COMP 432 Machine Learning

Classifier Metrics

Computer Science & Software Engineering
Concordia University, Fall 2021

Beyond *Accuracy*

- Scikit-learn classifiers report "accuracy" by default
  - e.g. `my_classifier.score(X, y_true)` returns accuracy
- Depending on application and/or the data set, "accuracy" may not even be a meaningful measure.
- Two main aspects to consider:
  - **Application-driven asymmetric costs** that explicitly assign a different "penalty" for making different kinds of mistakes, based on some knowledge of stakeholder priorities
  - **Data-driven asymmetric costs**, implicitly caused by class imbalance in the training data, e.g., some classes are easier to collect or to label than other classes.

WANT THIS TO INFLUENCE PREDICTIONS
(Design loss carefully, with stakeholders involved)

DON'T WANT THIS TO INFLUENCE PREDICTIONS
(Try different class weightings as part of hyperparameter search.)

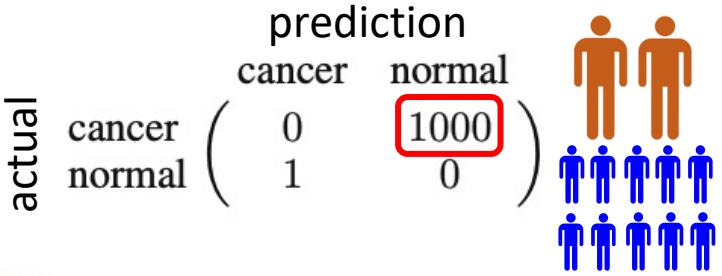
- Examples of application-driven asymmetric loss:
  - In early-stage cancer screening, the cost of "*failing to detect a case of cancer*" may be higher than the cost of "flagging a patient who turns out not to have cancer."
  - A bank may decide that the cost of "*lending to a person who defaults on their debt*" is much higher than the cost of "*denying a loan to person who would have paid their debt*"

- Examples of data-driven class imbalance:
  - You cobbled together a training set having 50% normal and 50% cancer, even though real population is highly imbalanced.
  - The bank you work for used their own extremely-conservative rules for many decades, so there are almost no "defaulted on loan" training cases.

Don't let data-driven imbalances prevent you from finding a model that best-respects your application-driven ones.

Example of “application-driven asymmetric loss” from Bishop

Figure 1.25 An example of a loss matrix with elements L_{kj} for the cancer treatment problem. The rows correspond to the true class, whereas the columns correspond to the assignment of class made by our decision criterion.



1.5.2 Minimizing the expected loss

For many applications, our objective will be more complex than simply minimizing the number of misclassifications. Let us consider again the medical diagnosis problem. We note that, if a patient who does not have cancer is incorrectly diagnosed as having cancer, the consequences may be some patient distress plus the need for further investigations. Conversely, if a patient with cancer is diagnosed as healthy, the result may be premature death due to lack of treatment. Thus the consequences of these two types of mistake can be dramatically different. It would clearly be better to make fewer mistakes of the second kind, even if this was at the expense of making more mistakes of the first kind.

```
X = [[0], [1], [2], [3]]
y = [-1, -1, +1, -1]
```

```
svm = SVC().fit(X, y)
svm.score(X, y)
```

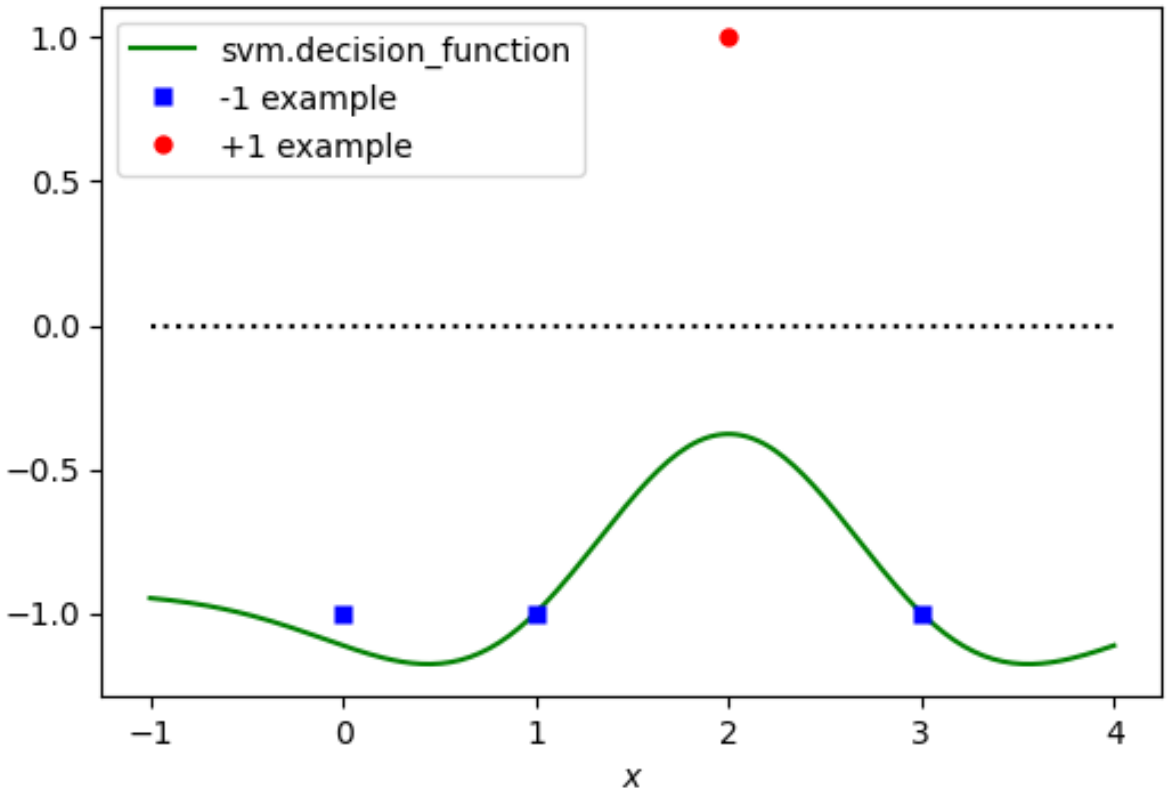
```
0.75
```

3 out of 4 correct using SVM with default configuration (RBF, gamma='scale', C=1.0)

```
confusion_matrix(svm.predict(X), y)
```

```
array([[3, 1],
       [0, 0]])
```

3 "true positives"
1 "false negative"
0 "false positives"
0 "true negatives"



```
X = [[0], [1], [2], [3]]
y = [-1, -1, +1, -1]
```

```
svm = SVC(class_weight={-1: 1.0,
                        +1: 1.5}).fit(X, y)
svm.score(X, y)
```

Same SVM, but trained with stronger penalty for classifying +1 as -1

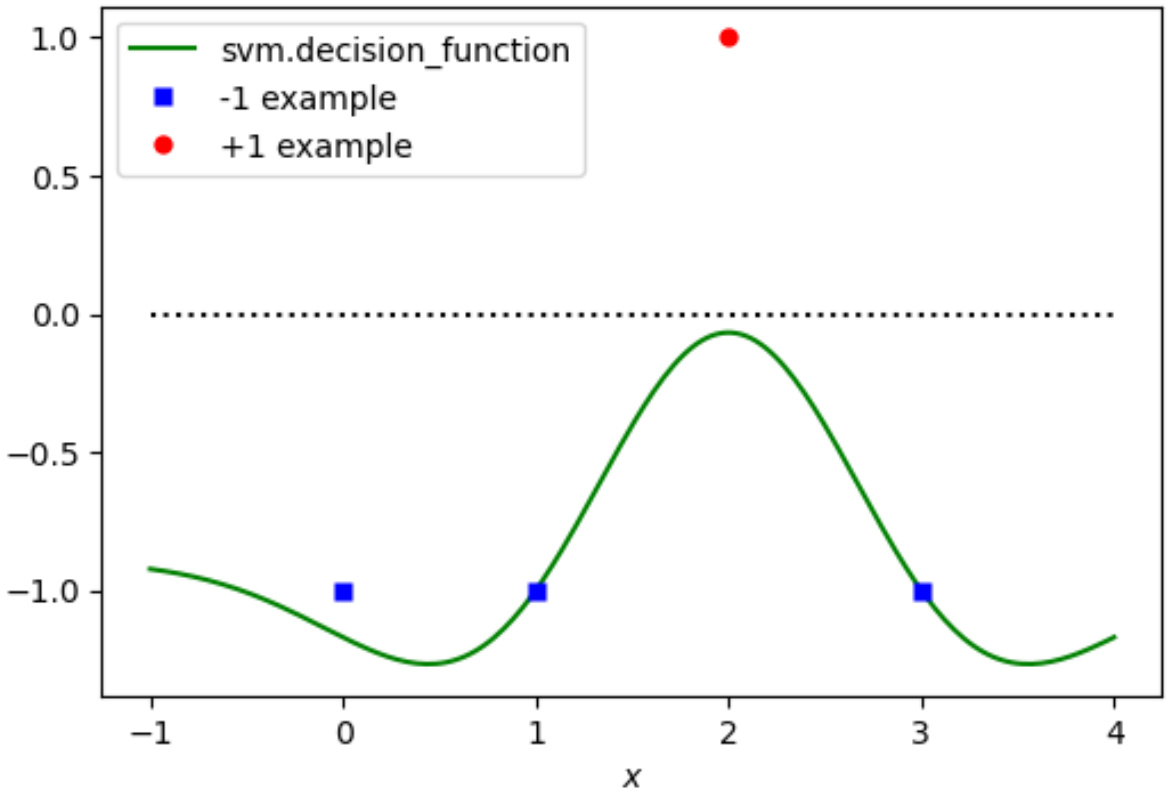
```
0.75
```

Notice that default "score" did not change. It is *unweighted,* all mistakes still considered equal.

```
confusion_matrix(svm.predict(X), y)
```

```
array([[3, 1],
       [0, 0]])
```

For cancer screening, the doctors will agree that this kind of mistake is much worse. Even though "accuracy" is not directly paying attention to class_weight, the training algorithm is.



```
X = [[0], [1], [2], [3]]
y = [-1, -1, +1, -1]
```

```
svm = SVC(class_weight={-1: 1.0,
                        +1: 1000.0}).fit(X, y)
svm.score(X, y)
```

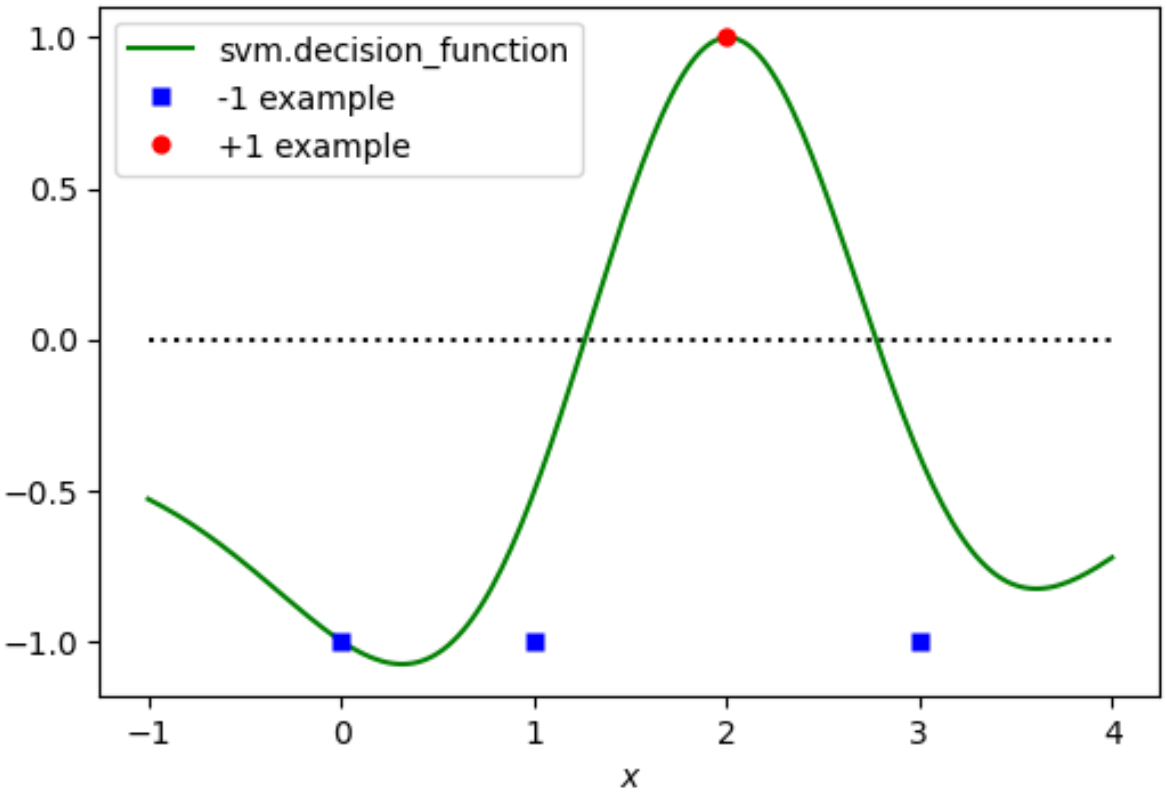
```
1.0
```

Score changed only because *predictions* changed, indirectly because of class_weight

```
confusion_matrix(svm.predict(X), y)
```

