Classification with imbalanced datasets

IMBALANCED DATASETS

- Imbalanced datasets contain much more data for one of the classes (the majority class) than the other. E.g. most people do not have cancer.
- Example of imbalanced dataset:
  - 990 negative instances (99%) (majority class)
  - 10 positive instances (1%) (minority class)
- This happens a lot in medical datasets (cancer vs. non-cancer)

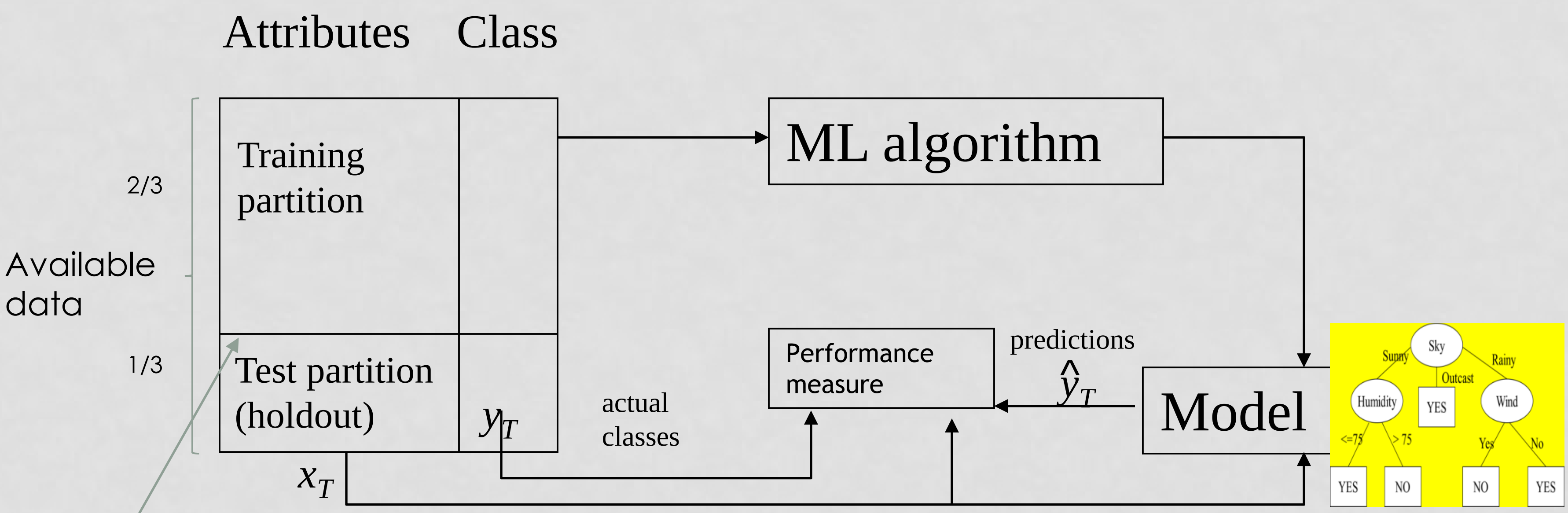
ISSUES IN IMBALANCED PROBLEMS

- **Evaluation** for imbalanced datasets
- **Learning/training** with imbalanced datasets

Stratified partitions

- Stratified partitions keep the same positive / negative class distribution in train and test sets

STANDARD HOLDOUT (TRAIN/TEST) FOR MODEL EVALUATION



$$Accuracy = \frac{1}{n}\sum_{k=1}^{n} y_k == \hat{y}_k$$

if we split randomly, it is likely that the test partition will not be representative of the original dataset.

Stratified partitions

- Stratified partitions keep the same positive / negative class distribution in train and test sets
- Example: if in the available data the distribution is 99%(-) / 1%(+), train and test should have the same distribution.
- This kind of partition is difficult to achieve in imbalanced datasets. It has to be enforced
- Both for train/test (holdout) and crossvalidation.

For holdout, use: (where y is the response variable)

- X_train, X_test, y_train, y_test = train_test_split(X, y, test_size0.33, random_state=42, **stratify = y**)

For crossvalidation, use:

- **sklearn.model_selection.StratifiedKFold**

ACCURACY NOT USEFUL FOR DOING MODEL EVALUATION ON IMBALANCED PROBLEMS

- The standard performance measure for classification is accuracy = average number of correctly classified instances

$$\boxed{Accuracy = \frac{1}{n}\sum_{k=1}^{n} y_k == \hat{y}_k}$$

- 99% (majority/negative class) / 1% (minority/positive class)
- A **trivial/naive (dummy) classifier** that always says "Negative" would already obtain 99% accuracy.
- However, this trivial classifier will **fail** with all minority class instances and be correct with all majority class instances.
- Accuracy is therefore not a useful metric in imbalanced problems because even trivial/dummy models obtain very high accuracies.
- An initial way of evaluating models properly is not to use the accuracy metric (which is an overall / global metric) but rather, break down accuracy by class:
  - Accuracy of the positive class
  - Accuracy of the negative class

THE CONFUSION MATRIX

- Accuracy is an overall / global measure
- But for imbalanced problems, it is interesting to break down the success rate by class:
  - Accuracy of the positive class (+)
  - Accuracy of the negative class (-)

	Clasiffied as +	Classified as -
Instances actually +	**TP (true positive)**	FN (false negative)
Instances actually -	FP (false positive)	**TN (true negative)**

THE CONFUSION MATRIX

- “+” = cancer and “-” = not-cancer
- 100 positive instances and 100 negative instances
- Notice that accuracy = (TP+TN) / (TP+TN+FP+FN)
- Notice that the accuracy is the same in both cases = (90+60)/200 = 0.75

Decision tree

	Clasiffied as +	Classified as -
Instances actually +	**TP 90**	FN 10
Instances actually -	FP 40	**TN 60**

THE (NORMALIZED) CONFUSION MATRIX

- Typically, confusion matrices are normalized by dividing by the amount of positive instances and the amount of negative instances.
- TPR and TNR can be interpreted as the accuracy of the positive and the negative classes, respectively.
- FPR and FNR can be interpreted as the missclassification errors of the negative and the positive classes, respectively

Confusion matrix

	Clasiffied as +	Classified as -
Instances actually +	**TP 90**	FN 10
Instances actually -	FP 40	**TN 60**

Normalized confusion matrix

	Clasiffied as +	Classified as -
Instances actually +	**TPR = 90/(90+10) = 0.9**	**FNR = 10/(90+10) = 0.1**
Instances actually -	**FPR = 40/(40+60) = 0.4**	**TNR = 60/(40+60) = 0.6**

THE CONFUSION MATRIX

- If we are more interested in the positive class (cancer) than in the negative class, which model should be choose?

Decision tree

	Clasiffied as +	Classified as -
Instances actually +	**TPR = 0.9**	FNR = 0.1
Instances actually -	FPR = 0.4	**TNR = 0.6**

KNN

	Clasiffied as +	Classified as -
Instances actually +	**TPR = 0.6**	FNR = 0.4
Instances actually -	FPR = 0.1	**TNR = 0.9**

- There is usually a trade-off between TPR and TNR:
  - It is difficult for models to get both high TPR and high TNR
  - Tipycally a model gets high TPR by sacrificing TNR (or the other way around)
- Notice that it is enough to use just two of those numbers to describe a classifier, because TPR+FNR=1 and TNR+FPR=2
  - TPR and TNR
  - TPR and FPR

RECALL AND PRECISIÓN (INSTEAD OF TPR AND TNR)

- **Recall** = How many (ratio) of the positive instances are correctly identified as positive (== TPR)
  - Recall == TPR = TP / (TP+FN) = TP / (number of positive instances)
  - Higher means that if an instance is positive, it is likely to be classified as positive by the model
- **Precision** = How many (ratio) of the instances classified as positive are actually positive
  - TP/(TP+FP) = TP / (TP+FP) = TP / (number of instances classified as positive)
  - Higher means that if a model classifies an instance as positive, it is very likely that it is actually positive.
  - It answers the question "if the model predicts "+", should I trust the prediction of the model?

Confusion matrix

	Clasiffied as +	Classified as -
Instances actually +	**TP 90**	FN 10
Instances actually -	FP 40	**TN 60**

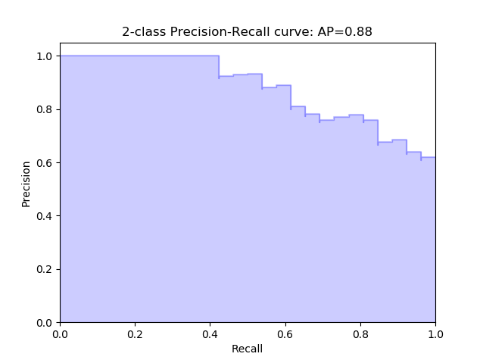
Recall = TPR = 90/(90+10) = 90/100 = 0.9
Precision = 90/(90+40) = 0.69

RECALL AND PRECISION

- **Recall** = How many (ratio) of the positive instances are correctly identified as positive (== TPR)
  - Recall == TPR = TP / (TP+FN) = TP / (number of positive instances)
  - Higher means that if an instance is positive, it is likely to be classified as positive by the model
- **Precision** = How many (ratio) of the instances classified as positive are actually positive
  - TP/(TP+FP) = TP / (TP+FP) = TP / (number of instances classified as positive)
  - Higher means that if a model classifies an instance as positive, it is very likely that it is actually positive.
- There is usually a trade-off between Recall and Precision. If a model does very well at precision, it is usually by sacrificing recall.

