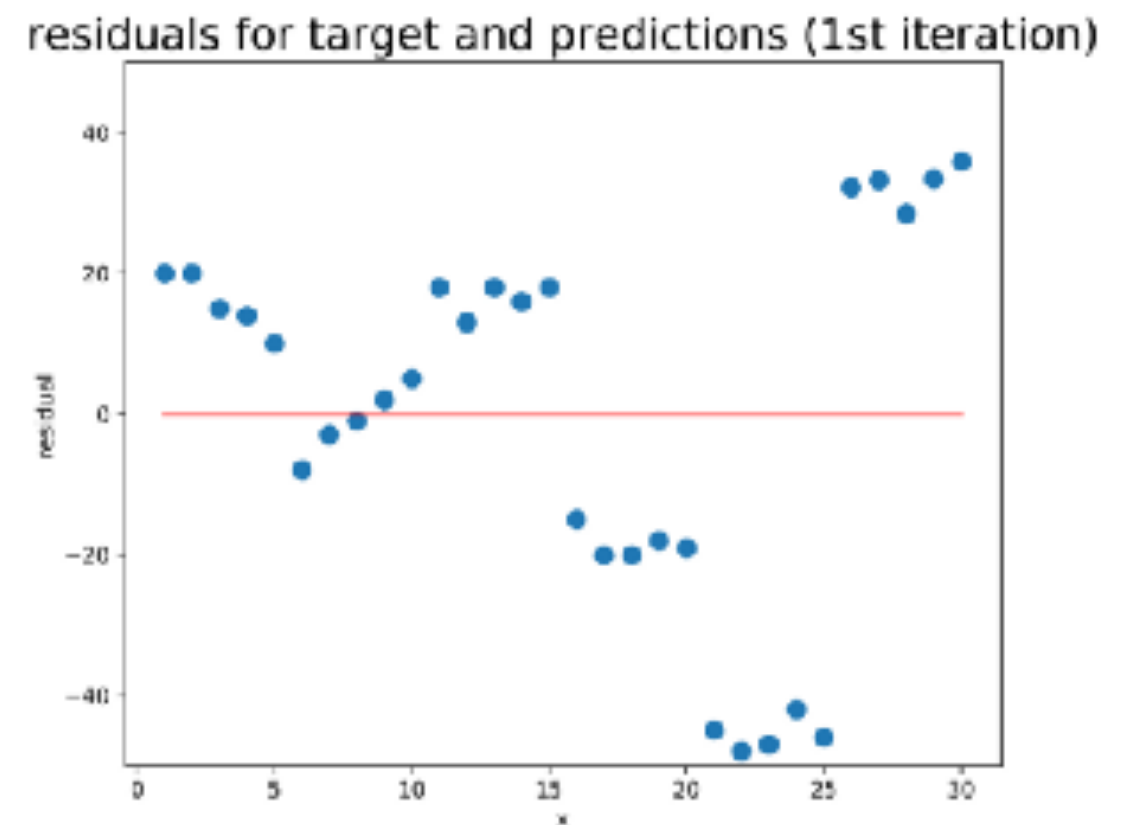
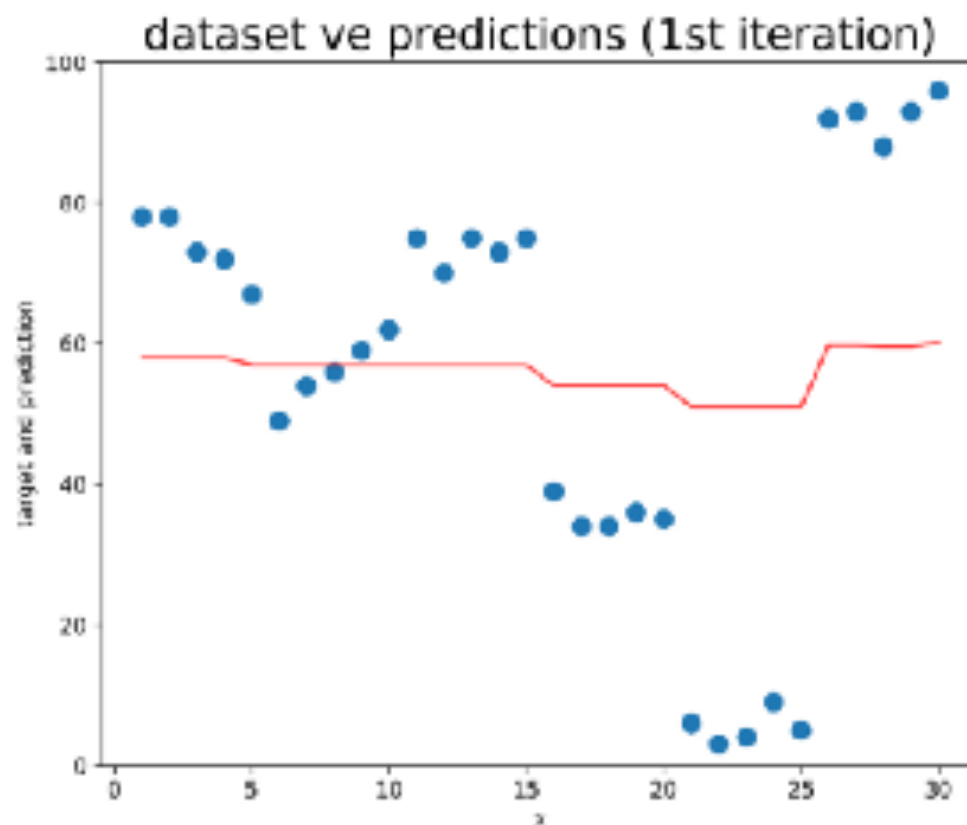
XGBOOST

- In a few words:
 - A tree boosting methods
 - Can be used for classification and regression
 - Weak learners
 - Default to 3 or 6 levels max
 - Introduces Regularization
 - Each new leaf adds some regularization cost
 - Gradient Boosting:
 - Explicitly do a gradient-descent-like approach