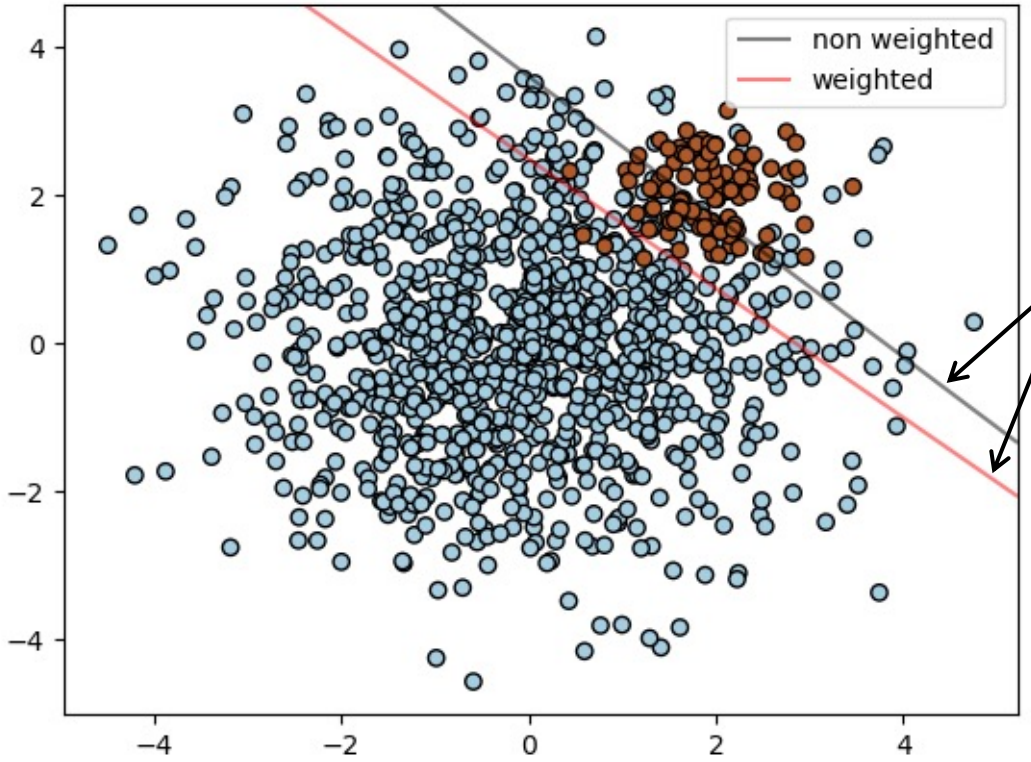
```
array([[3, 0],
       [0, 1]])
```

3 “true positives”
0 “false negative”
0 “false positives”
1 “true negatives”



Example of unbalanced classes

```
sklearn.svm.SVC(kernel='linear', class_weight={-1: 1.0,
                                               +1: 10.0})
```



"I want a *false negative* to cost 10x than a false positive, *for the purpose of training*."

class_weight influences decision boundary.

Which is better? Always use your application-driven success metric to decide. It is possible that the optimal class_weight to use during training is not identical to the class_weight you are using to measure test-time success.

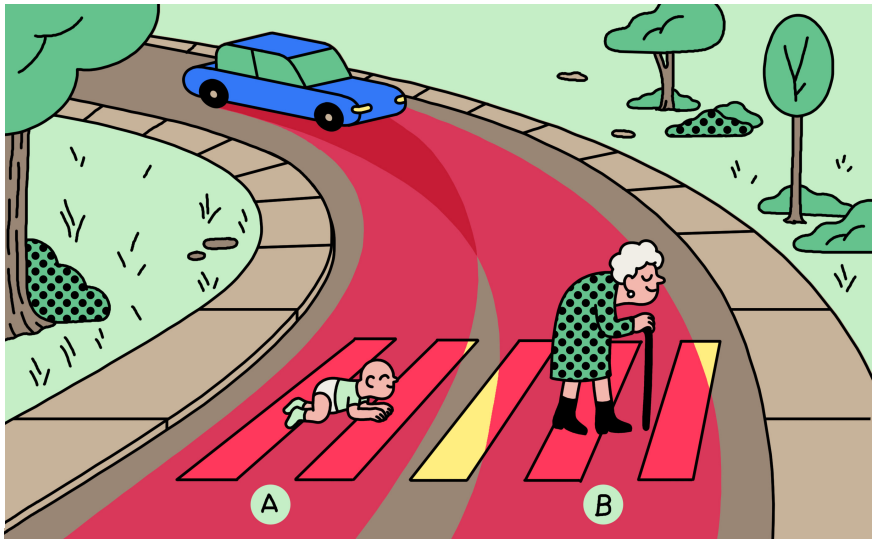
Must I try multiple class_weights? In practice often not. You can achieve a similar "shifting" effect on the decision boundary by simply adjusting the threshold on your score function, whatever class_weight your trained with. (Basis of computing ROC curves.)

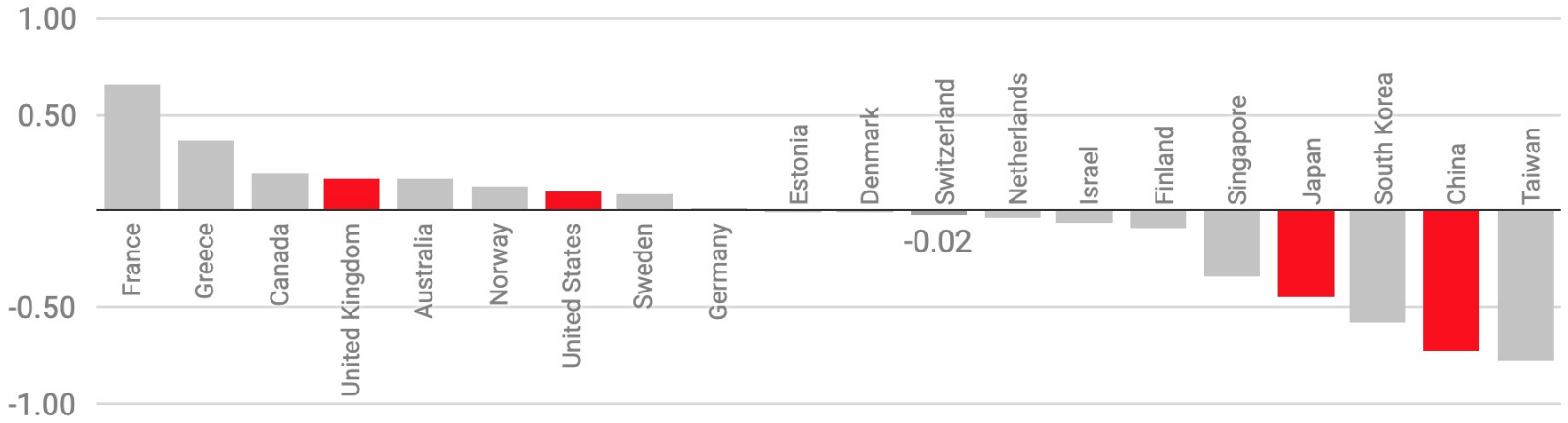
Image source: https://scikit-learn.org/stable/modules/svm.html#unbalanced-problems

Should a self-driving car kill the baby or the grandma? Depends on where you're from.

The infamous "trolley problem" was put to millions of people in a global study, revealing how much ethics diverge across cultures.

Image credit: SIMON LANDREIN





A comparison of countries piloting self-driving cars: If the bar is closer to 1, respondents placed a greater emphasis on sparing the young; if the bar is closer to -1, respondents placed a greater emphasis on sparing the old; 0 is the global average.

https://www.technologyreview.com/2018/10/24/139313/a-global-ethics-study-aims-to-help-ai-solve-the-self-driving-trolley-problem/

SHARKEYE

Screenshot of https://www.sharkeye.org/

We fly drones on standardized survey flights to capture video footage of nearshore waters where sharks may be present.

We use Salesforce AI to scan the video to automatically detect great white sharks in the footage with 95% accuracy.

What does this “95%” mean in practice?
Absent details, this claim means *nothing*!

2x2 "confusion matrix"

known \ prediction	*n*	*p*
n	T_n	F_p
p	F_n	T_p

Binary classifiers

- Classifier predicts *positive* or *negative* for each example
- **Goal:** Evaluate performance w.r.t. a <u>particular</u> data set
- Relevant quantities of that data set:
  - N_p # actual positives in the data set
  - N_n # actual negatives in the data set
- Relevant quantities of classifier *applied* to data set:
  - T_p # actual positives correctly classified ("*true positives*")
  - F_n # actual positives incorrectly classified ("*false negatives*")
  - F_p # actual negatives incorrectly classified ("*false positives*")
  - T_n # actual negatives correctly classified ("*true negatives*")
- Obviously $N_p = T_p + F_n$ and $N_n = T_n + F_p$

Also called "**recall**" or "**sensitivity**"

Binary classifier performance

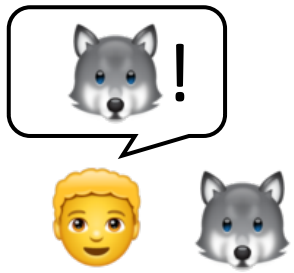
- The ***true positive rate*** (TPR) is important quantity:

$$TPR = \frac{T_p}{T_p + F_n} = \frac{T_p}{N_p}$$

- **"The probability that a randomly-selected <u>actual positive</u> will receive a <u>positive prediction</u>."**

targets $\mathbf{t} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$

predictions $\hat{\mathbf{t}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}^T$



$$TPR = \frac{5}{5+1} = 0.833$$

$$p(\text{cries wolf} \mid \text{wolf})$$

Chance that boy *cries wolf* when there *actually is a wolf*

Binary classifier performance

- The ***false positive rate*** (FPR) is also important:

$$FPR = \frac{F_p}{F_p + T_n} = \frac{F_p}{N_n}$$

- **"The probability that a randomly-selected <u>actual negative</u> will receive a <u>positive prediction</u>."**

targets $\mathbf{t} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$

predictions $\hat{\mathbf{t}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}^T$

$$FPR = \frac{2}{2+2} = 0.5$$

$$p(\text{cries wolf} \mid \neg\text{wolf})$$

Chance that boy *cries wolf* when there *isn't a wolf*

Binary classifier performance

- The ***precision*** (P) is defined to be:

$$P = \frac{T_p}{T_p + F_p} \leftarrow \text{not equal to } N_p \text{ or } N_n$$

(Unlike TPR/FPR, don't need to know total positives or total negatives, only how many *predicted* positives were correct.)

- **"The probability that a randomly-selected positive prediction will turn out to be an actual positive."**

targets $\mathbf{t} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T$

predictions $\hat{\mathbf{t}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}^T$

$$P = \frac{5}{5+2} = 0.714$$


?

$p(\text{wolf} \mid \text{cries wolf})$

Chance that there *actually is a wolf* given that boy *cries wolf*

These measures used for *any* predictive test, whether or not ML involved in the prediction

22q11.2 deletion syndrome

Diagnosis and Testing

MENU

How accurate is the noninvasive prenatal testing (NIPT) blood test for 22q11.2 deletion syndrome?

Noninvasive prenatal testing (NIPT) is a screening option for pregnant women, which gives a pregnant woman the option of having a blood test to determine the chances that the fetus has a chromosomal disorder. It was created to screen for Down syndrome, but many additional conditions have been added to the test including 22q11.2 deletion syndrome. Unfortunately, NIPT for 22q11.2 deletion syndrome is not as accurate as it is for Down syndrome. A 2015 study found a high number of false positives (80%) when testing fetuses for 22q11.2 deletion syndrome through NIPT. This implies that only 20% of the pregnancies with a positive result for 22q11.2 deletion syndrome actually had 22q11.2 deletion syndrome. Parents are recommended not to make decisions about the pregnancy solely based off of NIPT results. Diagnostic testing directly on a fetal DNA sample obtained with chorionic villus sampling (CVS) or amniocentesis can confirm or rule out 22q11.2 deletion syndrome accurately. More research needs to be conducted to better understand and improve the accuracy of NIPT for detection of 22q11.2 deletion syndrome.

informal sense of "accurate"

You should think: wait, does this mean 80% FPR, or 20% precision?