Confusion matrix of a trivial classifier (always predicts "+")

	Clasiffied as +	Classified as -
Instances actually +	**TP 100**	FN 0
Instances actually -	FP 100	**TN 0**



Recall = TPR = 1.0
Prec = 100/(100+100) = 0.5

METRICS DERIVED FROM THE CONFUSION MATRIX, APPROPRIATE FOR IMBALANCED PROBLEMS

- Single-value metrics, not sensitive to class imbalance (unlike accuracy): if the metric value is high, that means that the model is doing reasonably well (in general, not for just one of the clases).
- Out of TPR/TNR:
  - Balanced accuracy (bac) = (TPR+TNR)/2
    - Good values if > 0.5
    - Unlike accuracy, in order to get high BAC it is neccessary to get both high TPR and TNR
    - Youden's J index: TPR+TNR-1 = BAC*2 - 1

EVALUATION WITH CONFUSION MATRIX

100000 test instances: 99500 close, 500 open

Classified as

c_2	OPEN	CLOSE
open	0.0	1.0
close	0.0	1.0

ACC: 0.995 = 99500/100000
BAC = 0.5 = (0 + 1) / 2

- BAC = balanced accuracy = (TPR + TNR) / 2
- Unlike accuracy, in order to get high BAC it is neccessary to get both high TPR and TNR

EVALUATION WITH CONFUSION MATRIX

100000 test instances: 99500 close, 500 open

Classified as

c_1	OPEN	CLOSE
open	0.60	0.40
close	0,005	0.995

Actual

ACC: 0.993 = (0.6*500+0.995*99500) / 100000
BAC = 0.80 = (0.60 + 0.995) / 2

Classified as

c_2	OPEN	CLOSE
open	0.0	1.0
close	0.0	1.0

ACC: 0.995 = 99500/100000
BAC = 0.5 = (0 + 1) / 2

Classified as

c_3	OPEN	CLOSE
open	0.80	0.20
close	0.054	0.946

ACC: 0.945 = (0.8*500+0.946*99500) / 100000
BAC = 0.873 = (0.80 + 0.946) / 2

- BAC = balanced accuracy = (TPR + TNR) / 2
- Unlike accuracy, in order to get high BAC it is neccessary to get both high TPR and TNR

METRICS DERIVED FROM THE CONFUSION MATRIX, APPROPRIATE FOR IMBALANCED PROBLEMS

- Single-value metrics, not sensitive to class imbalance (unlike accuracy): if the metric value is high, that means that the model is doing reasonably well (in general, not for just one of the clases).
- Out of Precision / Recall:
  - F1 score = harmonic mean of precision and recall
  - $F1 = 2 * \frac{P * R}{P + R}$

INDEX

- **Evaluation** for imbalanced datasets
- **Learning/training** with imbalanced datasets:
  - Resampling
  - Thresholding

TRAINING MODELS FOR IMBALANCED DATASETS

- So far, we know how to evaluate models for imbalanced datasets
- But the techniques for training the model are still the same
- It is known that standard techniques do not work well under imbalance: they tend to learn well the majority class (high TNR) but not so well the minority class (low TPR).
- That is, we get results like this:

Classified as

Actual

c_1	OPEN	CLOSE
open	0.60	0.40
close	0,005	0.995

Classified as

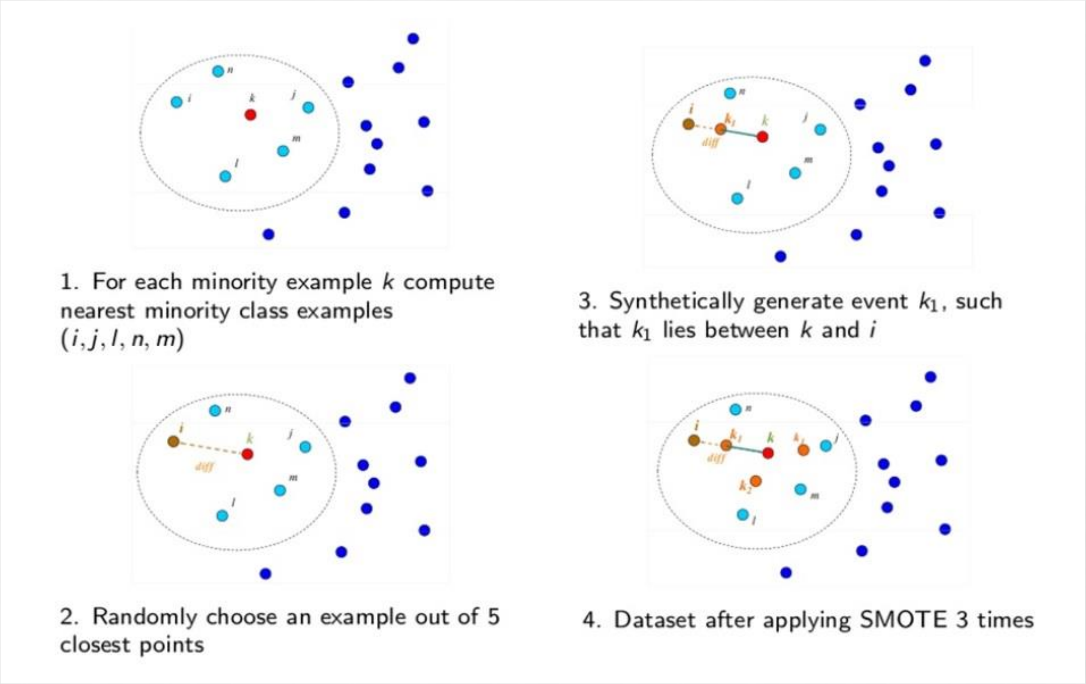
c_2	OPEN	CLOSE
open	0.0	1.0
close	0.0	1.0

TRAINING MODELS FOR IMBALANCED DATASETS

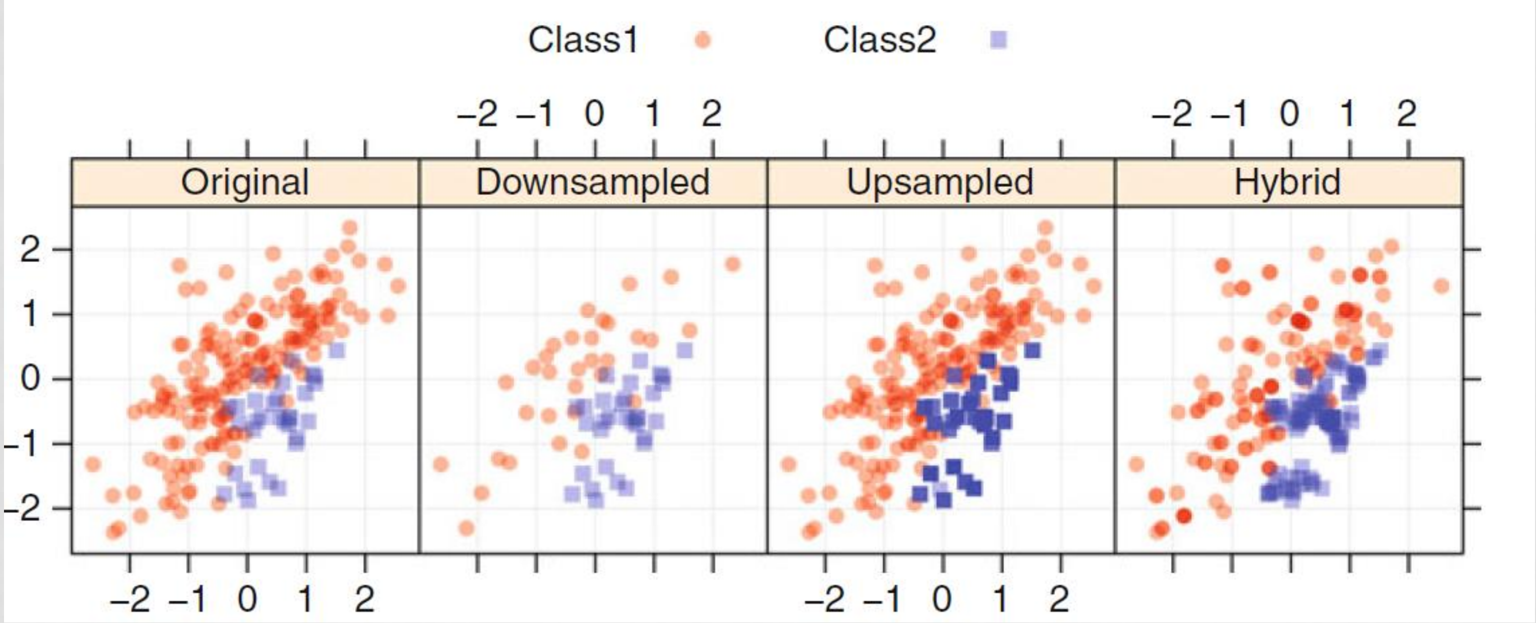
- Solution 1: train several models (e.g. with different methods) and select the one that works well for some metric (e.g. BAC or F1). Also, do hyper-parameter tuning but optimize BAC or F1
- Solution 2: sampling:
  - Undersampling: remove instances from the majority class. Problem: information is lost.
  - Oversampling: replicate instances for the majority class (or change weights, like in Boosting, if the method is able to deal with instance weights). Problem: overfitting
  - SMOTE: Synthetic Minority Over-sampling Technique
- Solution 3: thresholding (threshold-moving, threshold-tuning)

SMOTE: SYNTHETIC MINORITY OVER-SAMPLING TECHNIQUE:

- Synthetic minority instances are created between existing minority instances
- Hyper-parameters:
  - How many neighbors?
  - ratio of minority instances to generate?