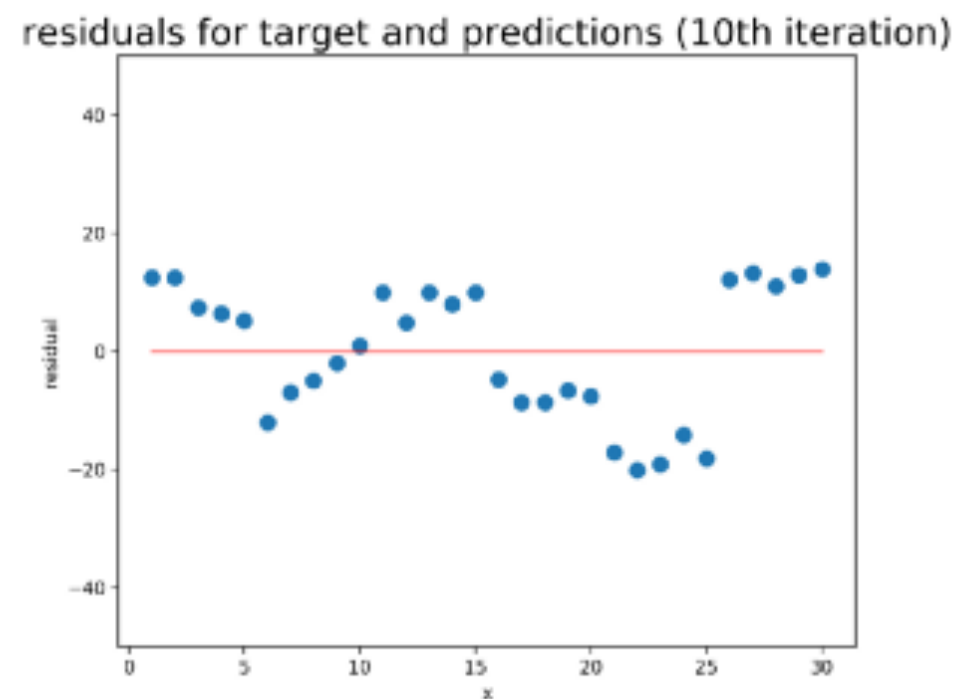
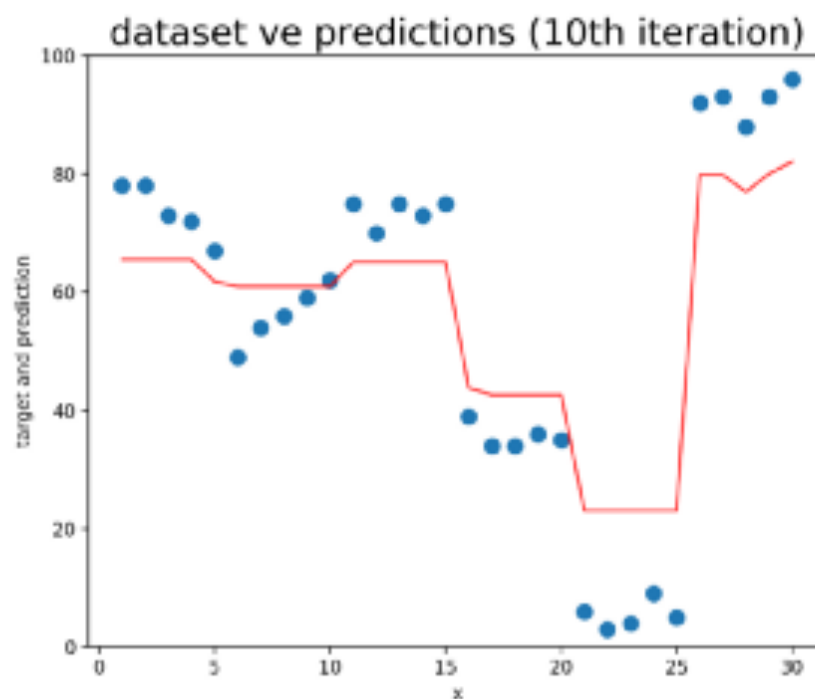
GRADIENT BOOSTING

- Gradient boosting is the application of boosting to explicit gradient descent

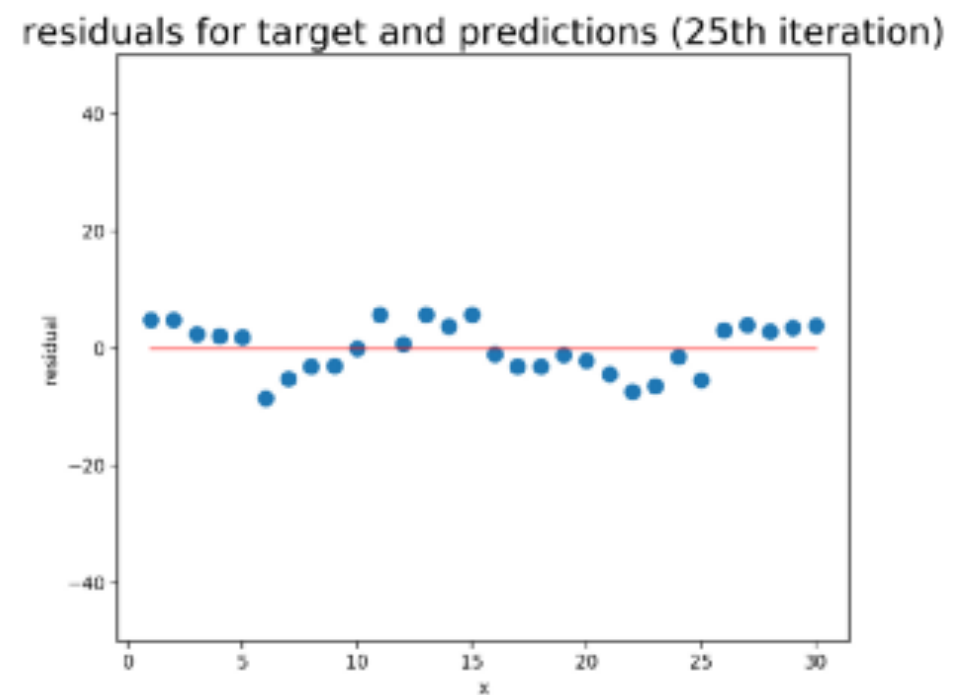
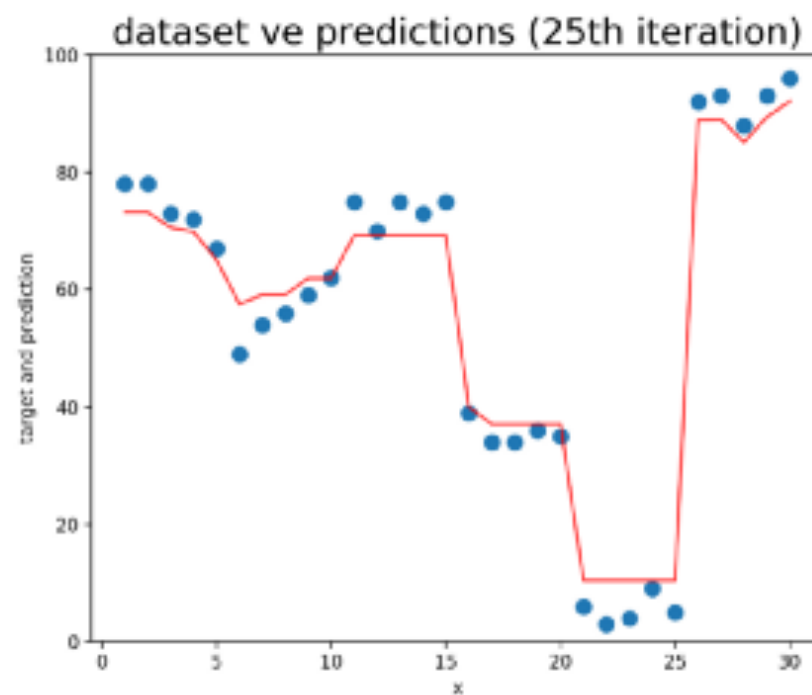
GRADIENT BOOSTING