... then "Aah, they meant 20% *precision. That's not good."*

At 20% precision, it may harmful to even *perform* this screening test at all. Why? Because, *in this application*, prospective parents would feel pressured to then perform the *diagnostic* test. The diagnostic test is *invasive* (needle into placenta) and has a 1% chance of causing *miscarriage*. There is significant cost to acting on this low-precision test.

https://www.thinkgenetic.com/diseases/22q112-deletion-syndrome/diagnosis-testing/2484

$$TPR = \frac{T_p}{T_p + F_n} = \frac{T_p}{N_p} \qquad FPR = \frac{F_p}{F_p + T_n} = \frac{F_p}{N_n}$$

Receiver Operating Characteristic

- The "ROC" curve measures classifier performance using only two computed values (*FPR*, *TPR*)

History [edit]

The ROC curve was first used during World War II for the analysis of radar signals before it was employed in signal detection theory.[40] Following the attack on Pearl Harbor in 1941, the United States army began new research to increase the prediction of correctly detected Japanese aircraft from their radar signals. For these purposes they measured the ability of a radar receiver operator to make these important distinctions, which was called the Receiver Operating Characteristic.[41]

$$TPR = \frac{T_p}{T_p + F_n} = \frac{T_p}{N_p} \qquad FPR = \frac{F_p}{F_p + T_n} = \frac{F_p}{N_n}$$

Receiver Operating Characteristic

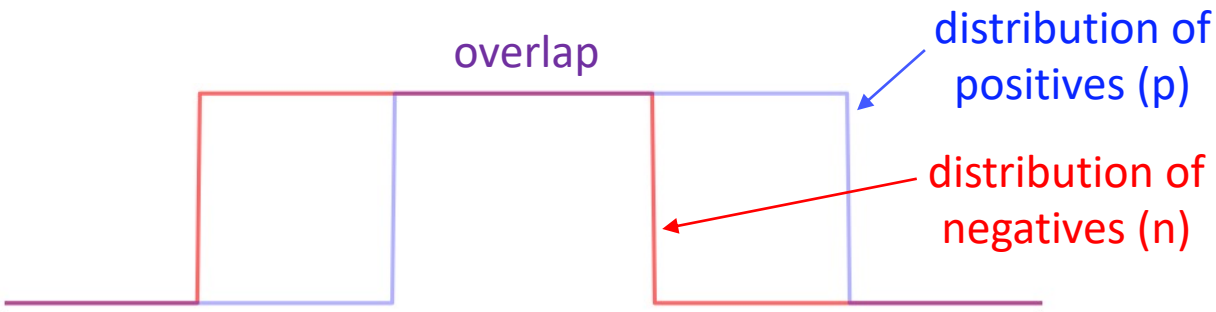
- The “ROC” curve measures classifier performance using only two computed values (*FPR*, *TPR*)
- A classifier with perfect performance, no errors of any kind, would be at point (0, 1) in ROC space
  - *FPR* = 0: no known negatives are classified as positives
  - *TPR* = 1: all known positives are classified as positives
- Classifiers often output a *score* that is *thresholded*:
  - SVM score = signed dist from hyperplane, threshold @ 0.0
  - Logistic regression score = probability, threshold @ 0.5
- Other choices of threshold give different (*FPR*, *TPR*)
  - threshold @ $-\infty$ gives (1, 1), @$+\infty$ gives (0, 0)

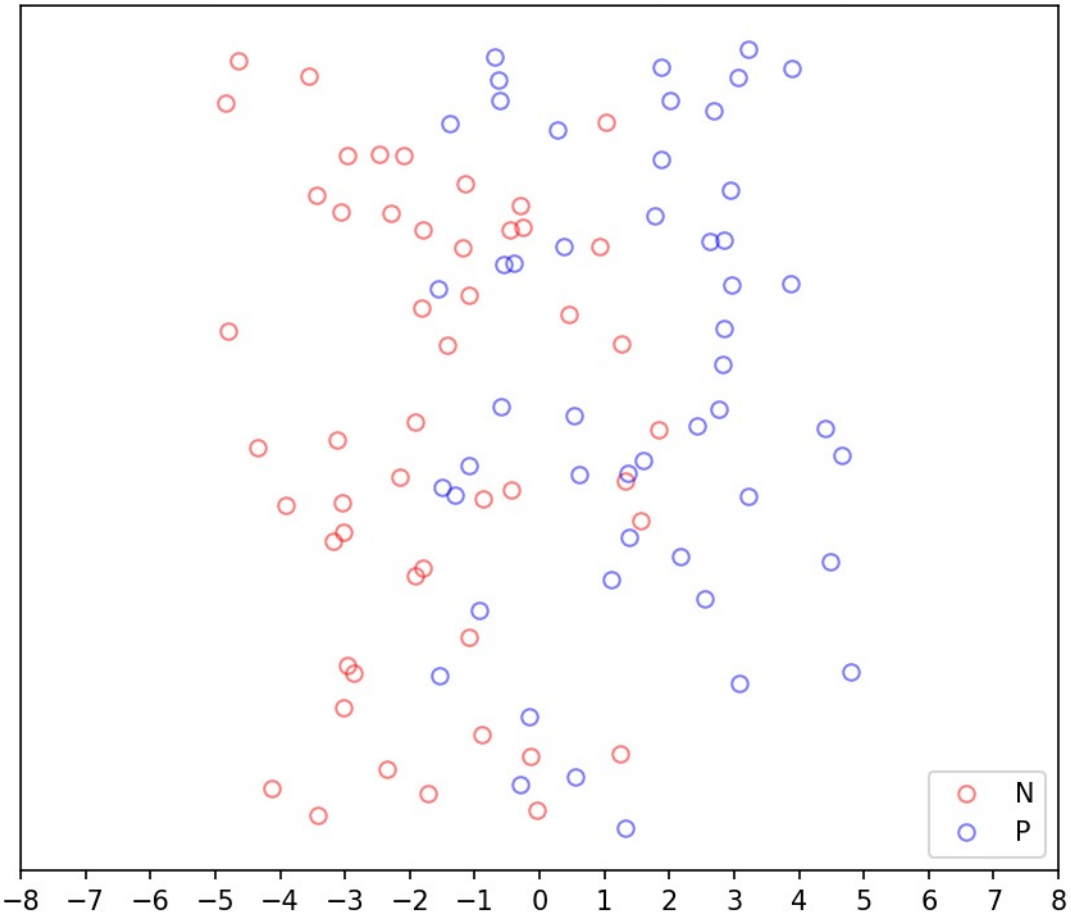
$$TPR = \frac{T_p}{T_p + F_n} = \frac{T_p}{N_p} = R \qquad P = \frac{T_p}{T_p + F_p}$$

Precision and Recall

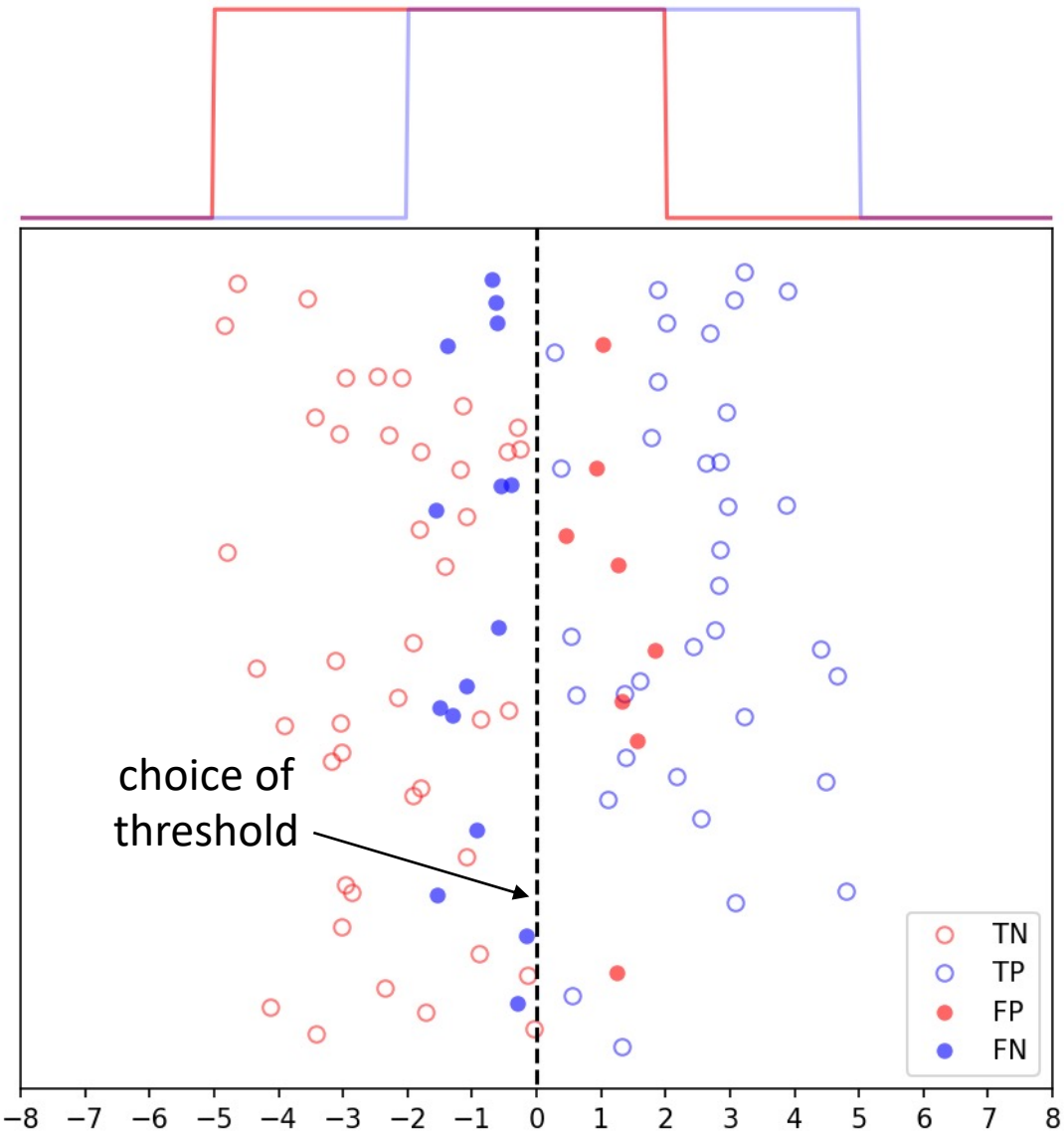
- Instead of computing (*FPR*, *TPR*) for a classifier we can compute (*R*, *P*) instead; same as (*TPR*, *P*)
- Perfect performance would be at point at (1, 1)
  - *R* = 1: all known positives are predicted as positive
  - *P* = 1: all predicted positives are known to be positive
- Example: score documents by relevance, threshold as "relevant" (positive) or "irrelevant" (negative)
  - *R* = fraction of *all relevant documents* that were *selected*
  - *P* = fraction of *selected documents* were actually *relevant*
- Again, different thresholds will give different (*R*, *P*)
  - threshold @ $-\infty$ (predict all positive) gives $(1, \frac{N_p}{N})$
  - threshold @ $+\infty$ (predict all negative) gives $(0, \frac{0}{0})$