SMOTE



Class1
Class2
Original
Downsampled
Upsampled
Hybrid
–2 –1 0 1 2
2
1
0
–1
–2

INDEX

- **Evaluation** for imbalanced datasets
- **Learning/training** with imbalanced datasets:
  - Resampling
  - Thresholding (or threshold-moving, or threshold-tuning)

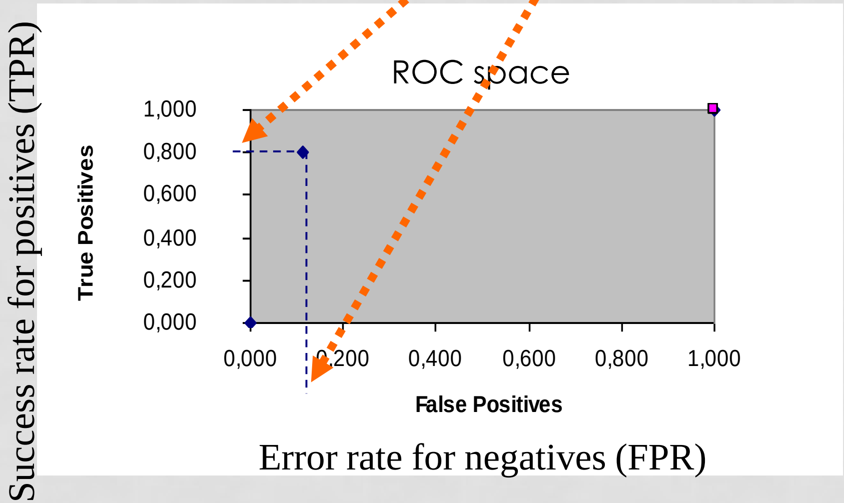
THRESHOLDING (AND ROC CURVES)

- We know that just two numbers, such as TPR and FPR are enough to describe the model.
- A discrete classifier in ROC space is a point.

Classified as

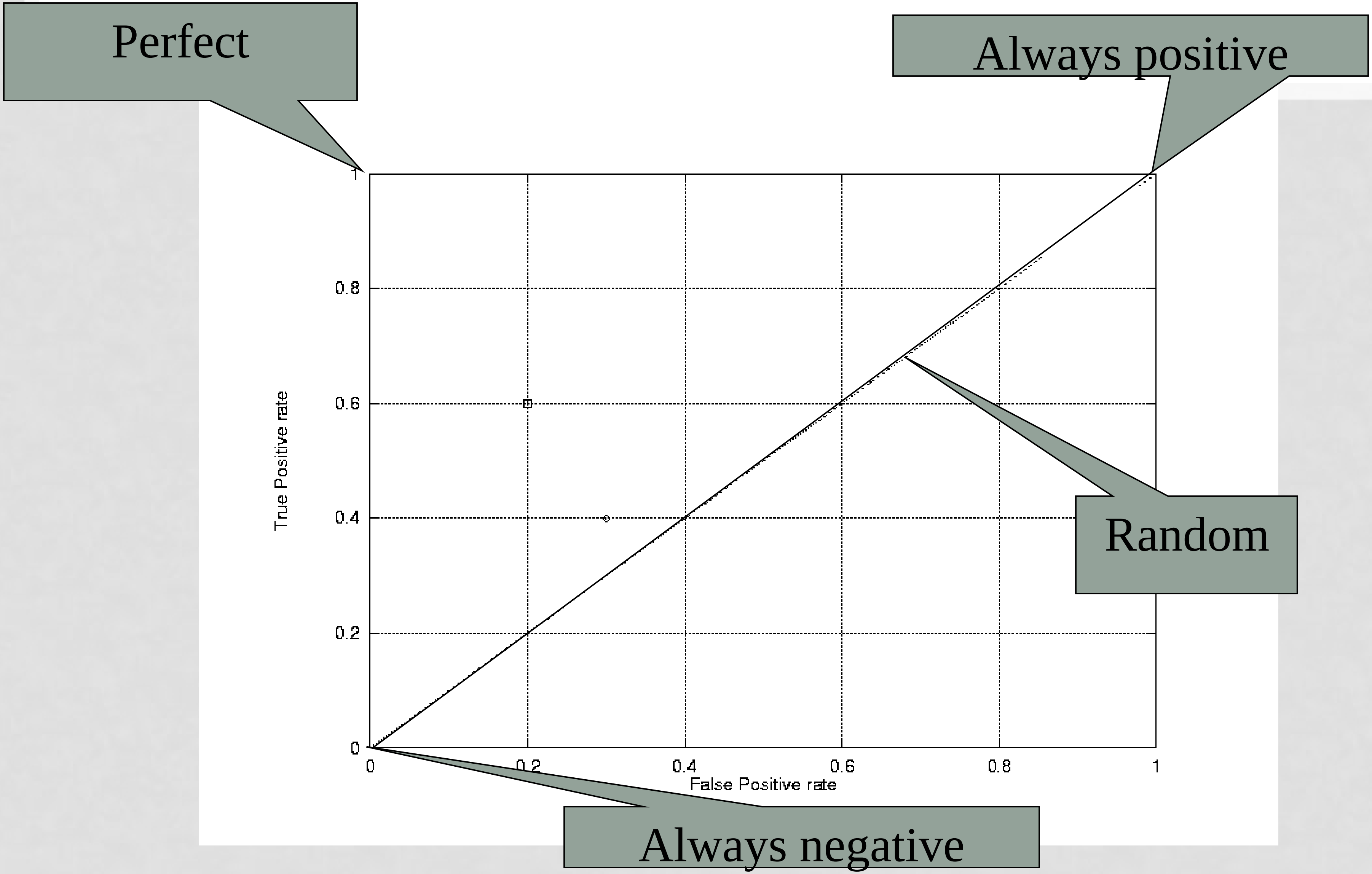
Actual

	POSITIVE	NEGATIVE
positive	**0,8**	**0,2**
negative	**0,121**	**0,879**



	POSITIVE	NEGATIVE
positive	**TPR**	FNR
negative	**FPR**	TNR

ROC SPACE



Perfect
Always positive
Random
Always negative
True Positive rate
False Positive rate
0
0.2
0.4
0.6
0.8
1

WHY DIAGONAL CORRESPOND TO RANDOM CLASSIFIERS?

- TPR = FPR
- Let's define a dummy classifier that **randomly** classifies instances as:
  - positive: 50% probability
  - negative: 50% probability
  - Just by chance it will guess correctly 50% of positives and it will fail with 50% of negatives
    - TPR = 0.5; FPR = 0.5

WHY DIAGONAL CORRESPOND TO RANDOM CLASSIFIERS?

- TPR = FPR
- Let's define a dummy classifier that **randomly** classifies instances as:
  - positive: with 90% probability
  - negative: with 10% probability
  - Just by chance it will guess correctly 90% of positives and 10% of negatives. Therefore, it will fail with 90% of negatives
    - TPR = 0.9; TNR = 0.1; FPR = 1-0.1 = 0.9

- In general, all classifiers for which TPR=FPR (the ones located in the diagonal) are dummy (trivial, naive) classifiers. Our models should do better than that.

DISCRETE (CRISP) CLASSIFIERS VS. SCORING CLASSIFIERS

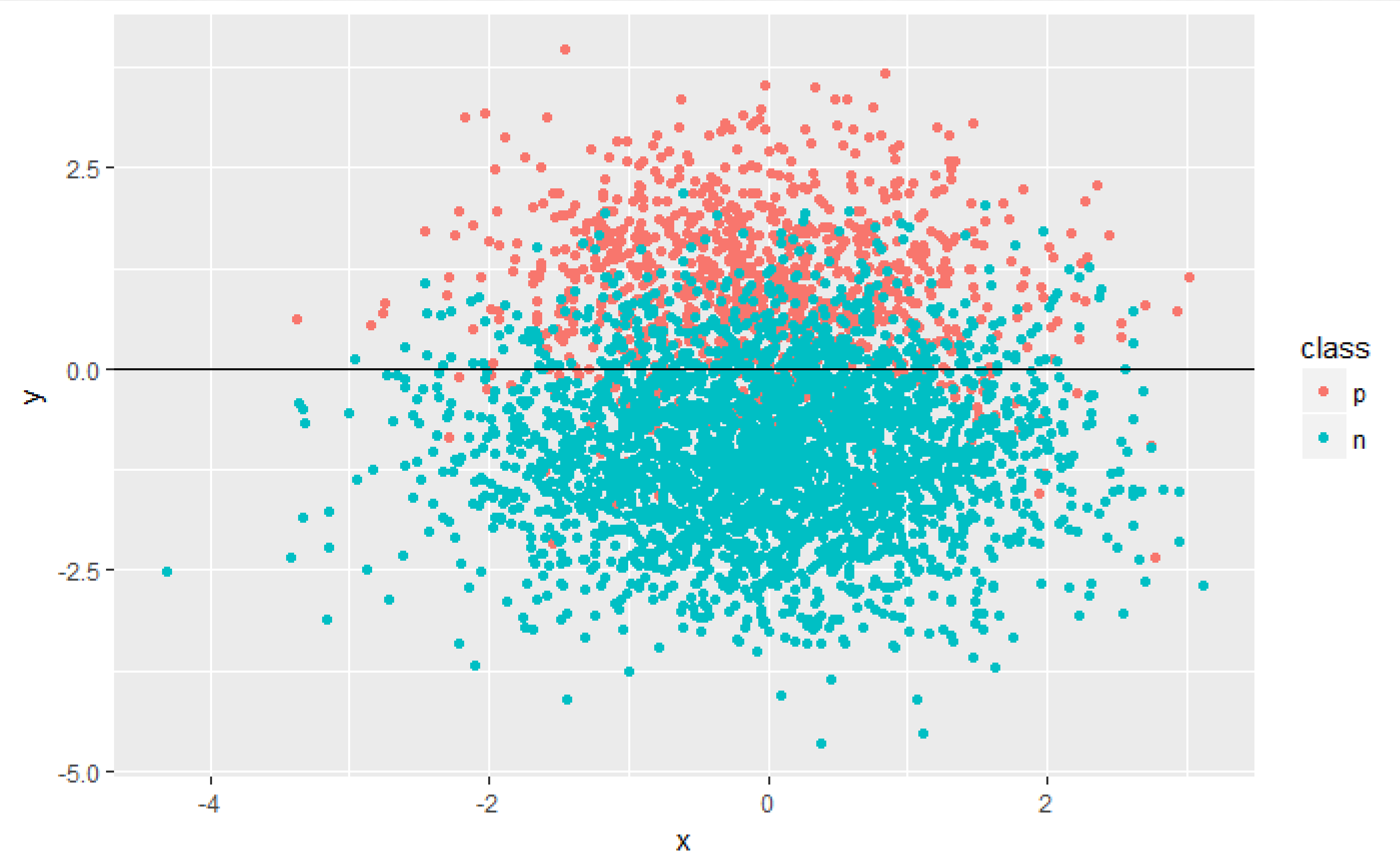
- A discrete (crisp) classifier predicts a class
- A scoring classifier (*sc*) predicts a class together with a real value (or score, g(x)) about the confidence on the prediction.
- Example: Random Forest (it is able to return a probability)
- Not all scores are probabilities
- For example, a linear classifier can inform about the distance (with sign) of the instance to the boundary.
  - score g(x) is between -inf and +inf
  - If the distance is negative and large: inside class 0
  - If the distance is positive and large: inside class 1