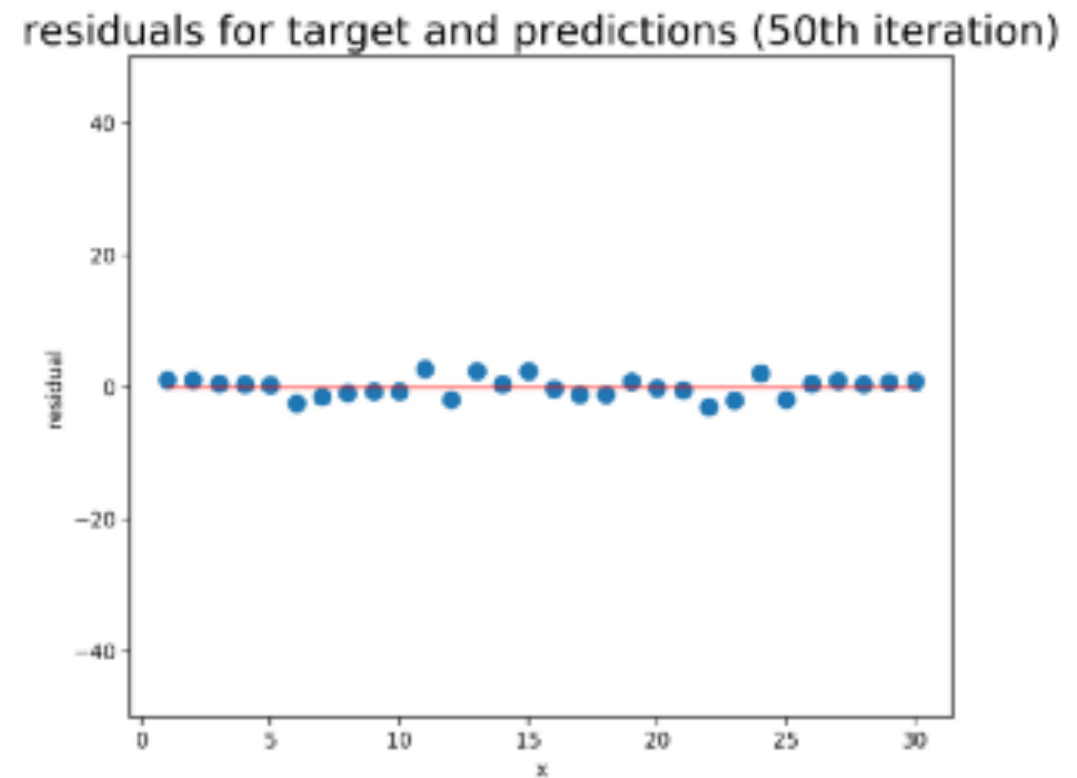
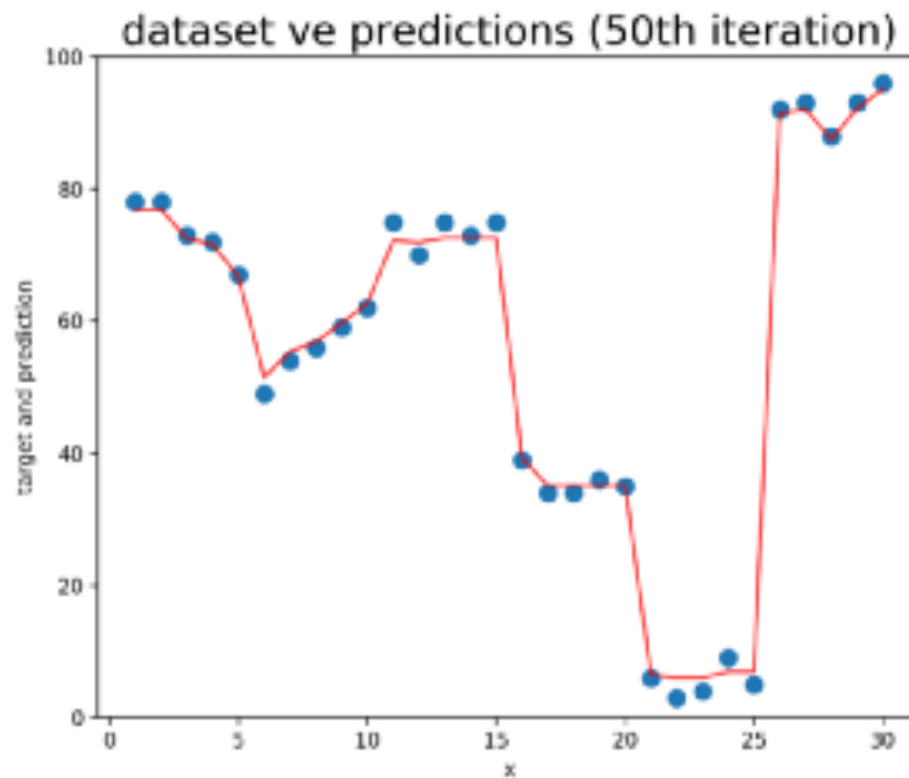
GRADIENT BOOSTING



GRADIENT BOOSTING



GRADIENT BOOSTING



XGBOOST IN A NUTSHELL

$$\begin{aligned}\text{obj}^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \omega(f_i) \\ &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{i=1}^t \omega(f_i)\end{aligned}$$

- In our loss for the tree, we decompose the prediction $\hat{y}$ as
 - Prediction given by previous tree + prediction of new tree.
 - ω regularization, explained later

XGBOOST IN A NUTSHELL

$$w_j = - \frac{G_j}{H_j + \lambda}$$

- w_j : score of a leaf
- Using RMSE as an objective:
 - G_j : Sum of errors (to residuals)
 - H_j : Number of items in the leaf
 - λ : Regularization parameter

GAIN ON A SPLIT

- $Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$
- $L, R \Rightarrow$ Left and Right children
- Sum of regularized averaged error of the children squared, minus that of parent, minus regularization γ