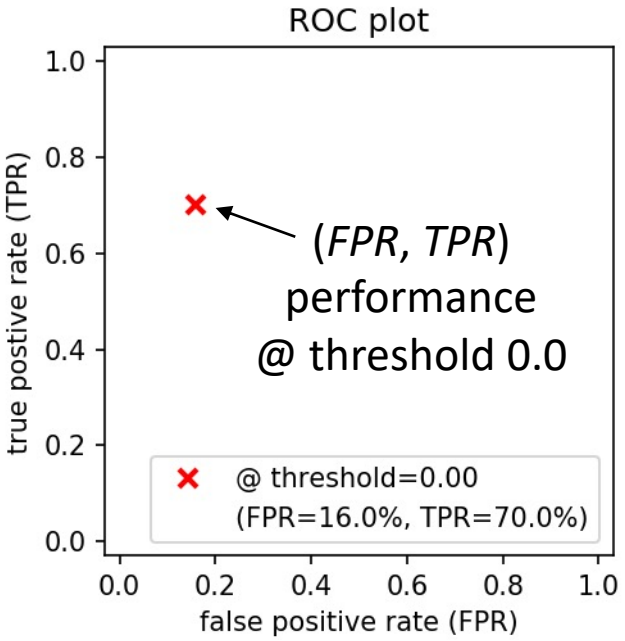
Example of ROC & PR

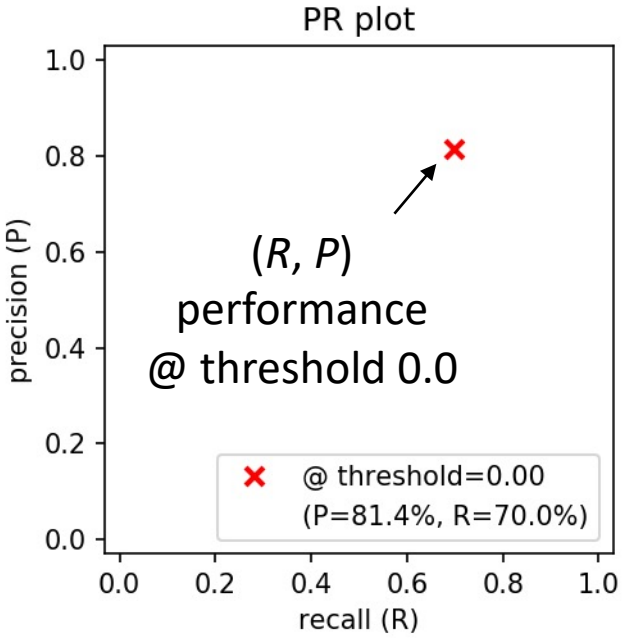




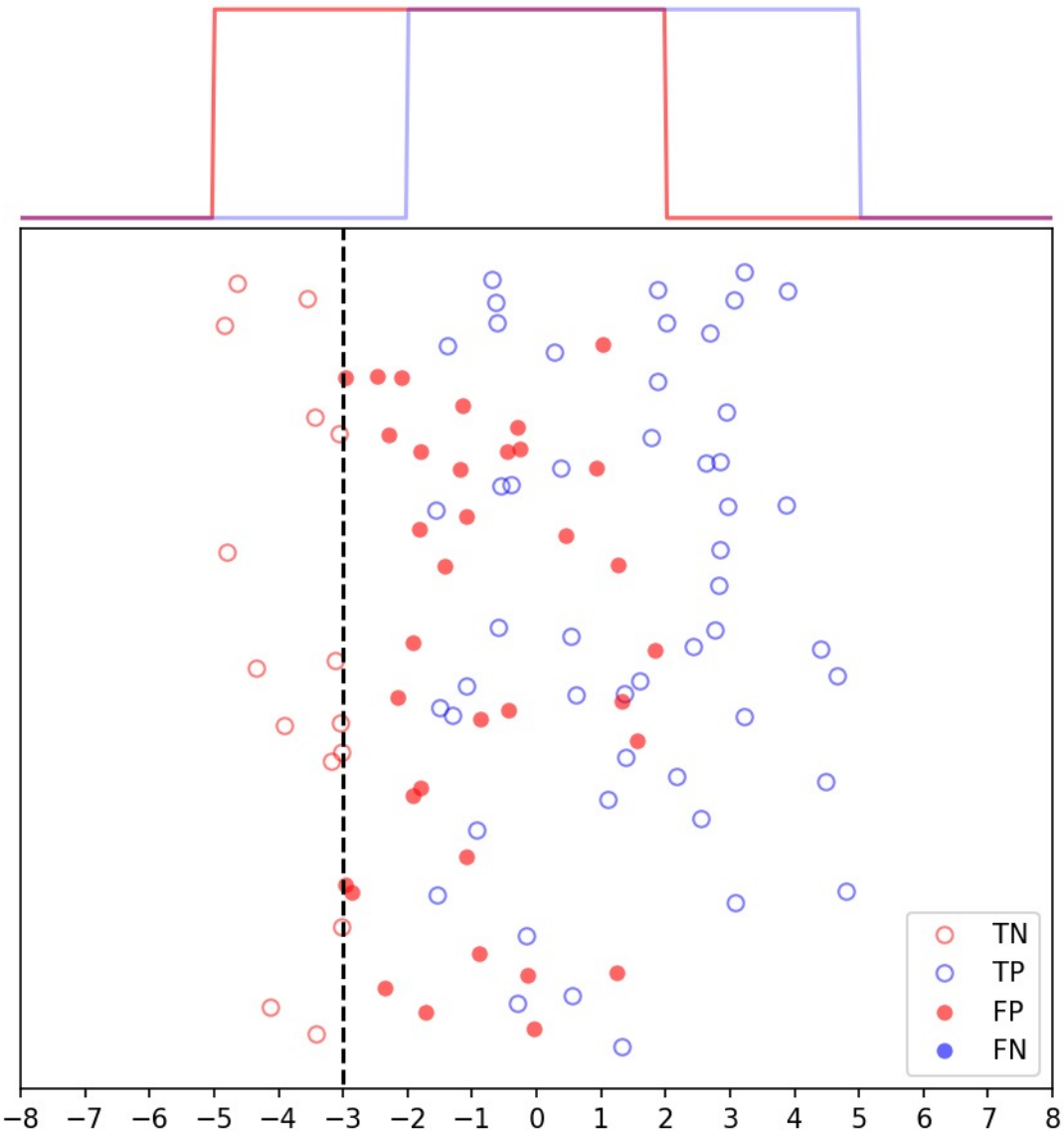
Example of ROC & PR

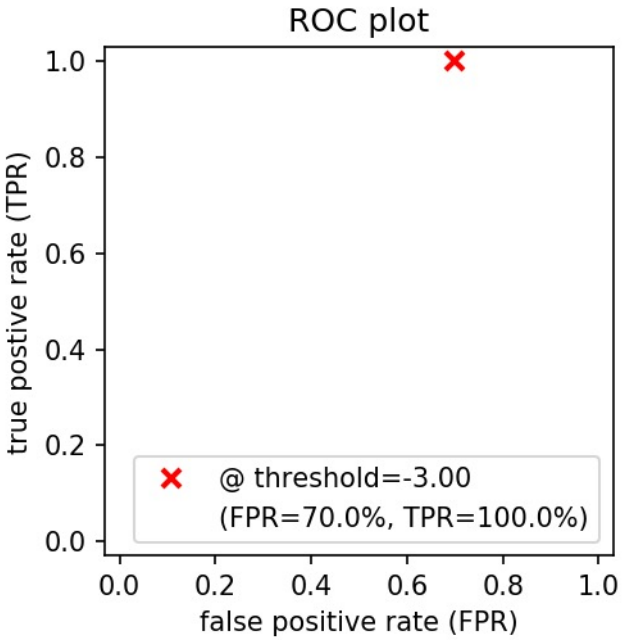


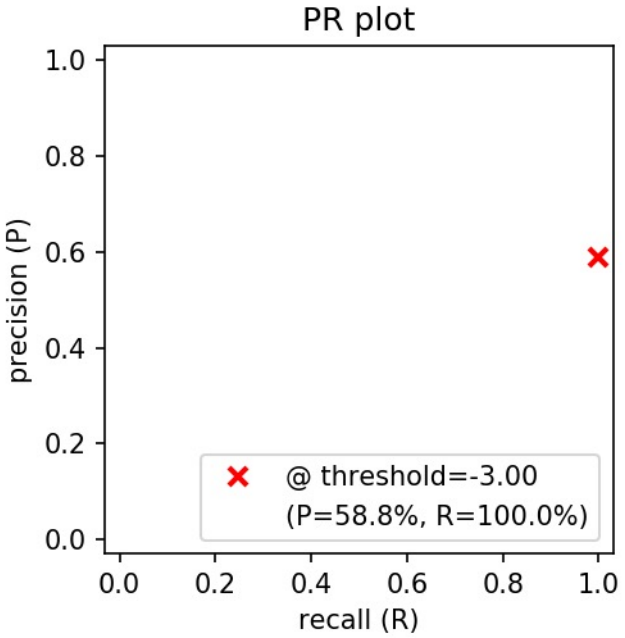




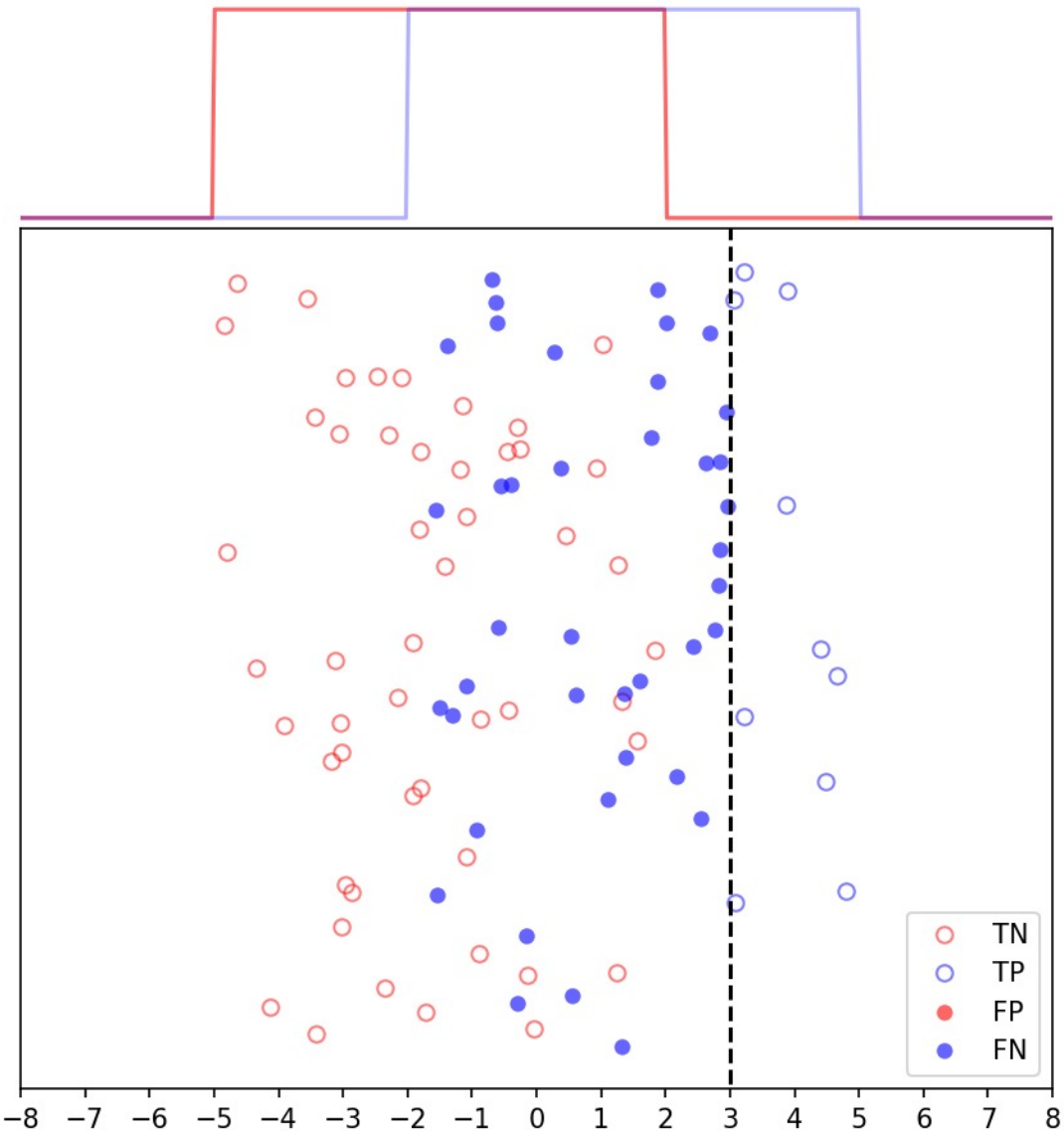
Example of ROC & PR