SCORING CLASSIFIERS

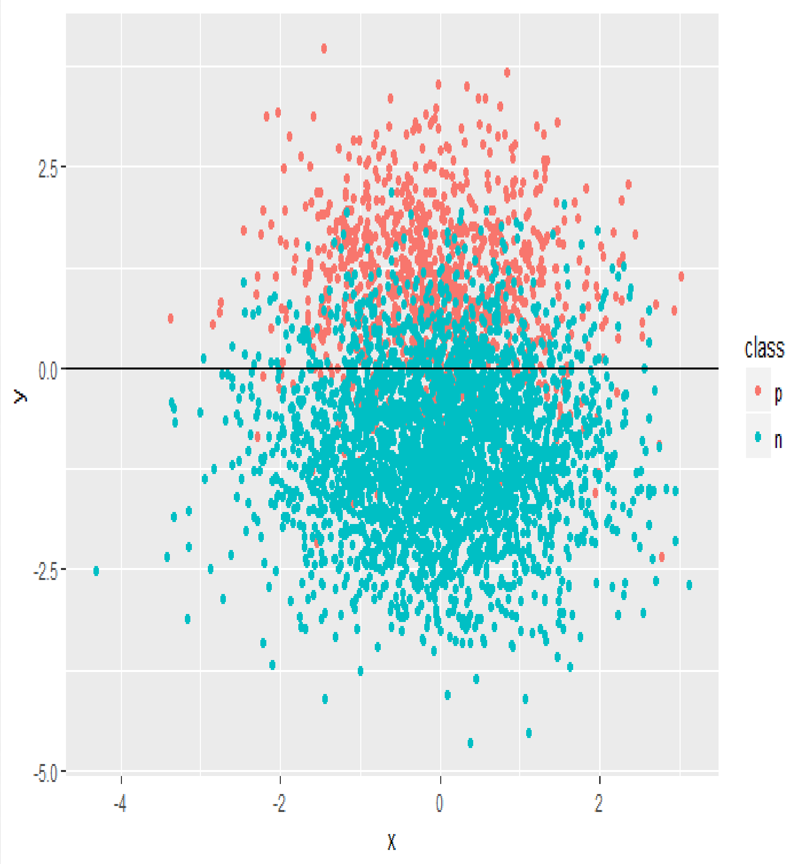
- It is easy to transform a *sc* into a discrete classifier by setting a threshold *t* :
  - If g(x) <= t then predict class 0
  - If g(x) > t then predict class 1
- For different t we obtain a different discrete classifier. That means that an *sc* is a set of points in ROC space (one per *t* value)
- When using probabilities, the default threshold is typically t=0.5:
  - If g(x) <= 0.5 then predict class 0
  - If g(x) > 0.5 then predict class 1
  - but in some cases it might not be the most appropriate threshold

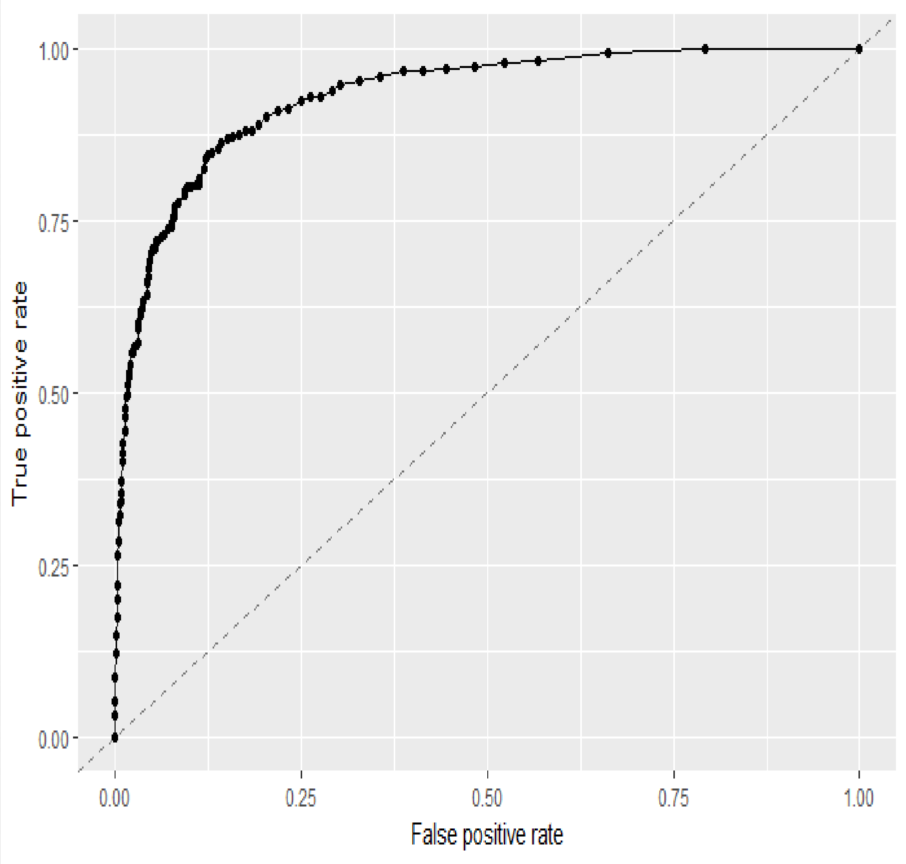
EXAMPLE OF SC

- Linear model
- The score is the distance (with sign) to the linear boundary
- An instance will be classified as positive if score >= threshold
  - (scores are positive in the red región, negative in the blue región)