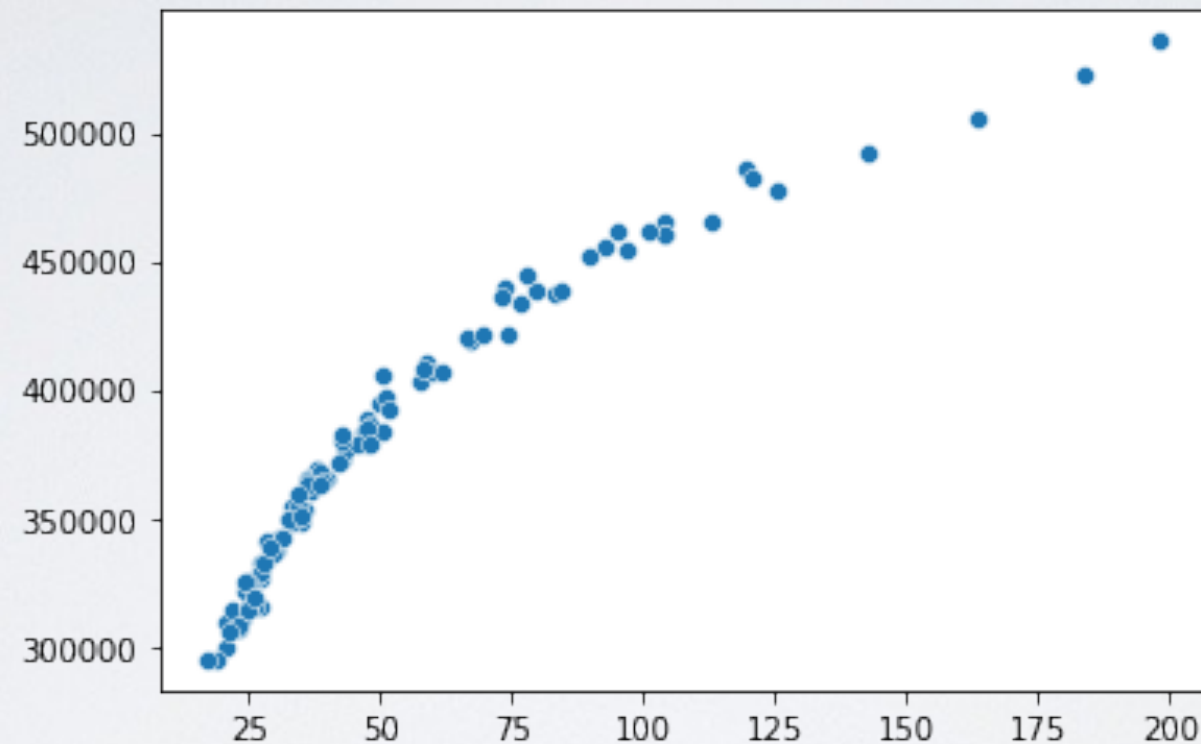
XGBOOST IN A NUTSHELL

- For First tree:
 - For each leaf
 - We compute the gain to find the best possible split,
 - If regularization makes the gain negative, do nothing
 - If we reach the maximal tree depth, do nothing
 - Compute the final score of the leaf : signed error. To add to the final prediction
- Next tree: same process, but compute error relatively to previous tree (residuals)
- When finished, for each prediction, sum the (signed) prediction of each tree (weighted by learning rate η)

LEARNING RATE

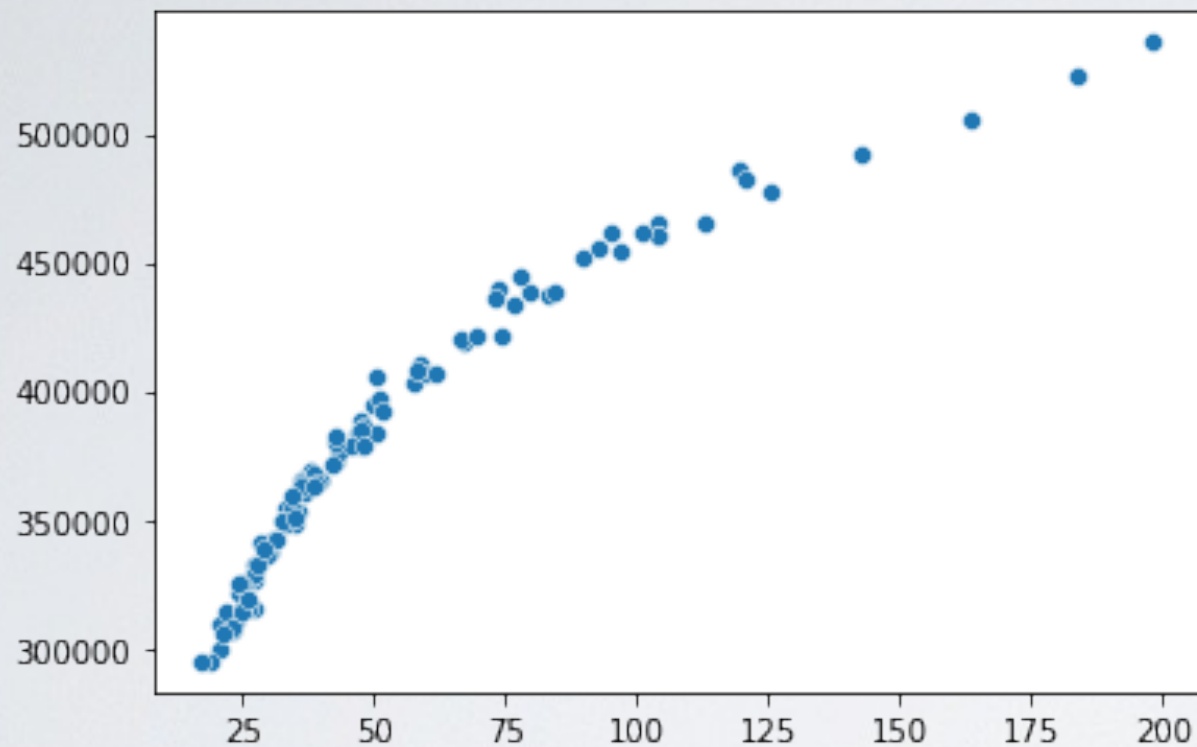
- As in most gradient descent methods, there is a learning rate η (eta) parameter, allowing to tune how fast we converge
 - To avoid the “ping-pong” effect around global minimum
 - In practice, the prediction of the previous tree is shrunk by η
- $\hat{y} = \eta \hat{y}_i^{(t-1)} + f_t(x_i)$