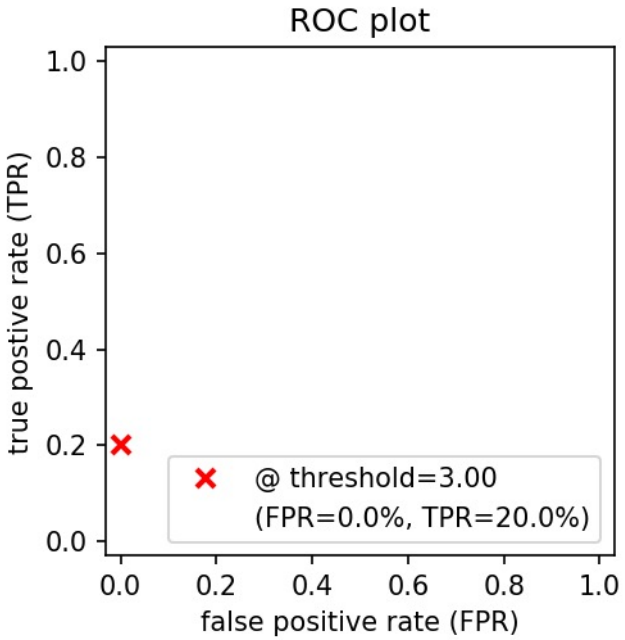


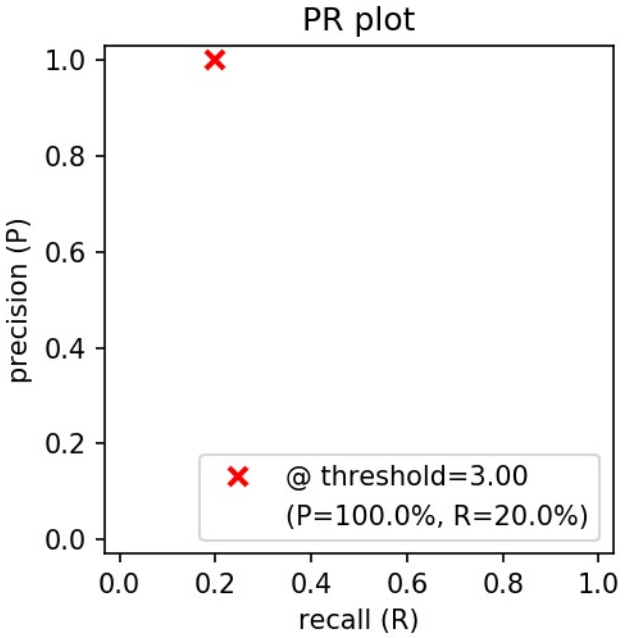




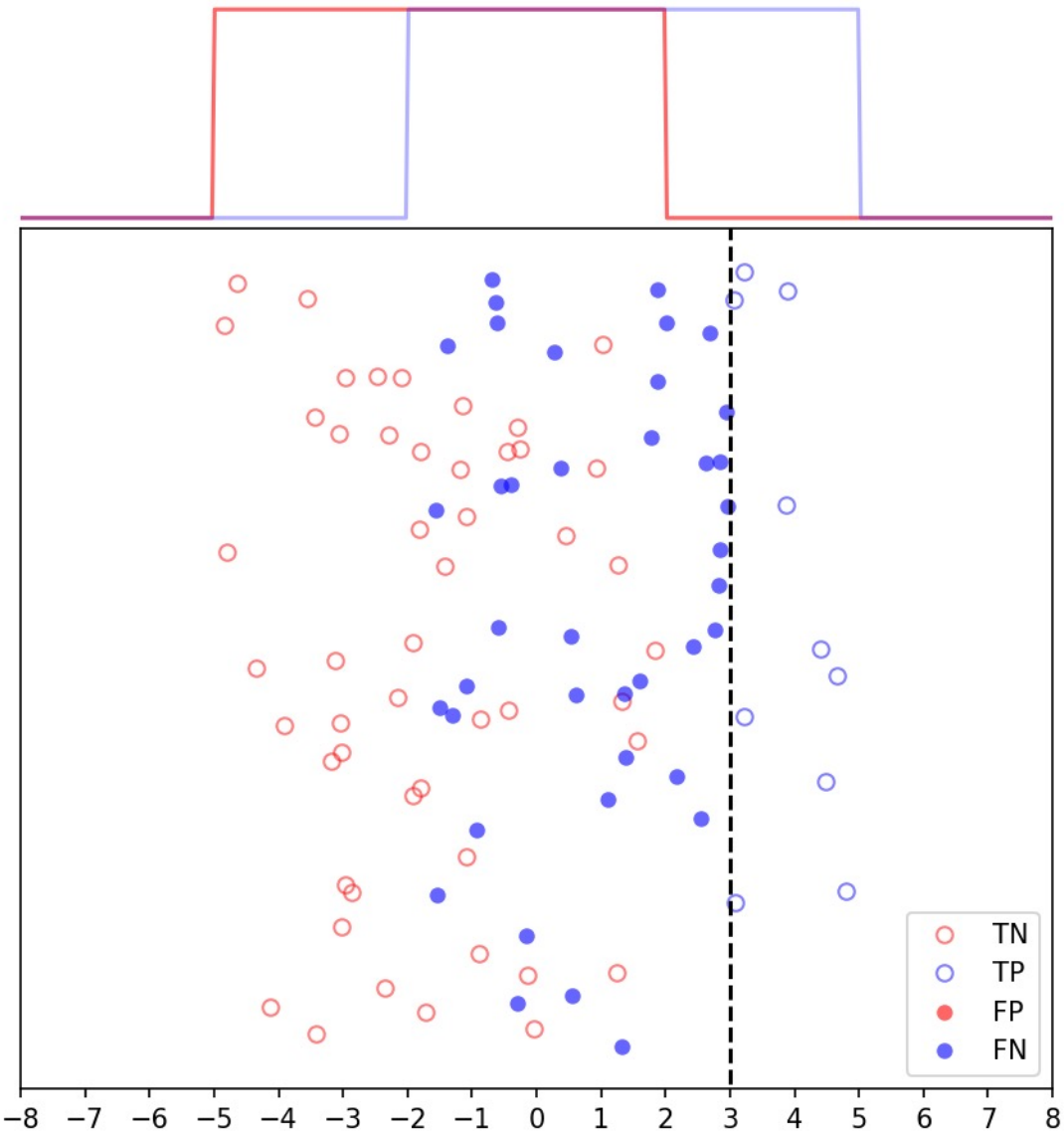
Example of ROC & PR

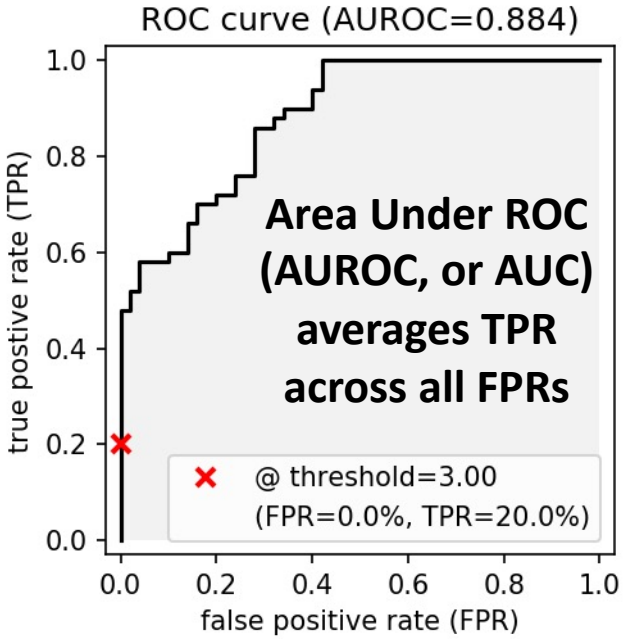


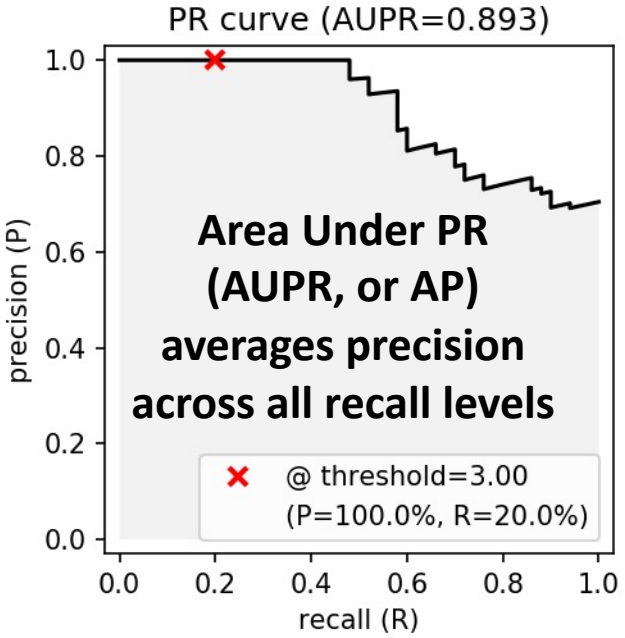




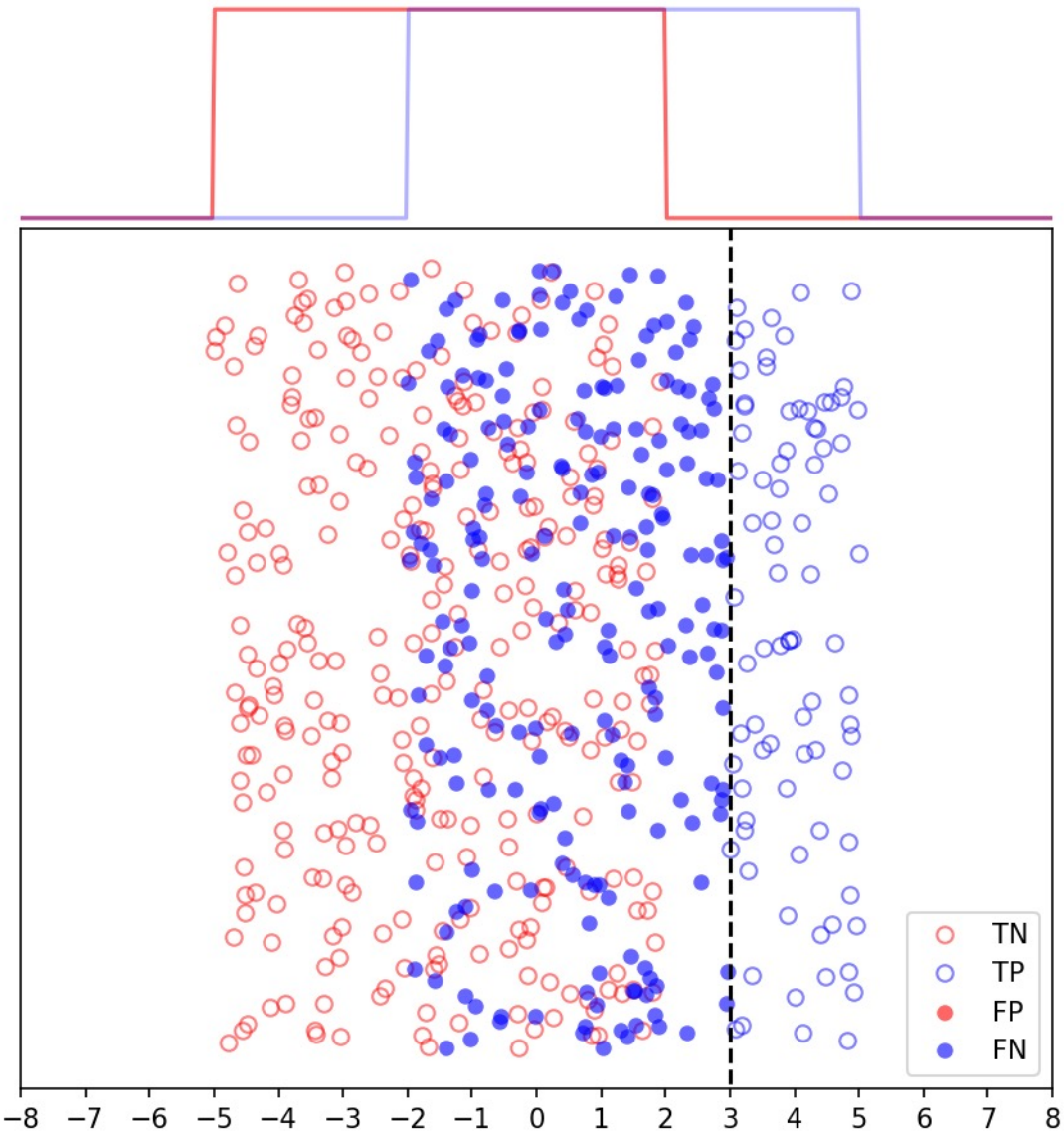
Example of ROC & PR

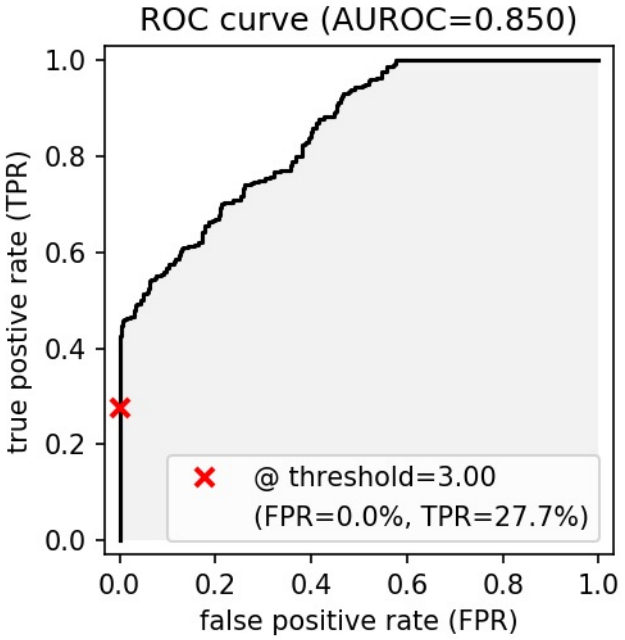