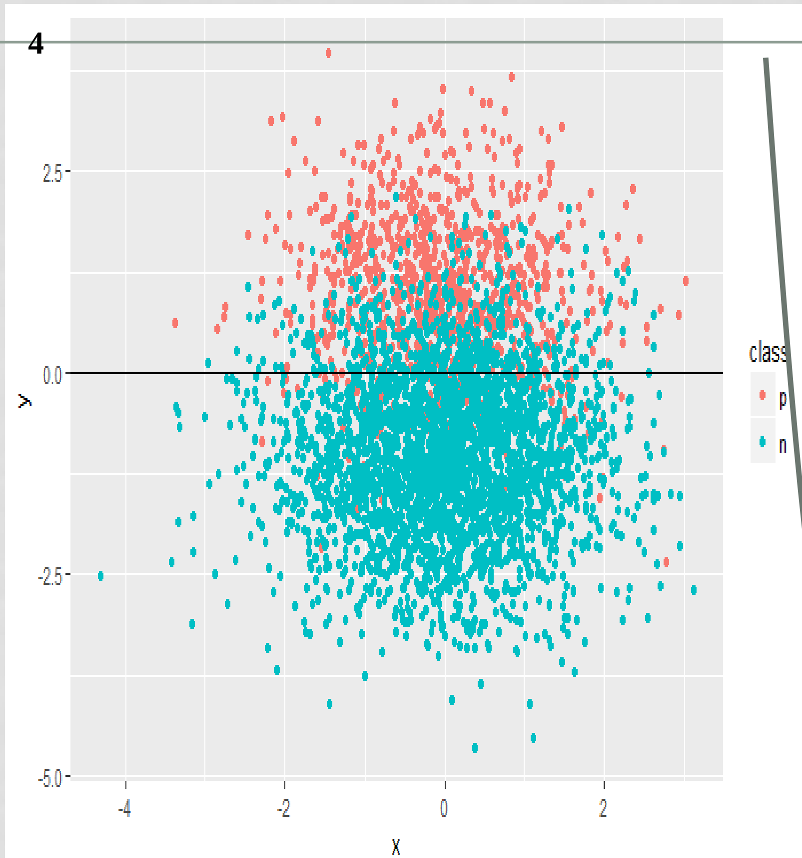


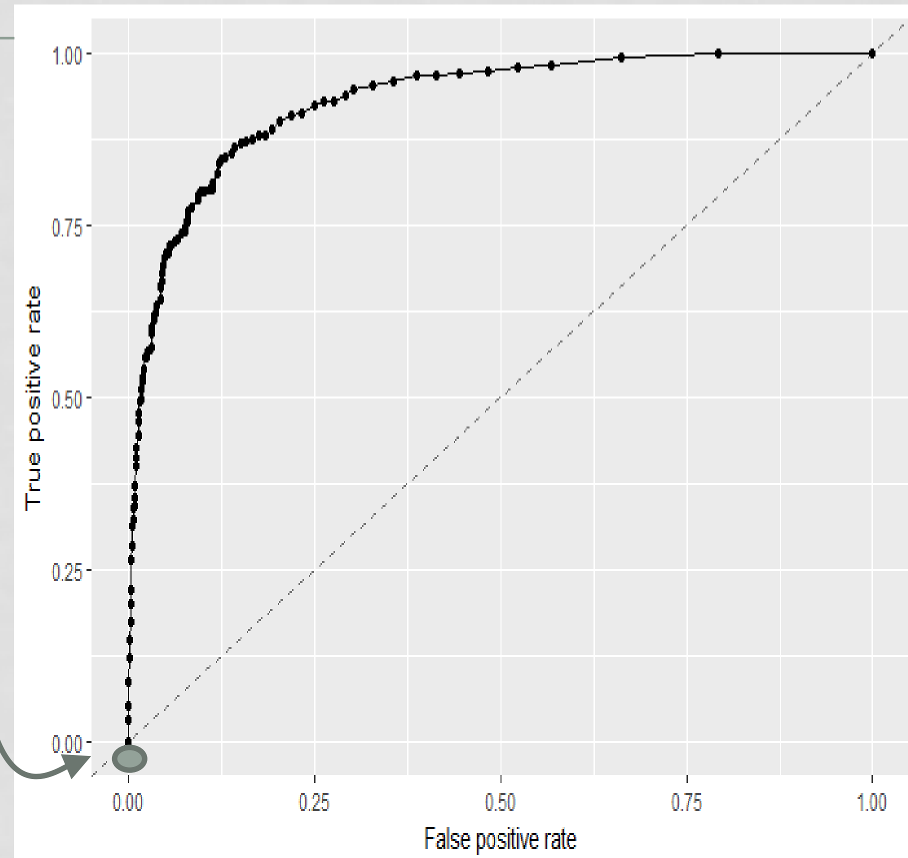
- For each threshold t, there is a point in ROC space





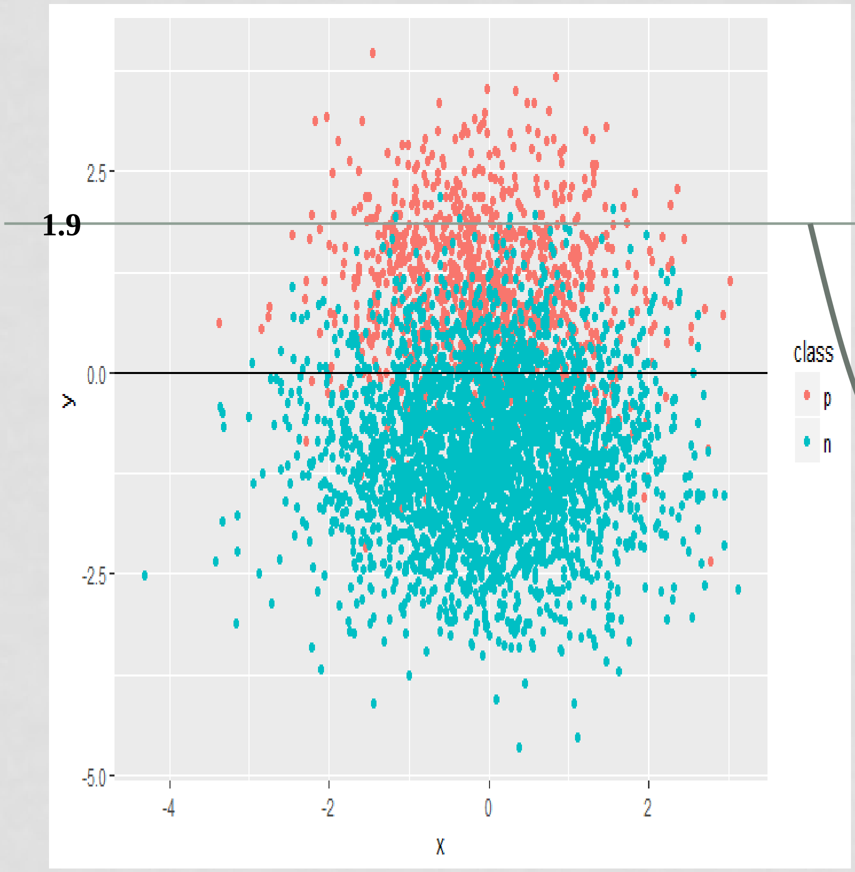
- For each threshold t, there is a point in ROC space
- Let's set t = 4
- If score >=4 then “positive” (red), otherwise “negative” (blue)

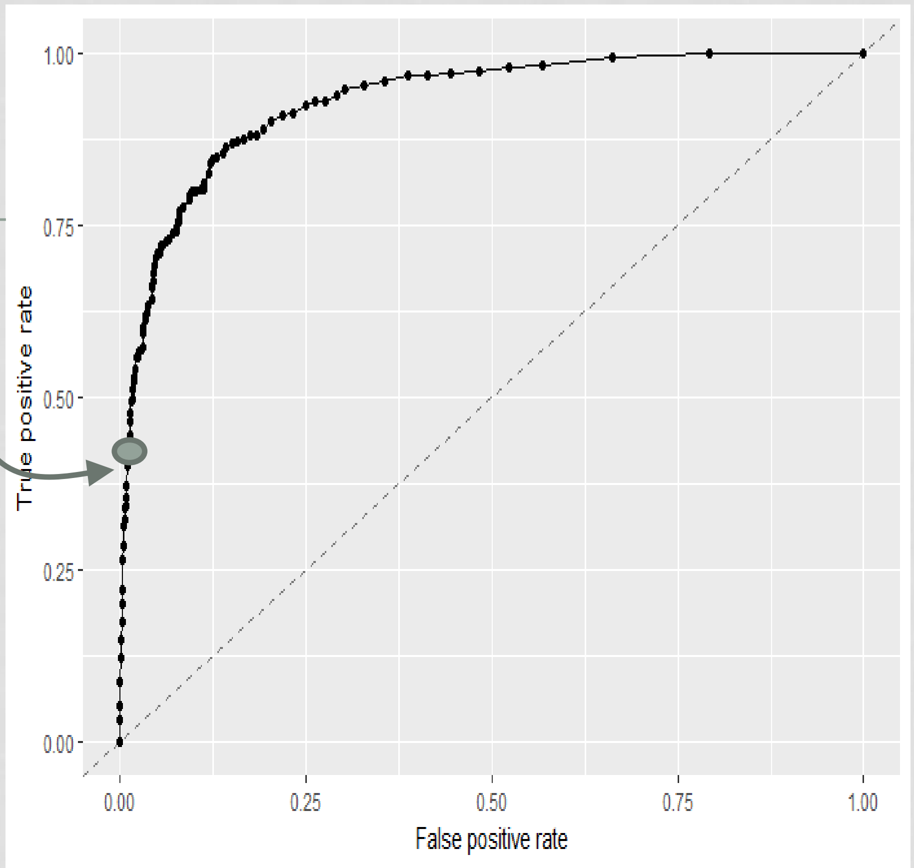




Everything classified as “n”: TPR=0, FPR=0

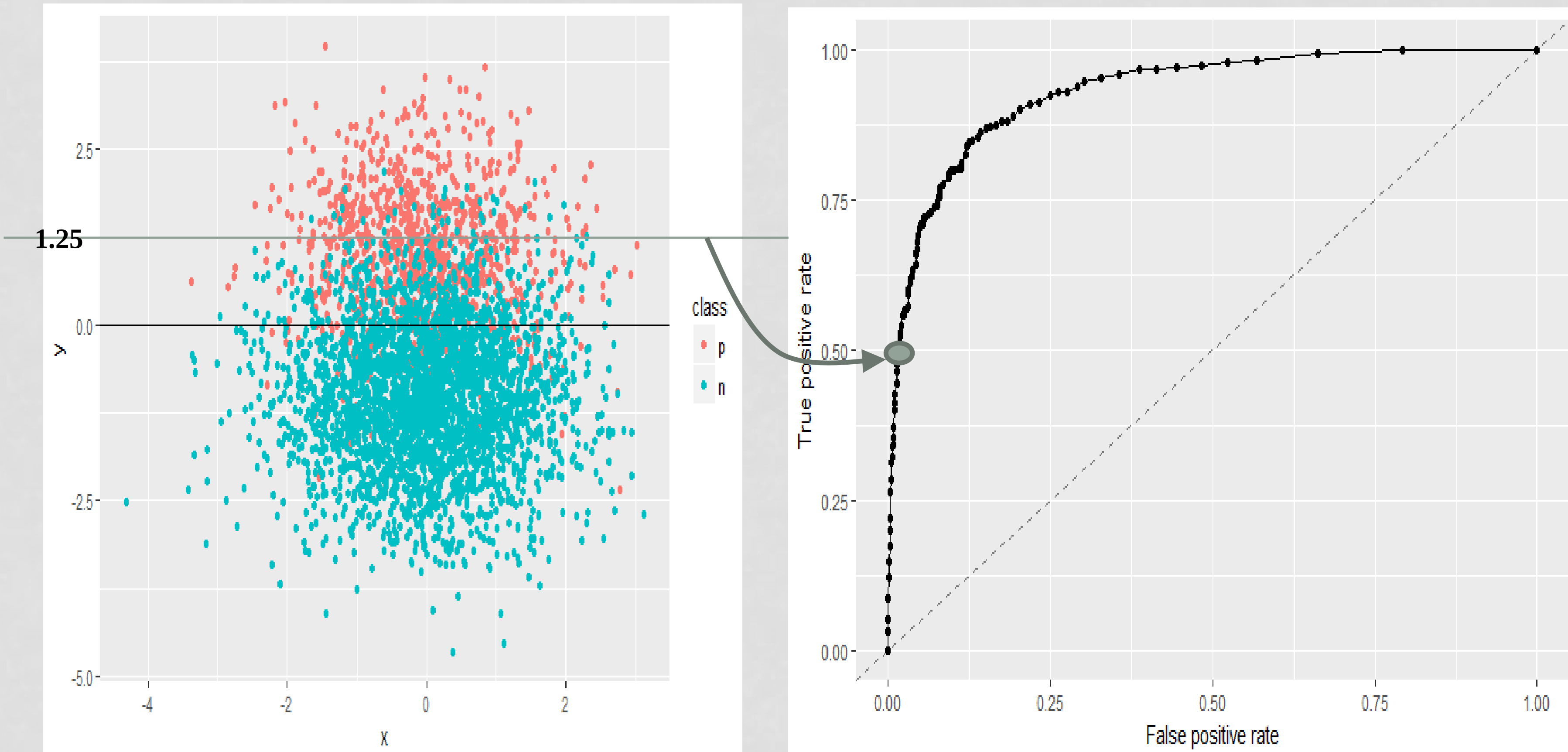
- For each threshold t, there is a point in ROC space
- Let's set t = 1.9
- If score >=1.9 then "positive" (red), otherwise "negative" (blue)