XGBOOST: EXAMPLE

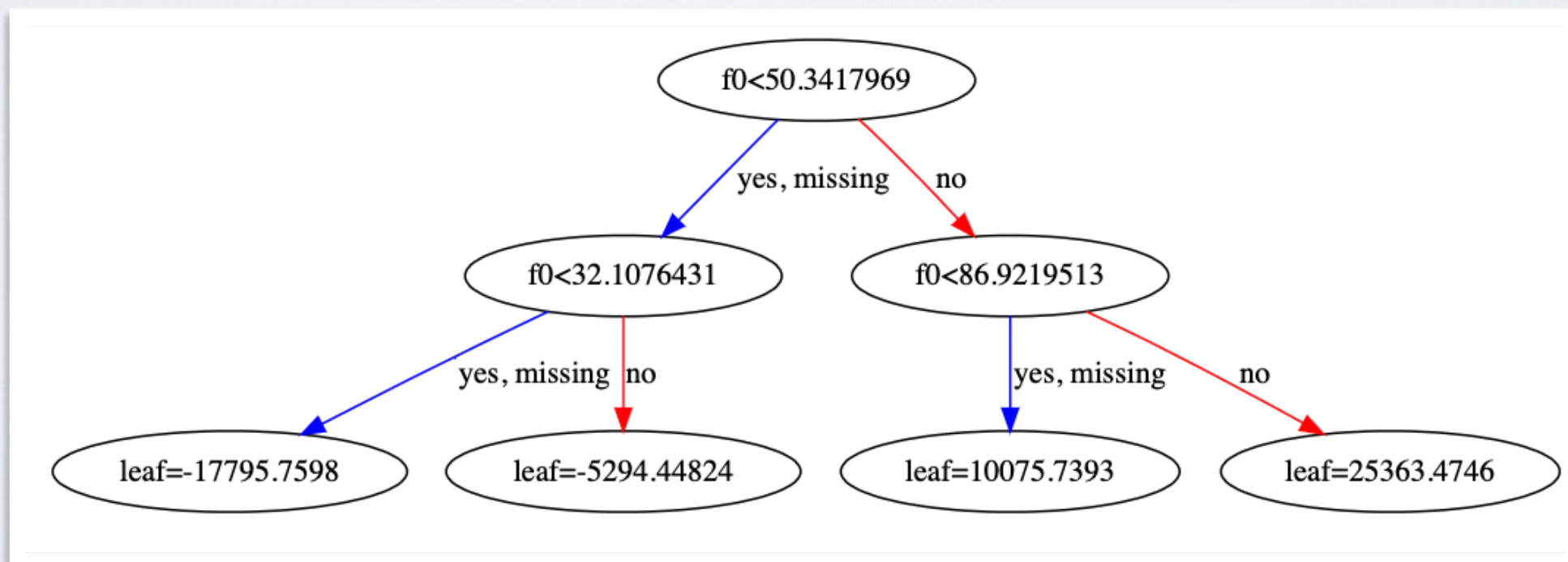


```
objective="reg:squarederror",  
learning_rate=0.3,  
base_score=np.mean(Ytrain),  
max_depth=2
```

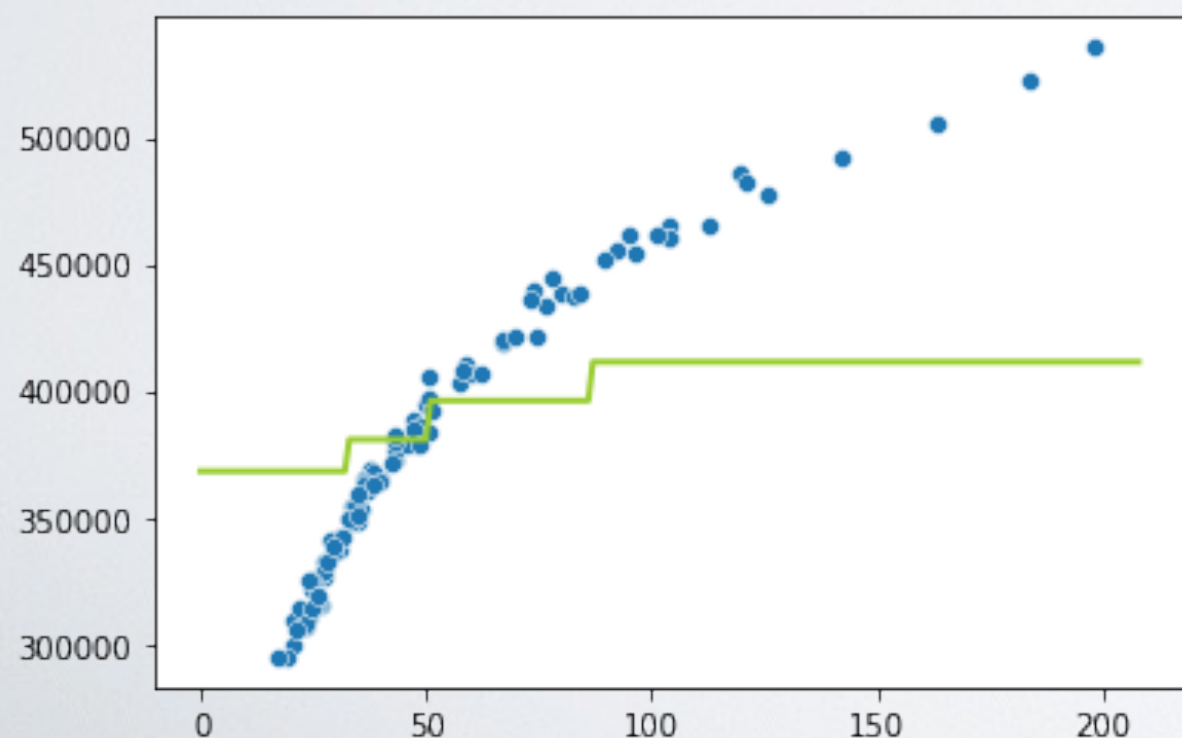
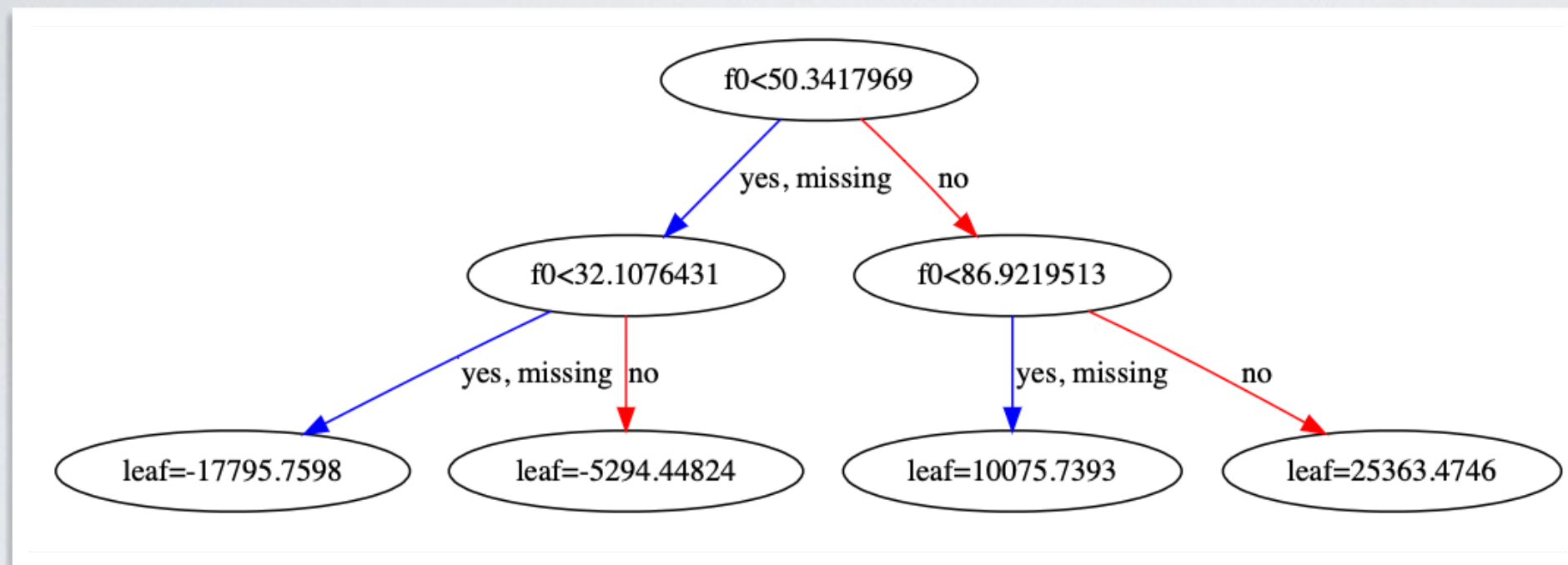
XGBOOST: EXAMPLE