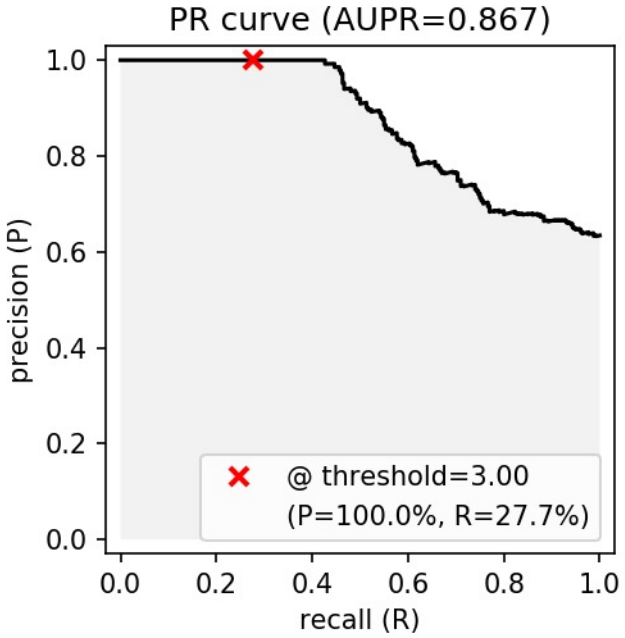




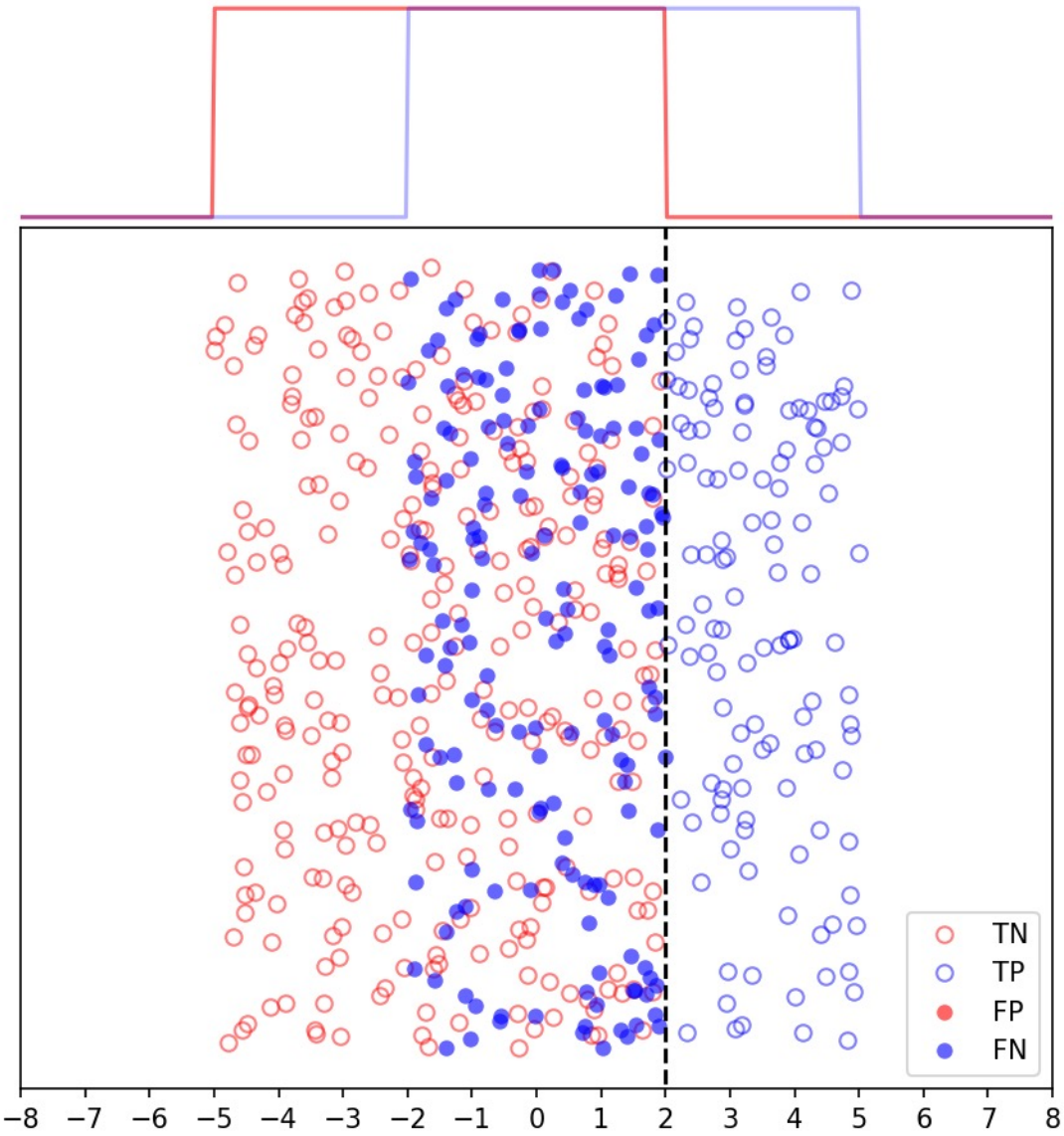
Example of ROC & PR

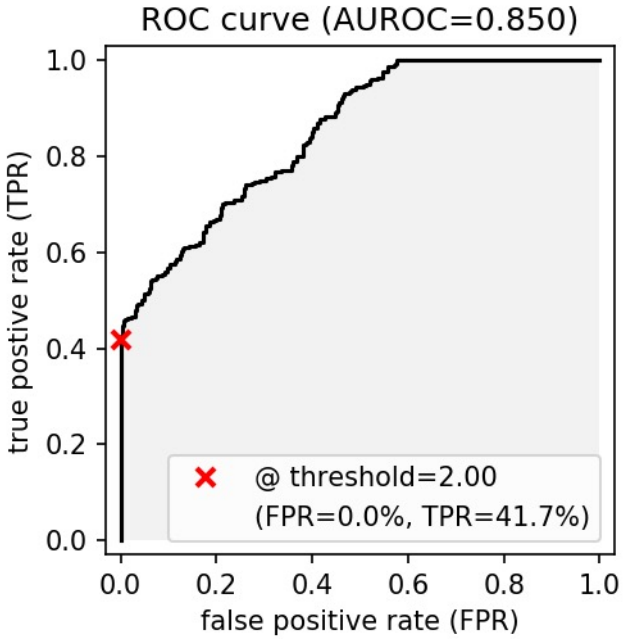


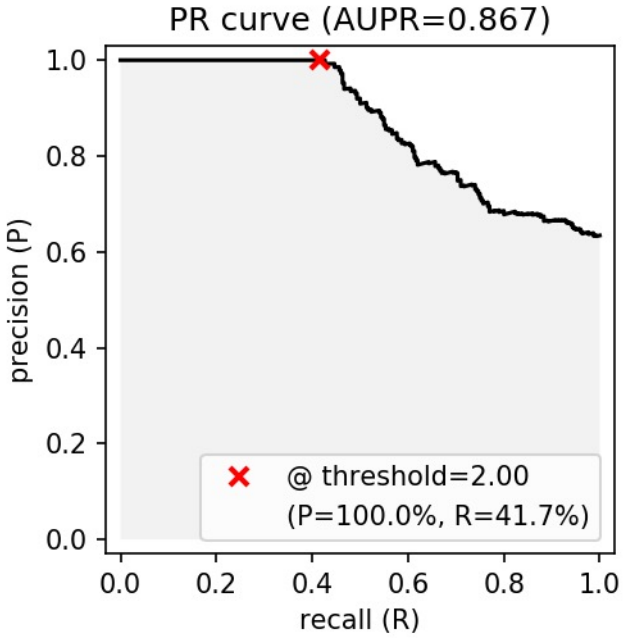




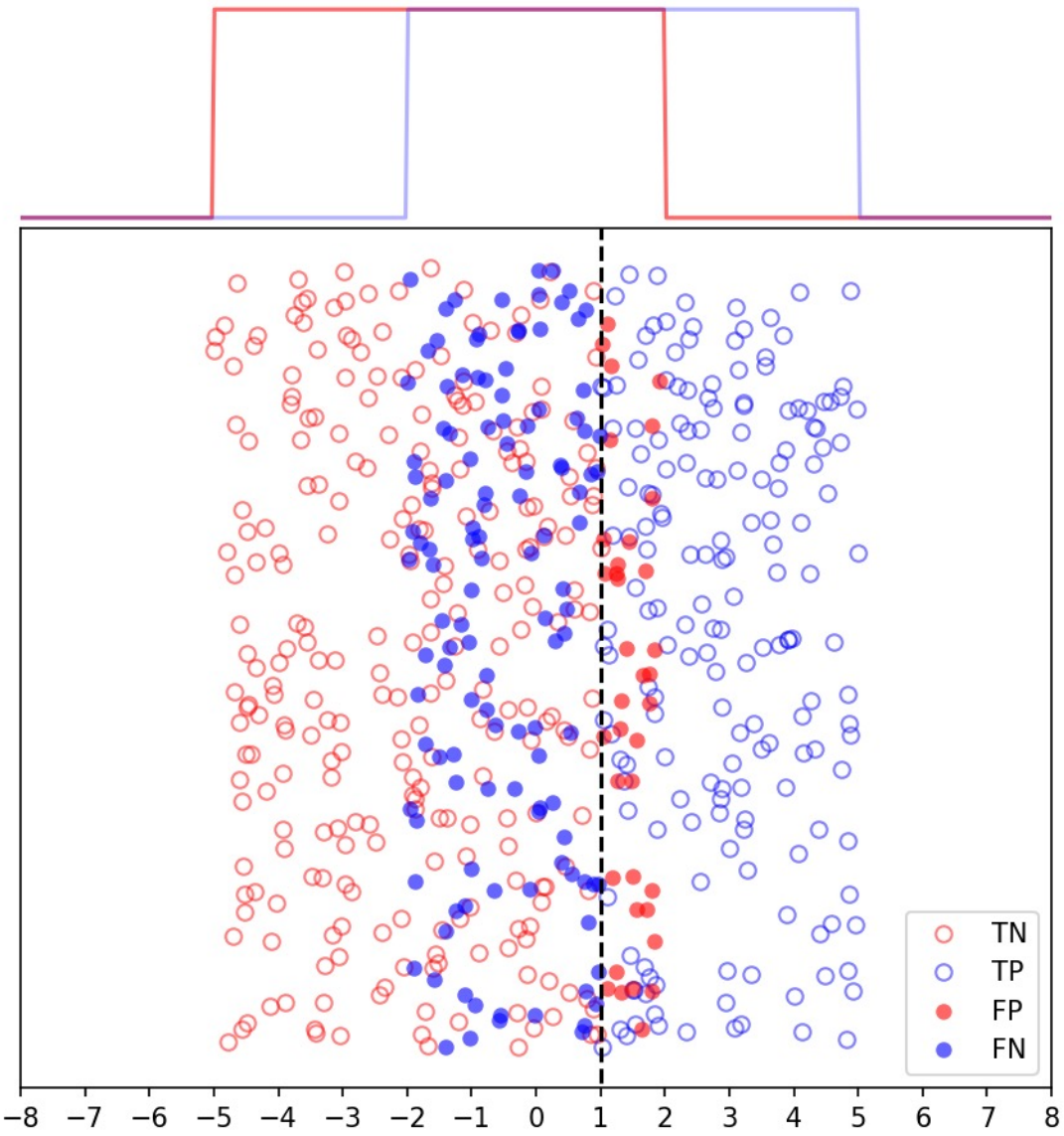
Example of ROC & PR

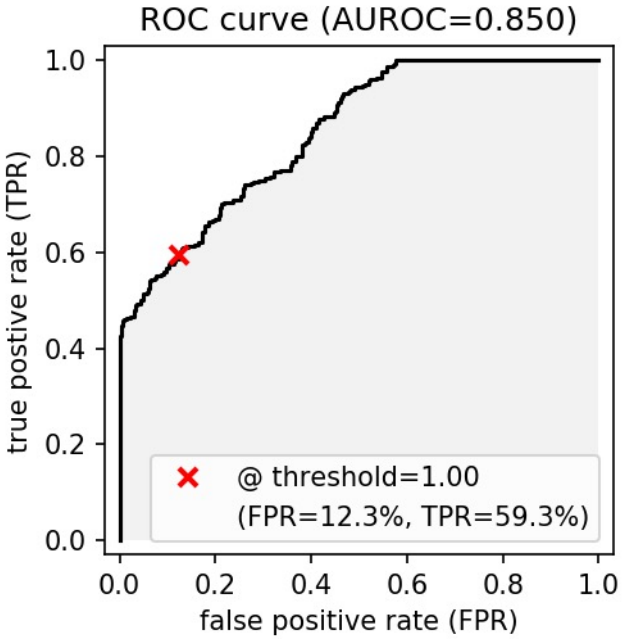


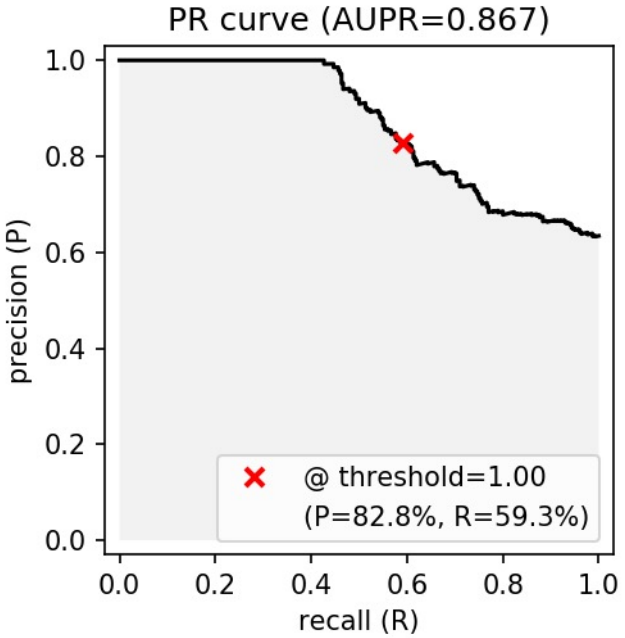