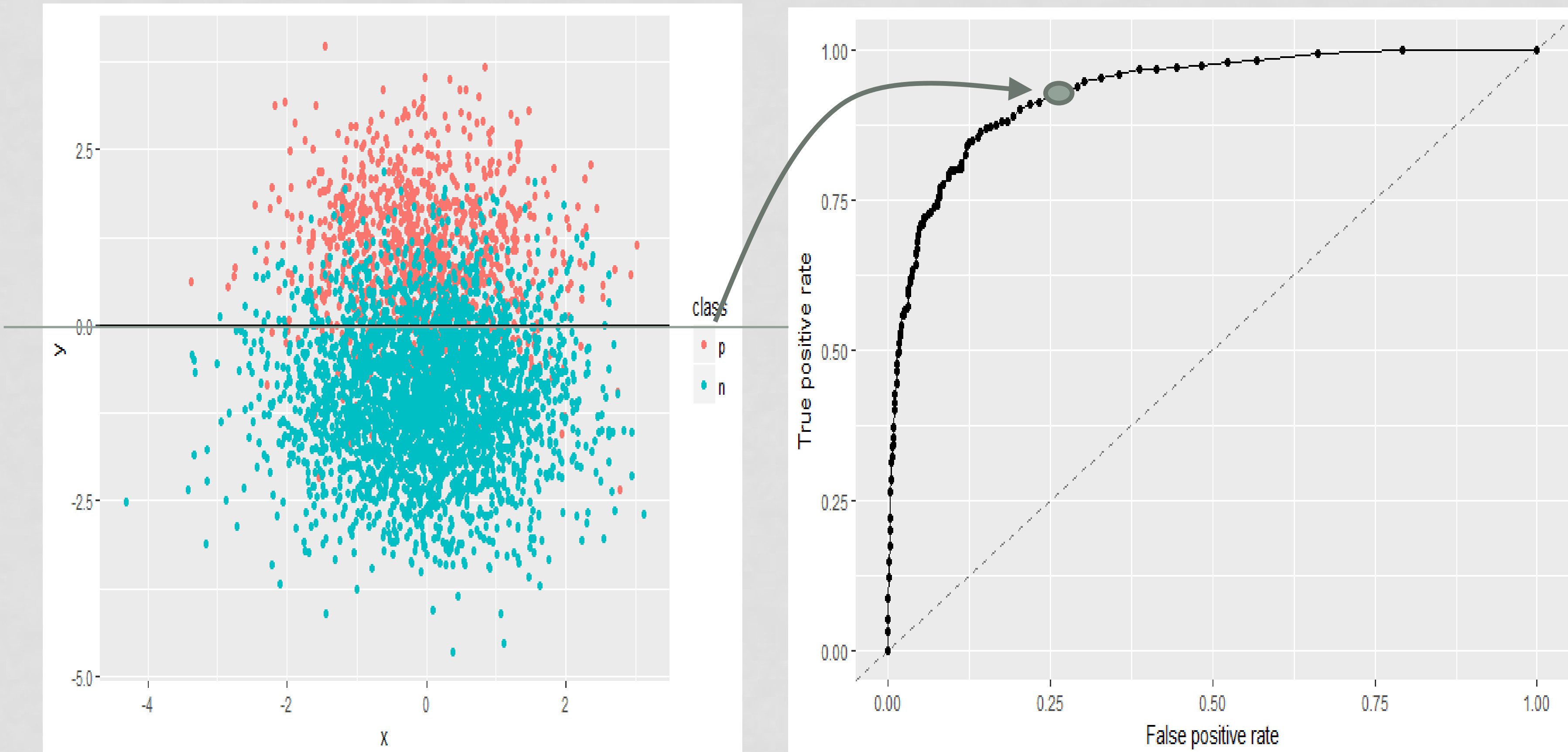
Above threshold: "p"/red, below threshold: "n"/blue.

- For each threshold t, there is a point in ROC space
- If score >=1.25 then “positive” (red), otherwise “negative” (blue)



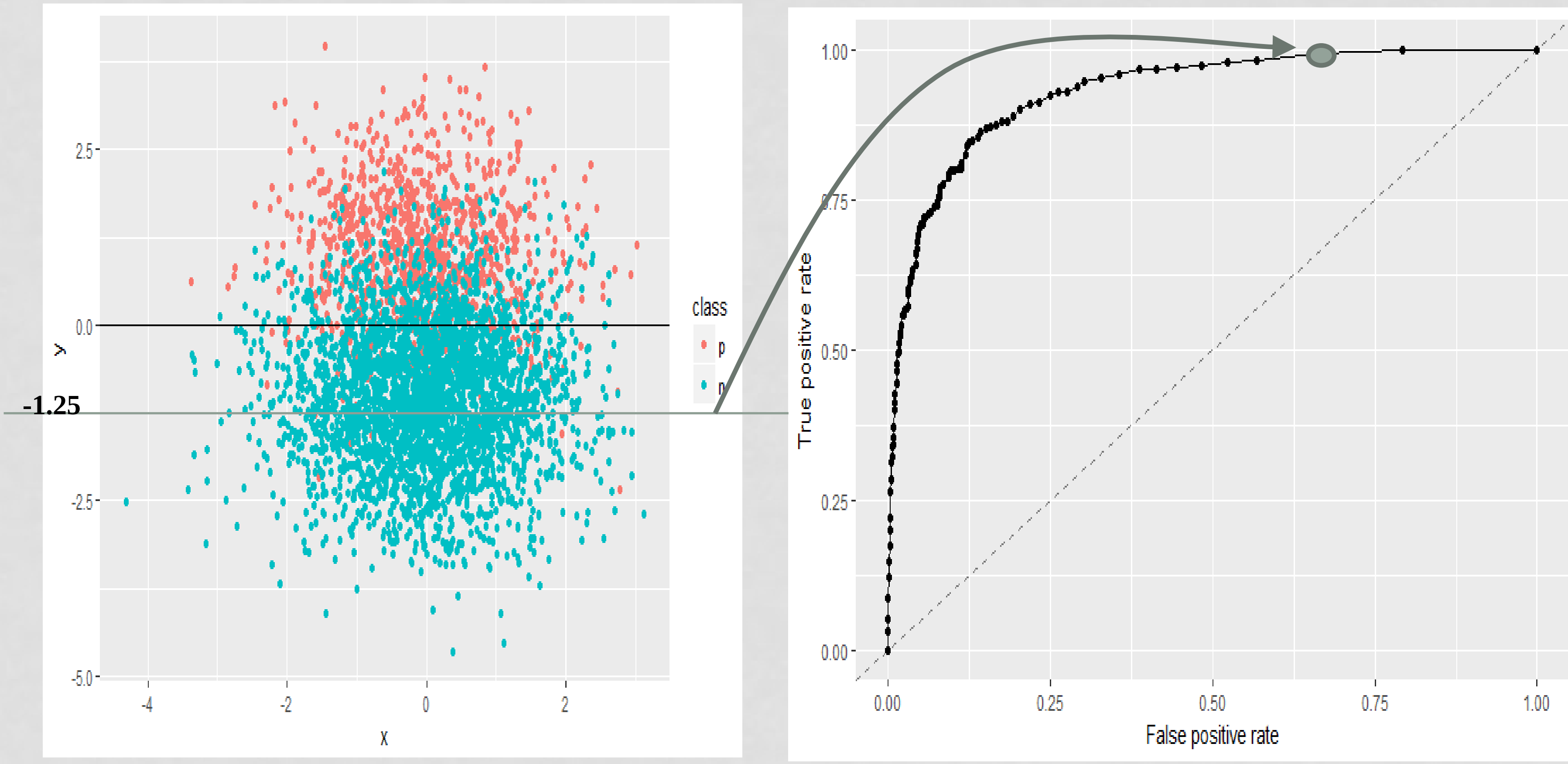
Half of the positives classified as “p” / red, most of the negatives classified as “n” / blue.