First tree



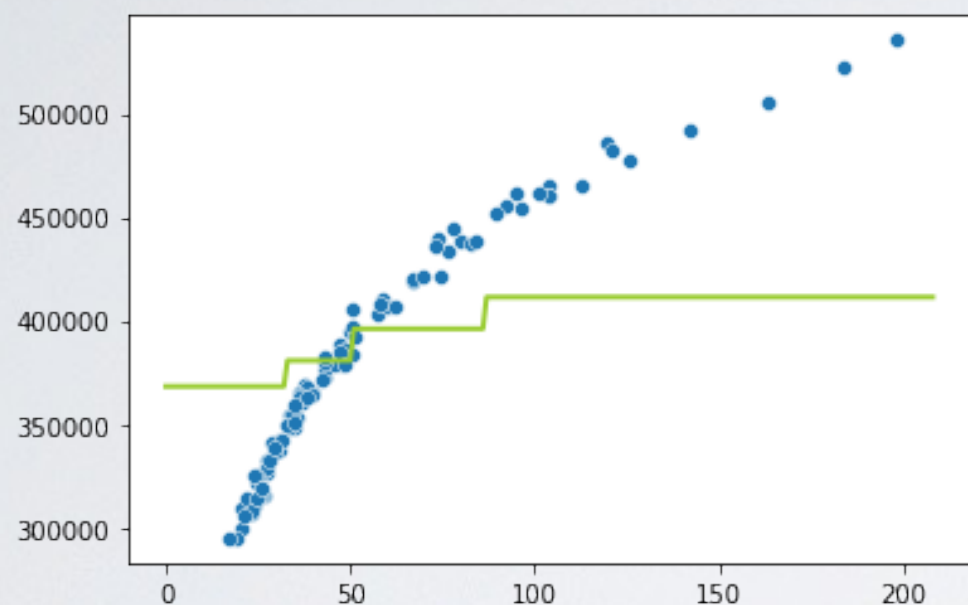
XGBOOST: EXAMPLE



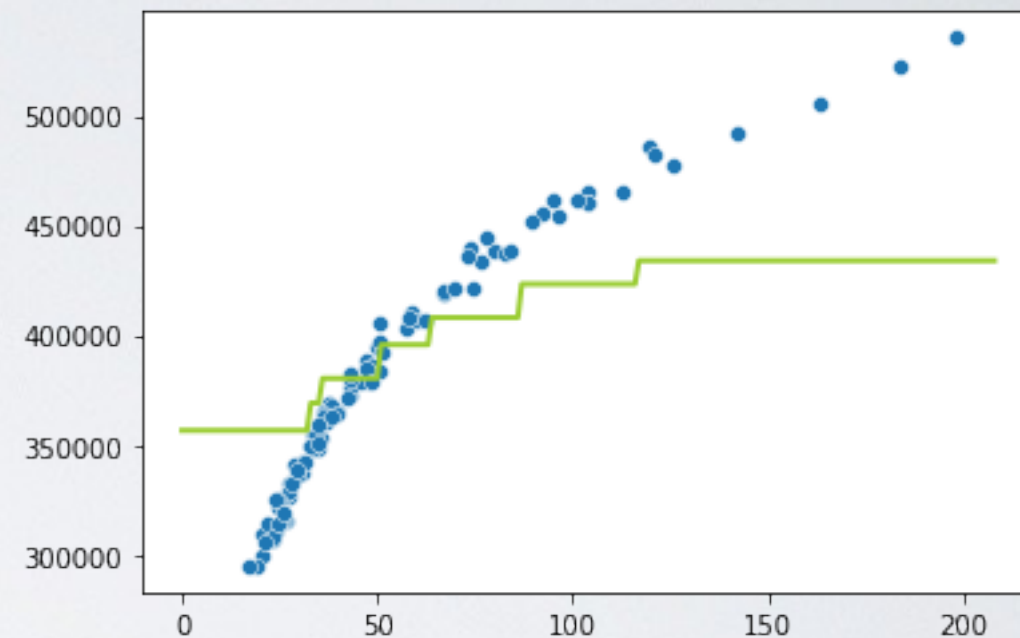
| single tree for prediction:
Learning rate effect...