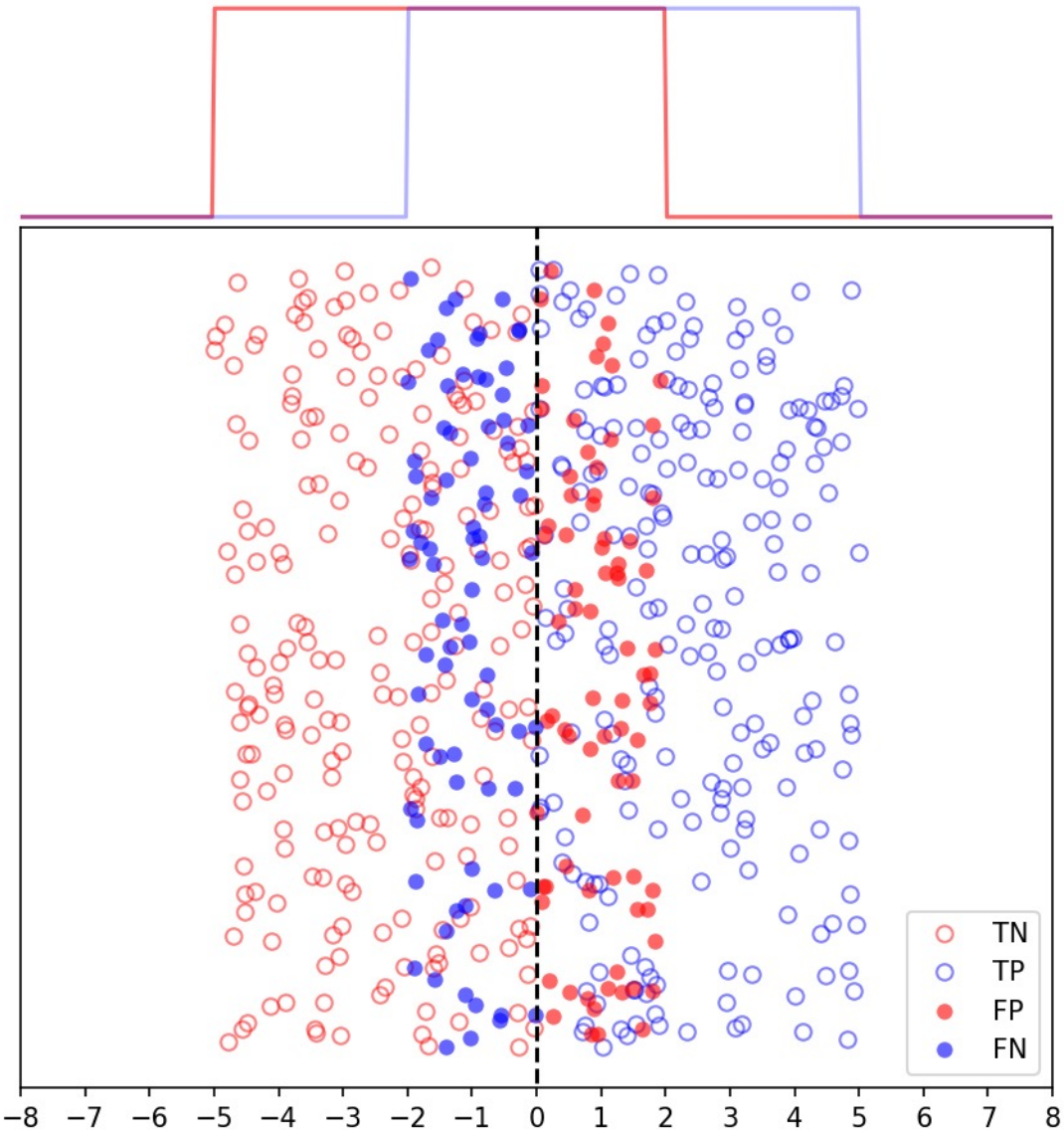
Example of ROC & PR

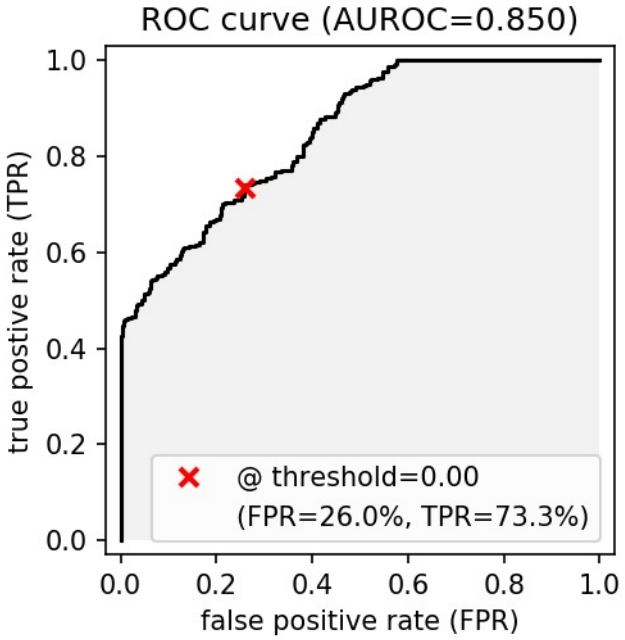


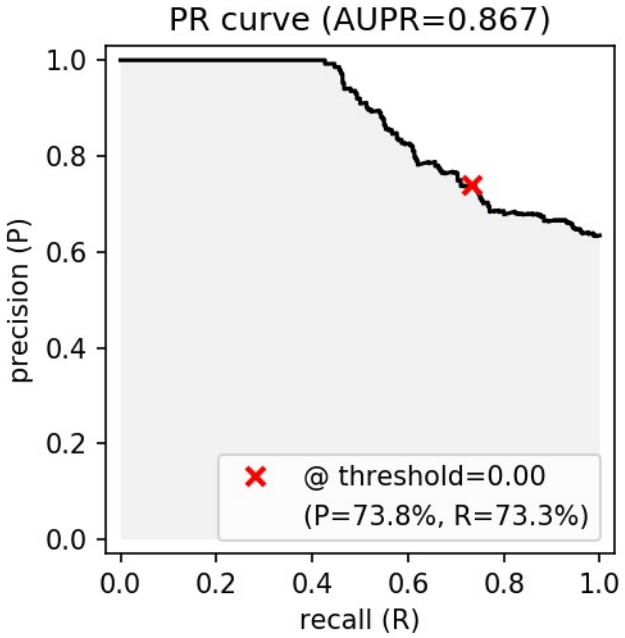




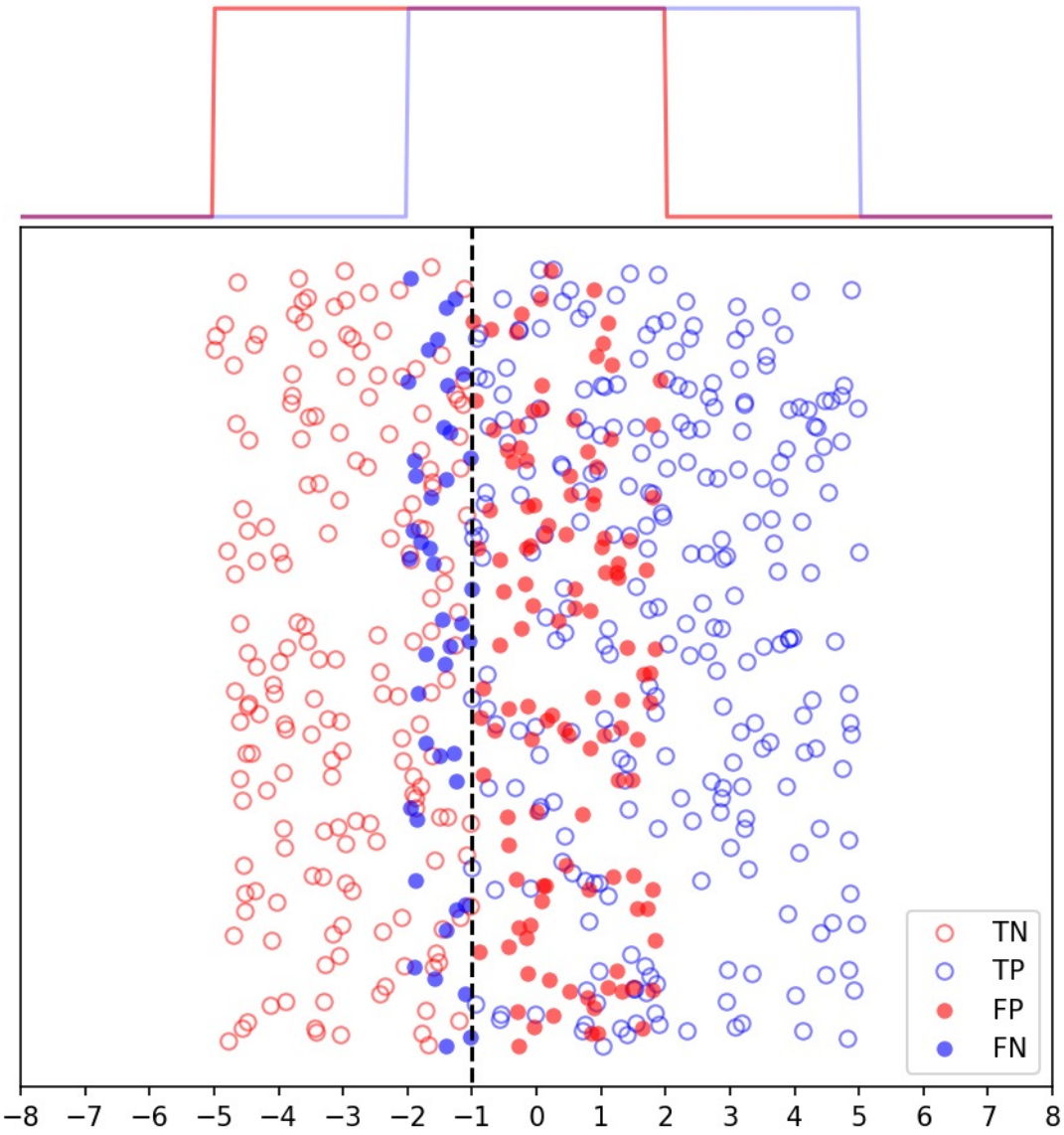
Example of ROC & PR

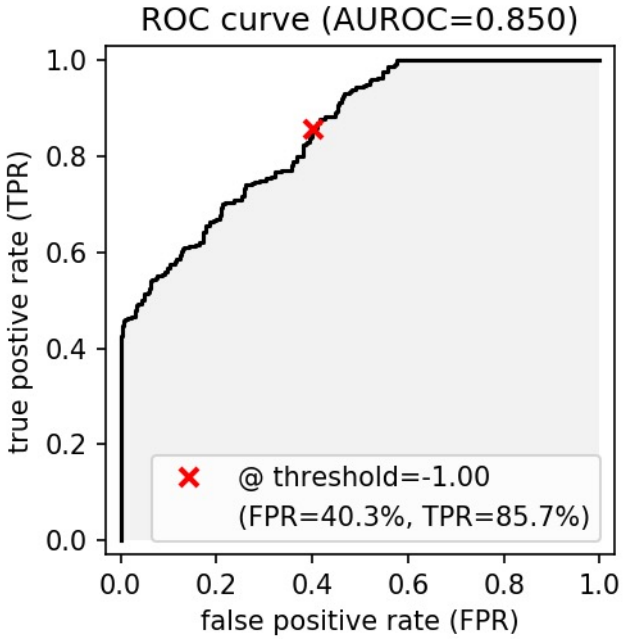


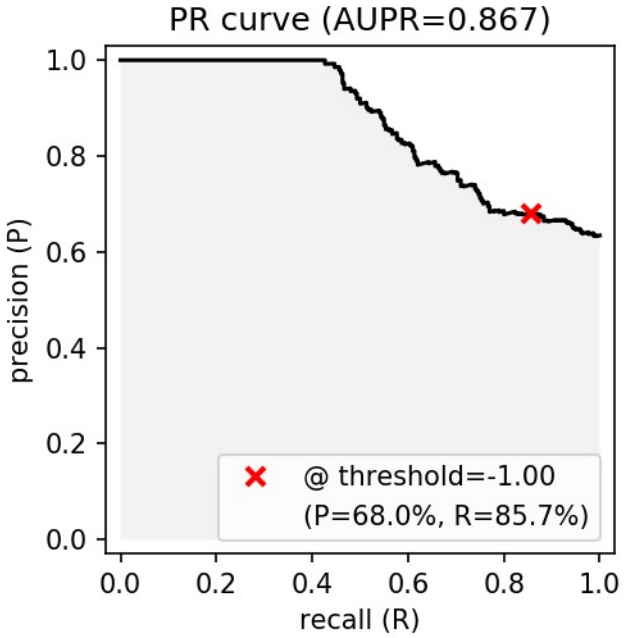




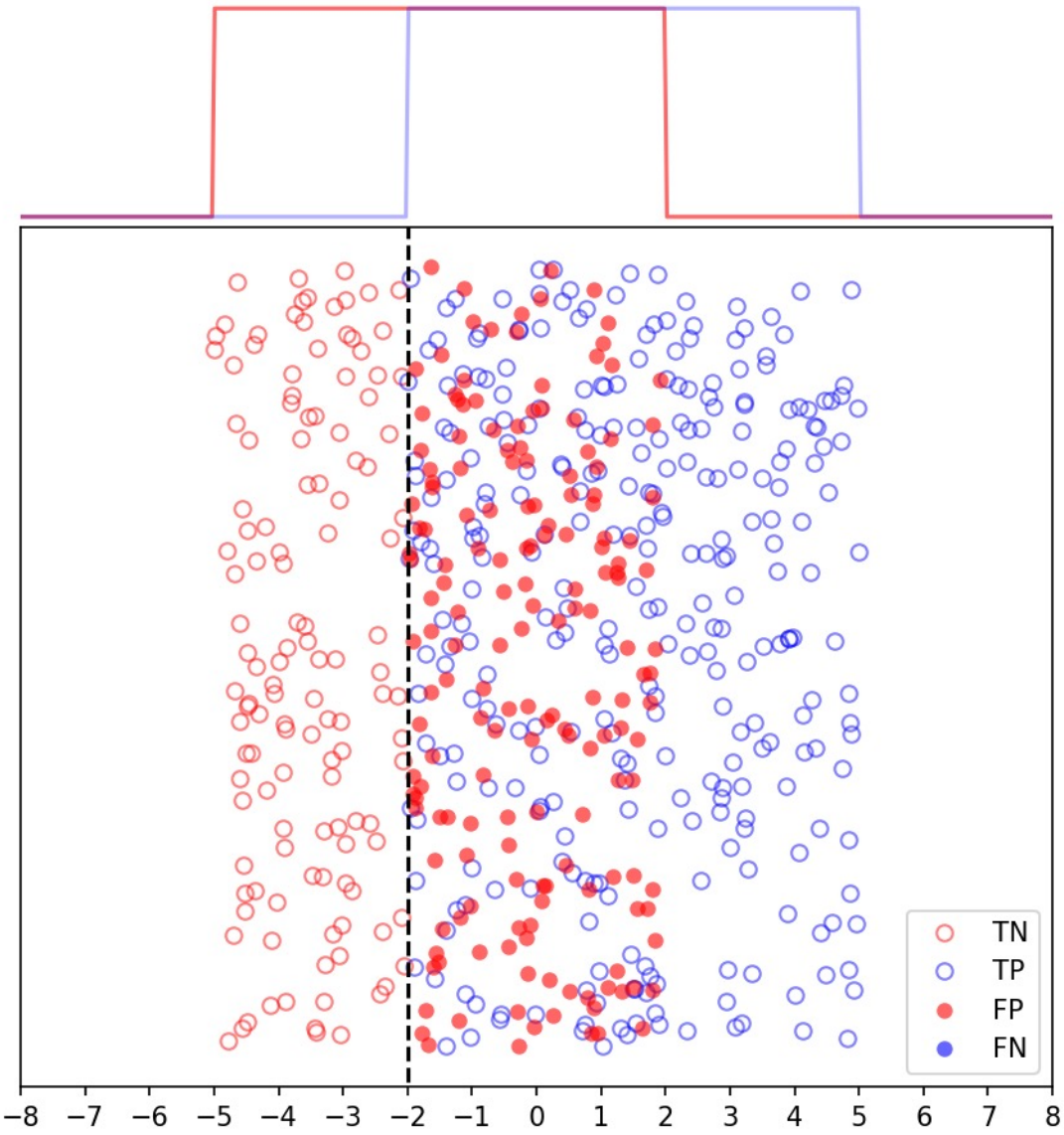