- For each threshold t, there is a point in ROC space
- If score >=0 then “positive” (red), otherwise “negative” (blue)



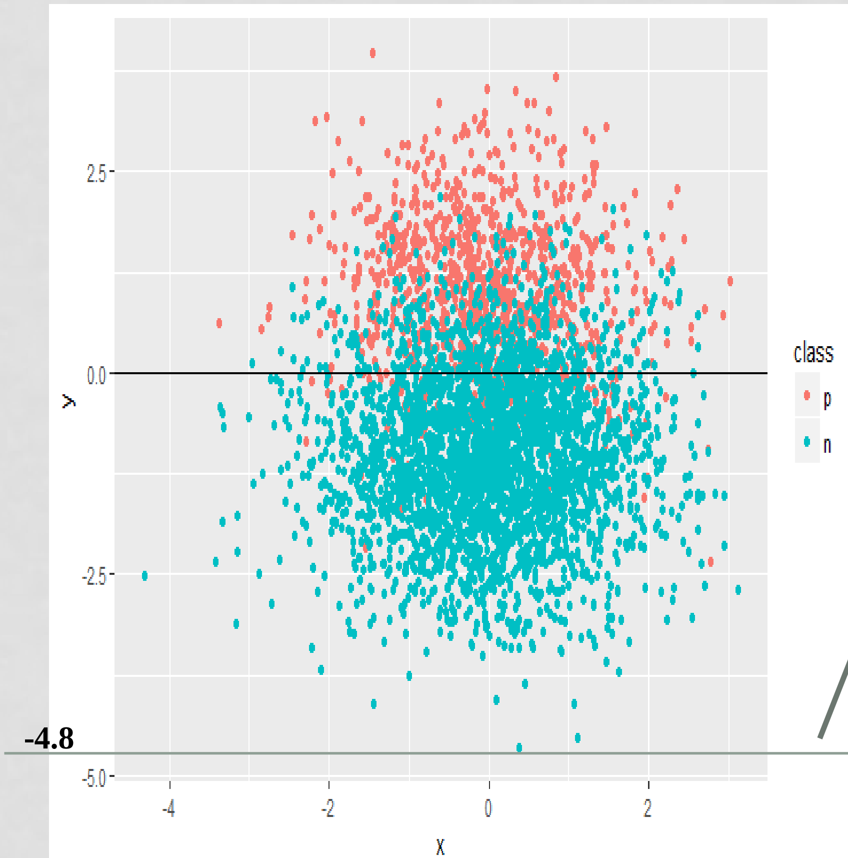
Most positives classified as “p”, but missclassifies some negatives (25%).

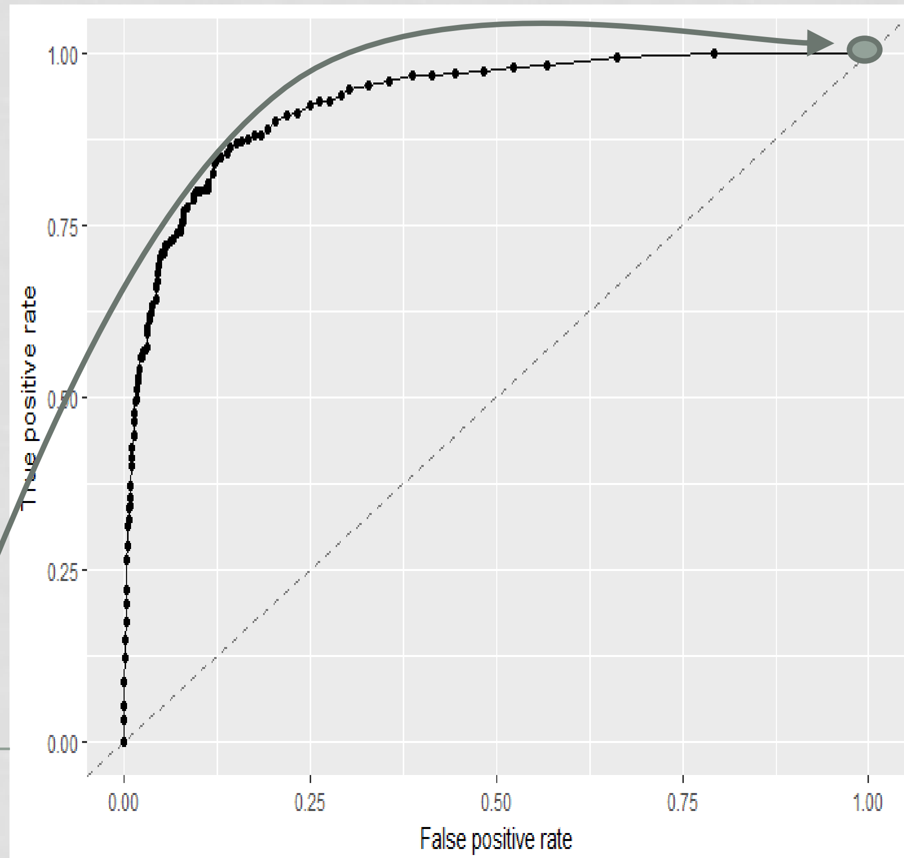
- For each threshold t, there is a point in ROC space
- If score >=-1.25 then “positive” (red), otherwise “negative” (blue)



Almost all positives as “p”, but lots of missclassified negatives.

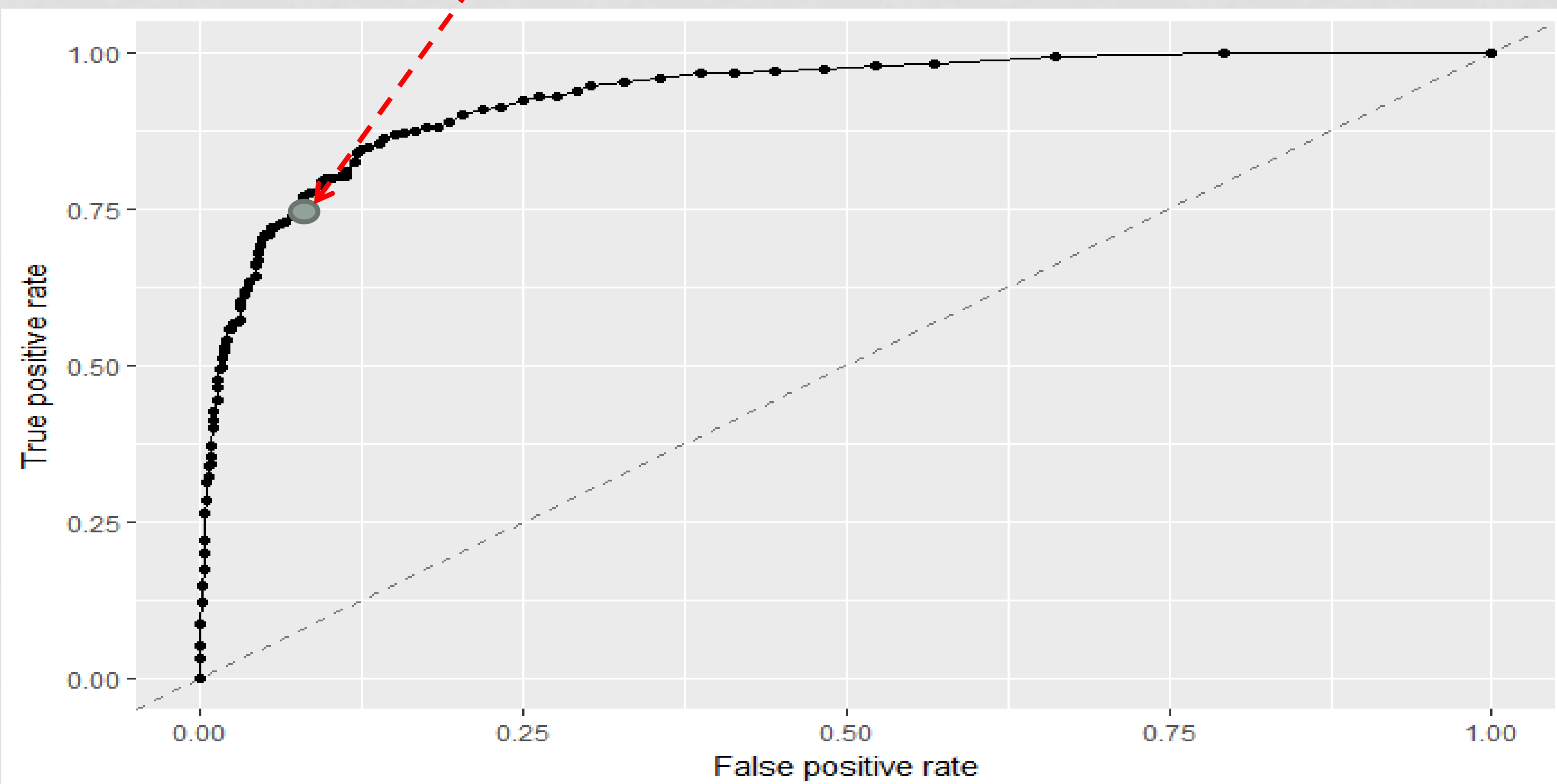
- For each threshold t, there is a point in ROC space
- If score >=-4.8 then “positive” (red), otherwise “negative” (blue)