XGBOOST: EXAMPLE

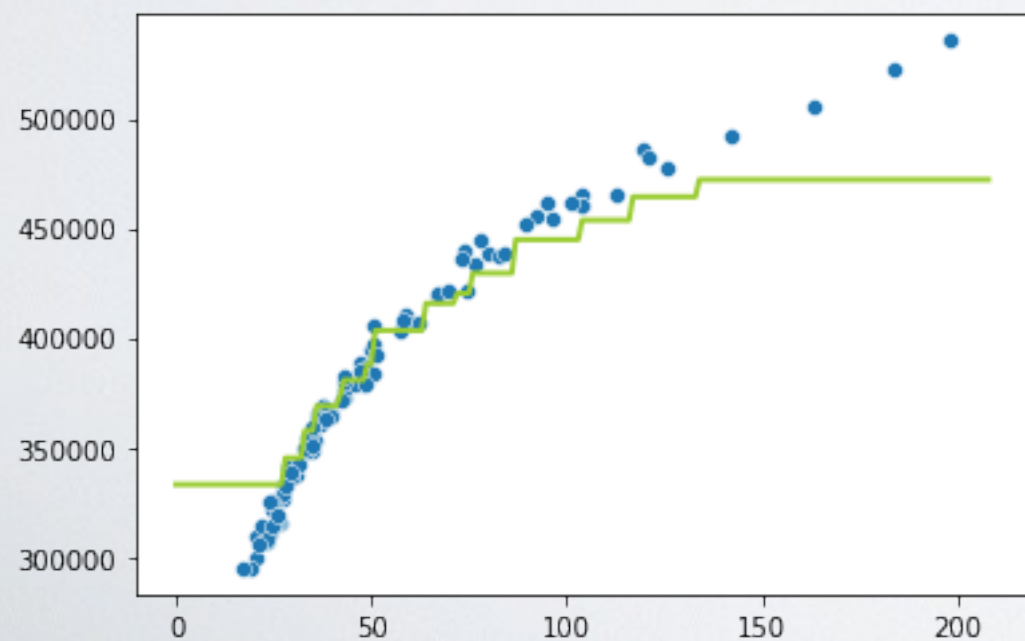
1



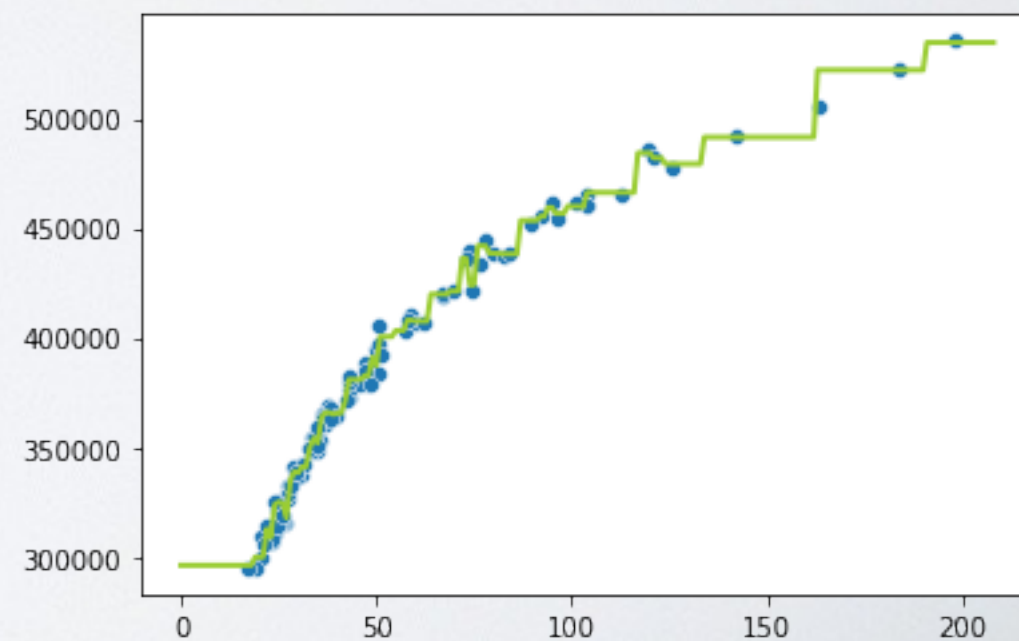
2

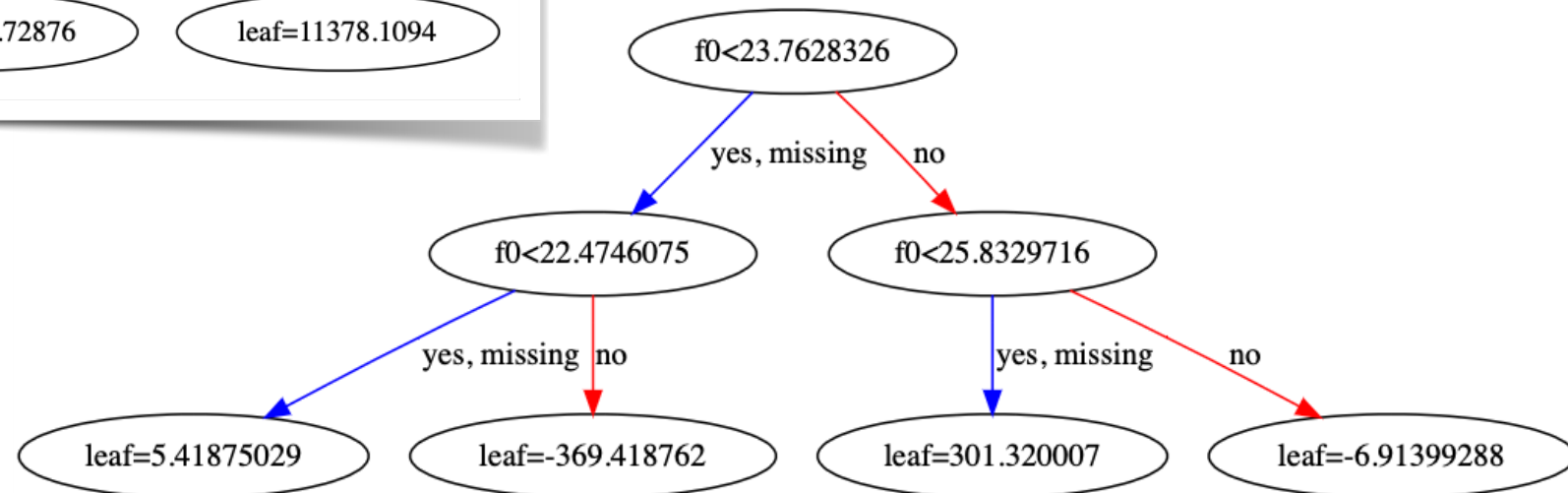
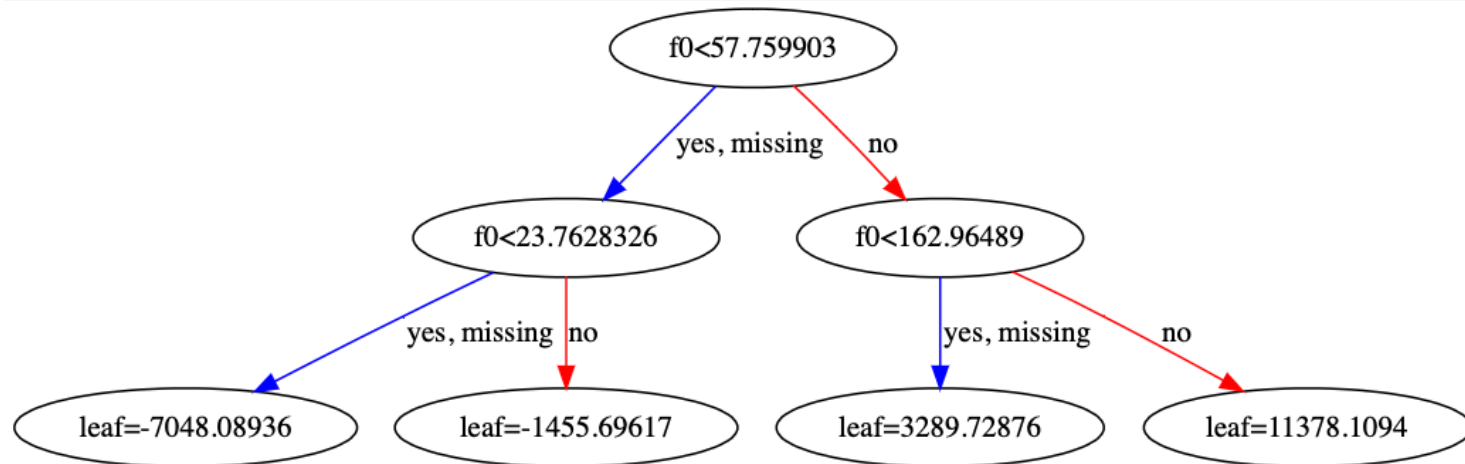
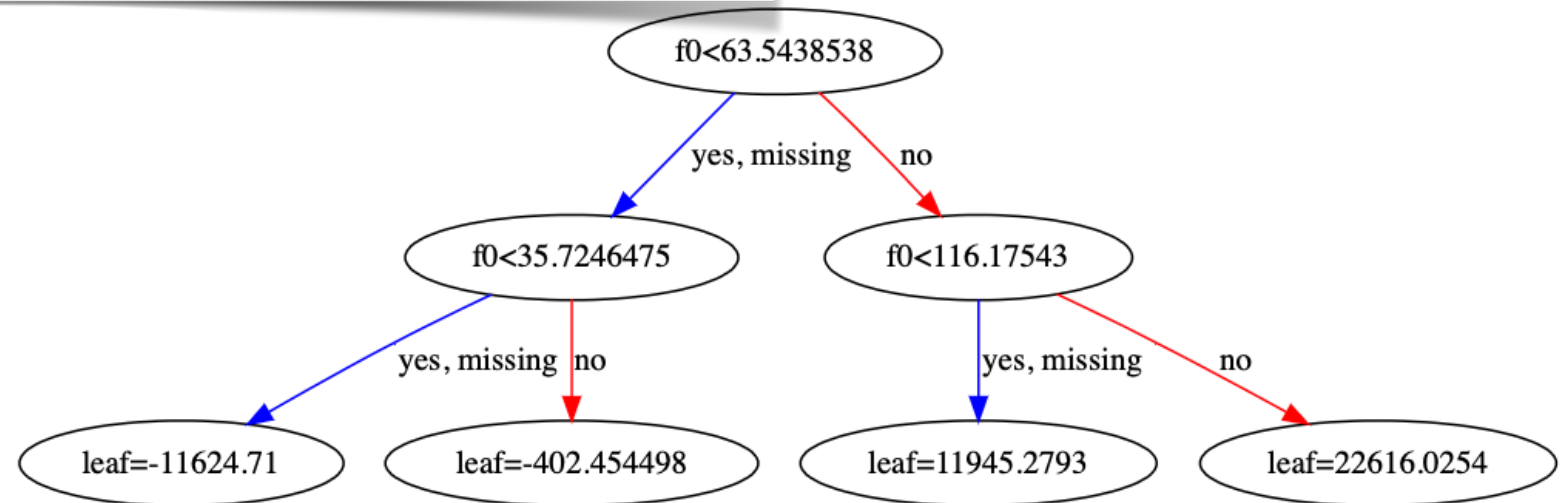
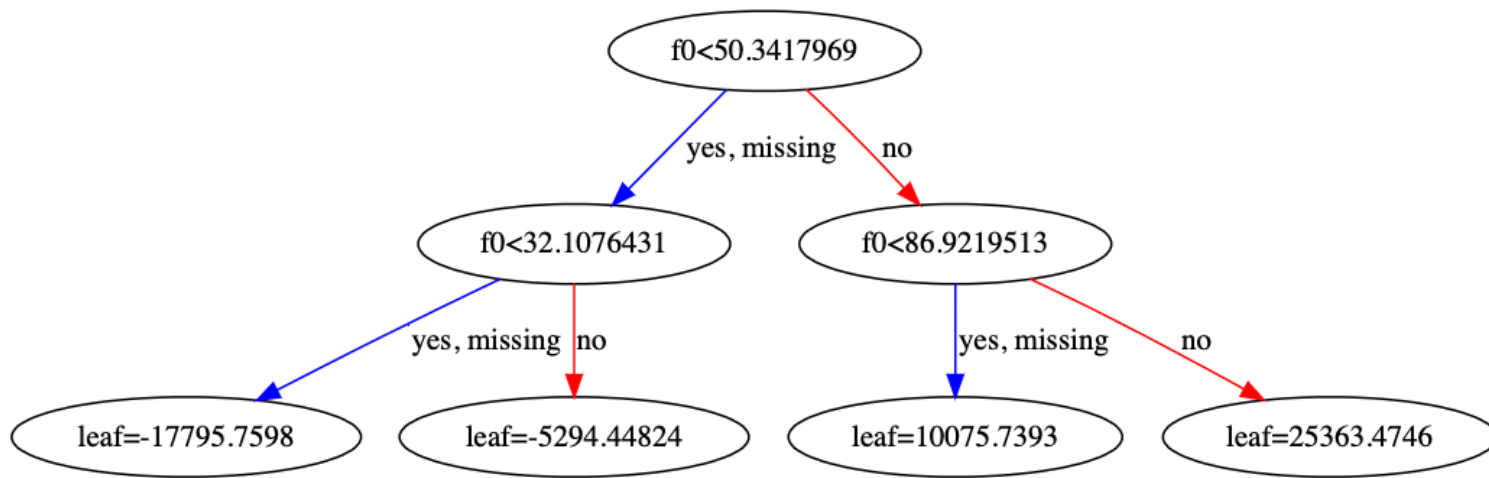


5



50





CLASSIC ML VS DNN

- Until now, I have presented “classic” methods.
- For most people, $ML=AI=DNN$. Are classic obsolete?
 - DNNs are good for problems with
 - Huge quantity of data
 - Structured data: variables of same nature related with each other (adjacent pixels, following words...)
 - If limited data, or tabular data: XGboost & Co. are the most used and usually most efficient methods