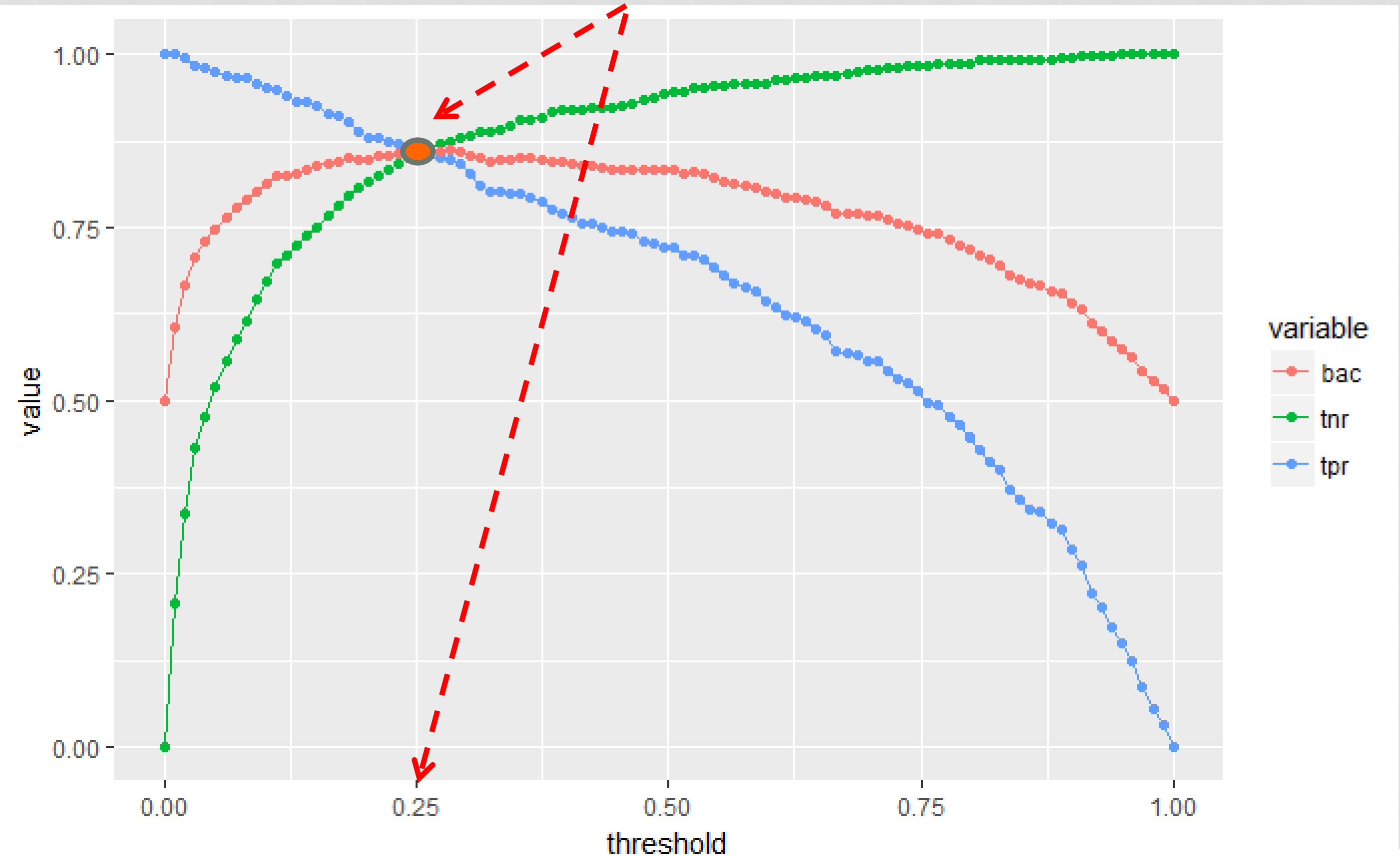
Everything classified as “p”, but missclassifies all negatives.

- We can use the ROC curve to set an appropriate threshold
- For example, we might consider that we need TPR = 75%, and set the appropriate threshold



THRESHOLDING

- In general, we can use "thresholding": tune/set the threshold to optimize some metric. Like "balanced accuracy"



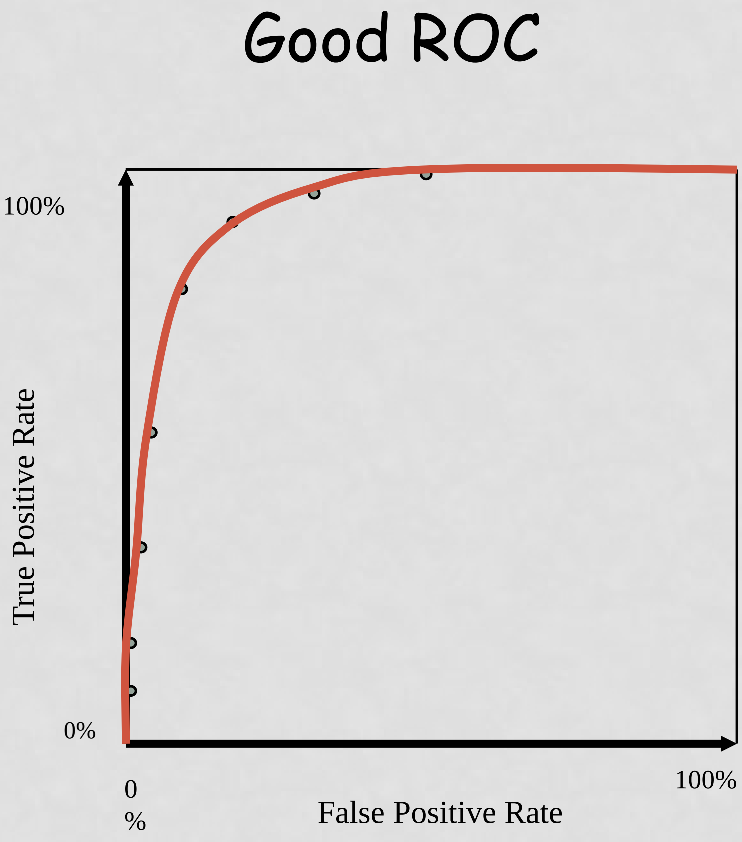
THRESHOLDING

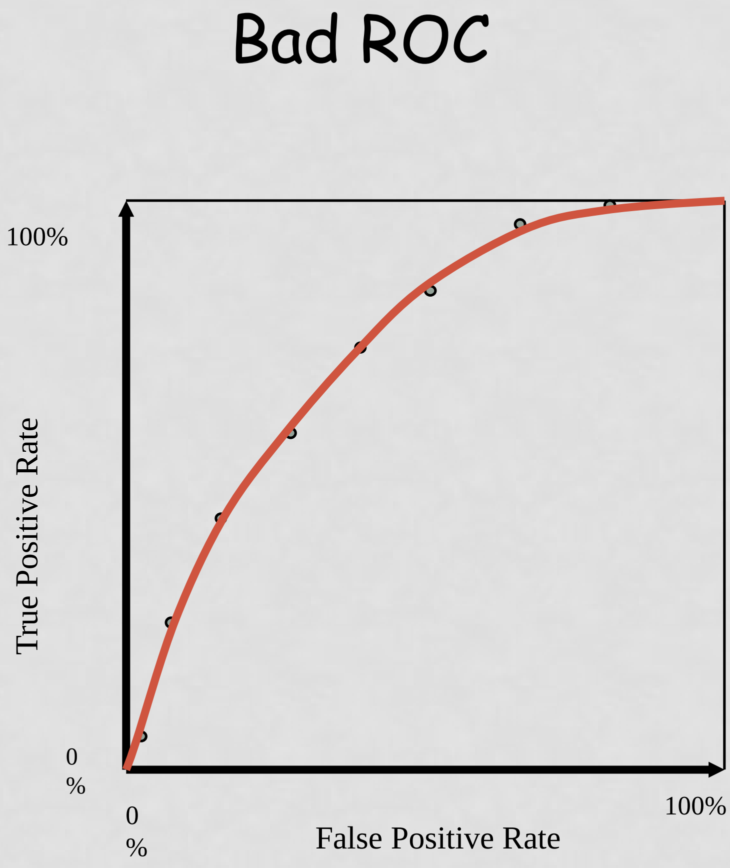
- Optimizing the threshold is not automated in sklearn
- In order to get scores out of a classifier for some validation data you can use:
  - scores = model.predict_proba(validation_set)
    - If the method is able to return probabilities
  - scores = model.decision_function(validation_set)
    - If the method is not able to return probabilities
    - SVM's can do both (using the probability=True parameter), but obtaining probabilities is very time consuming
  - instead of predict()
- You can try different thresholds (0, 0.1, 0.2, ..., 0.9, 1.0) and select the one that optimizes some metric (like balanced_accuracy_score) on the validation set.

THE AUC METRIC

- In addition to thresholding, ROC curves are useful for evaluating models, if they are scoring classifiers.

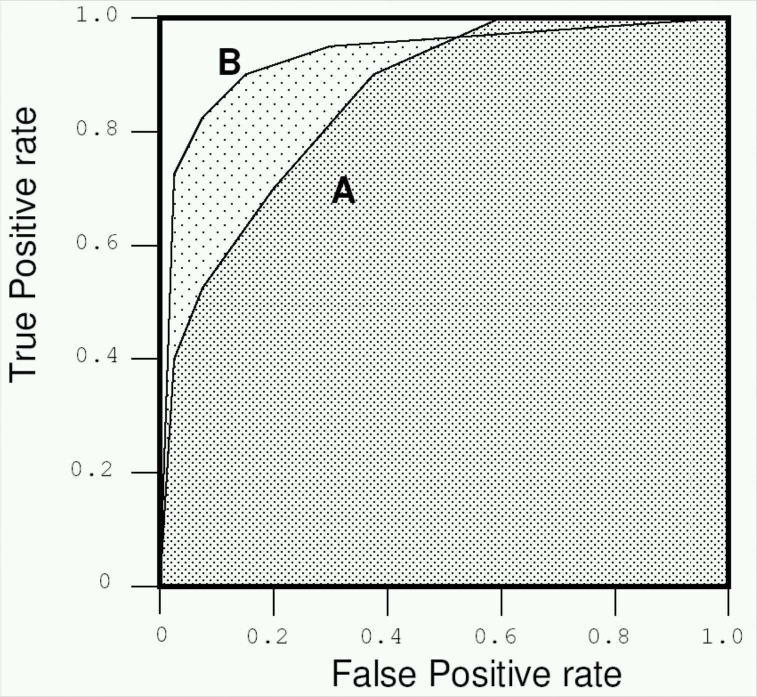
THE AUC METRIC





THE AUC METRIC
(AREA UNDER THE ROC CURVE)

- Select the model with larger AUC
- AUC is useful for evaluating scoring classifiers on imbalanced datasets



© Tom Fawcett

AUC FOR MORE THAN TWO CLASSES

- Construct the ROC curve for each pair of classes, and compute the average AUC

$$AUC_{HT} = \frac{1}{c(c-1)} \sum_{i=1}^{c} \sum_{j=1, j<i}^{c} AUC(i, j)$$

OVER/UNDERSAMPLING IN SCIKIT LEARN

- New module required: imbalanced
  - conda install -c conda-forge **imbalanced-learn**