Example of ROC & PR

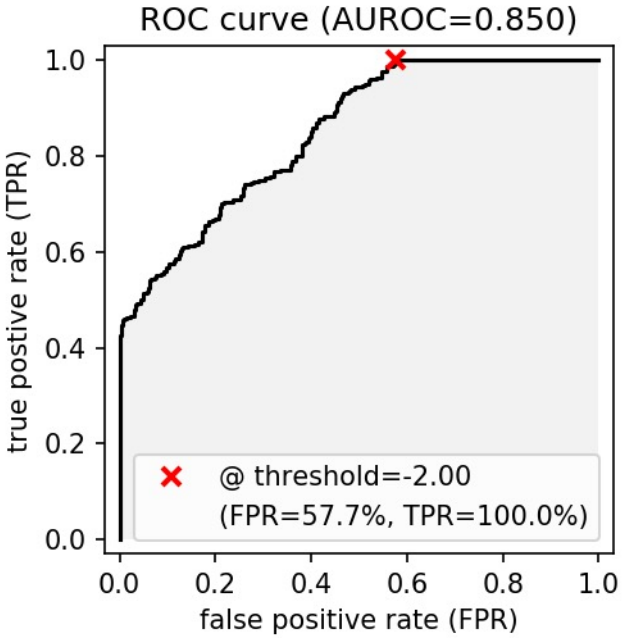


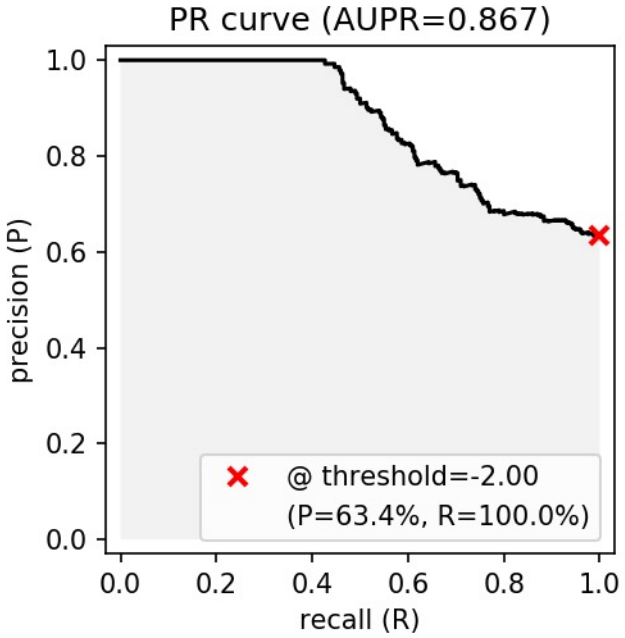




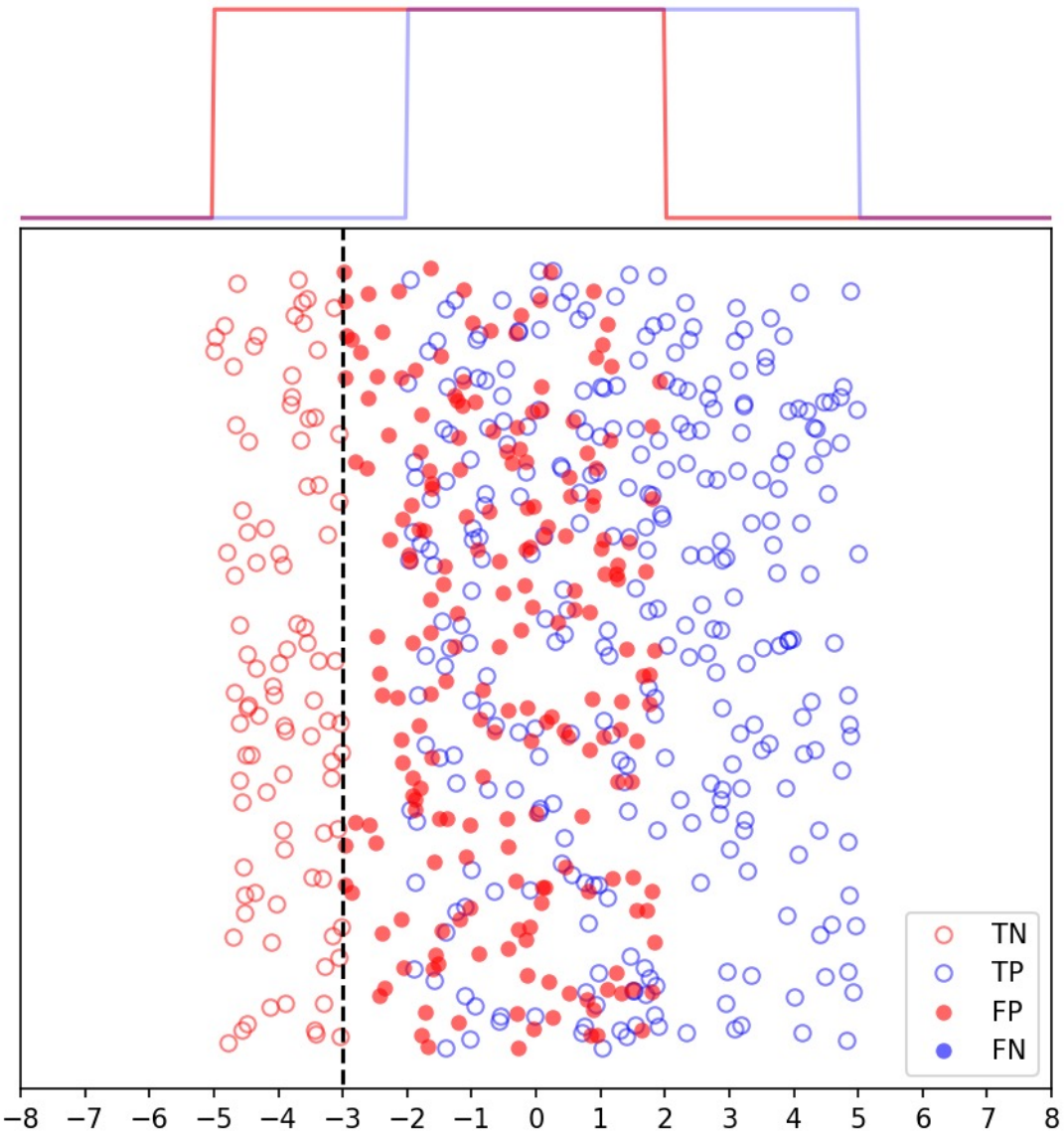
Example of ROC & PR

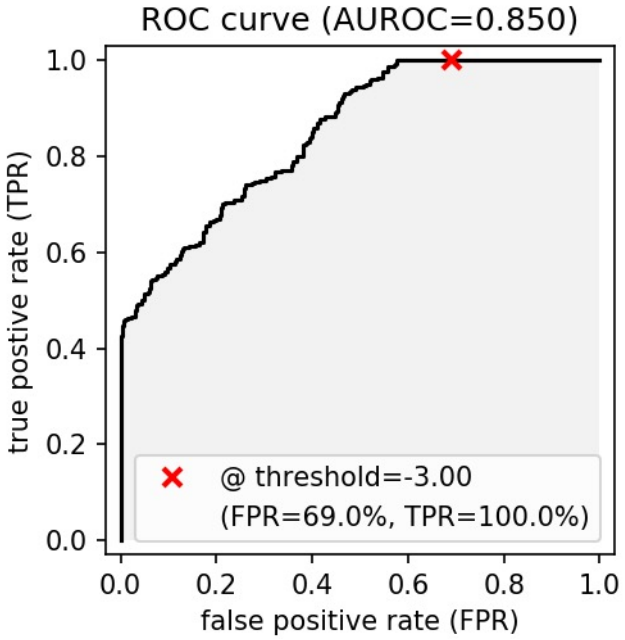


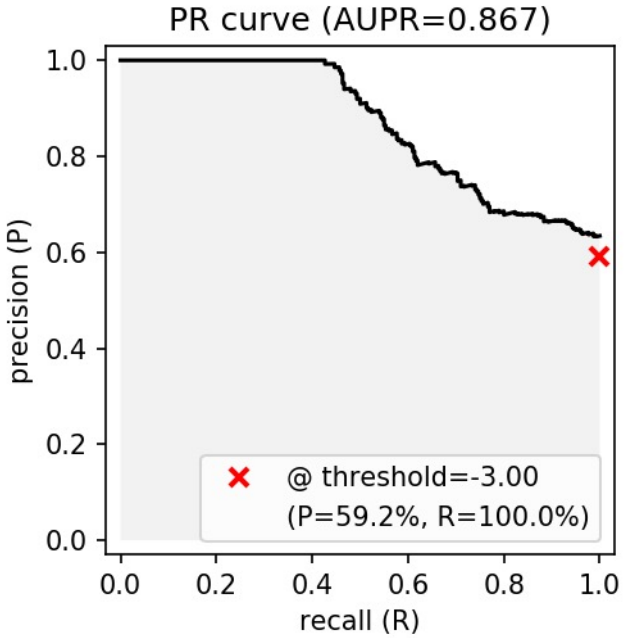




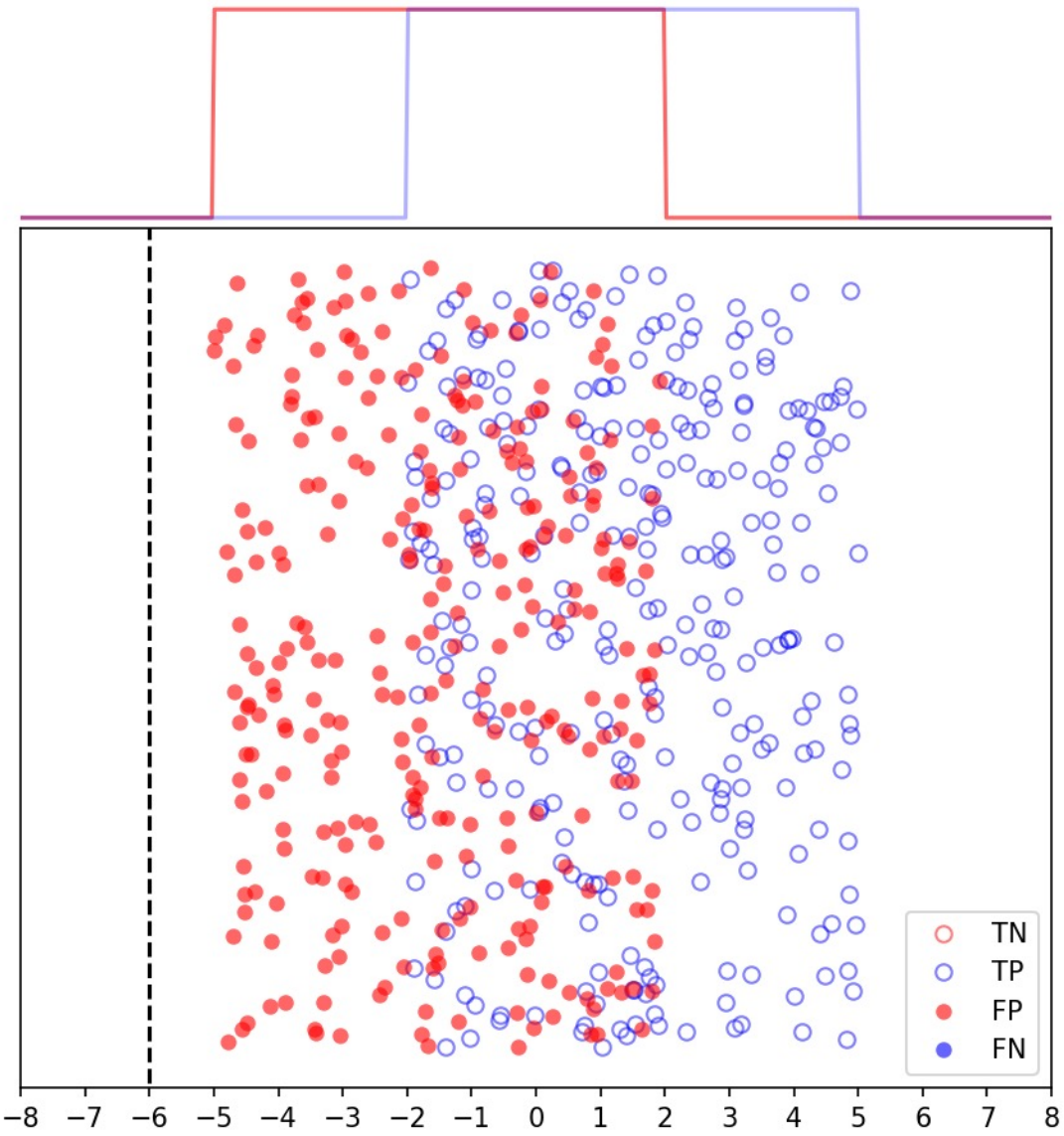
Example of ROC & PR

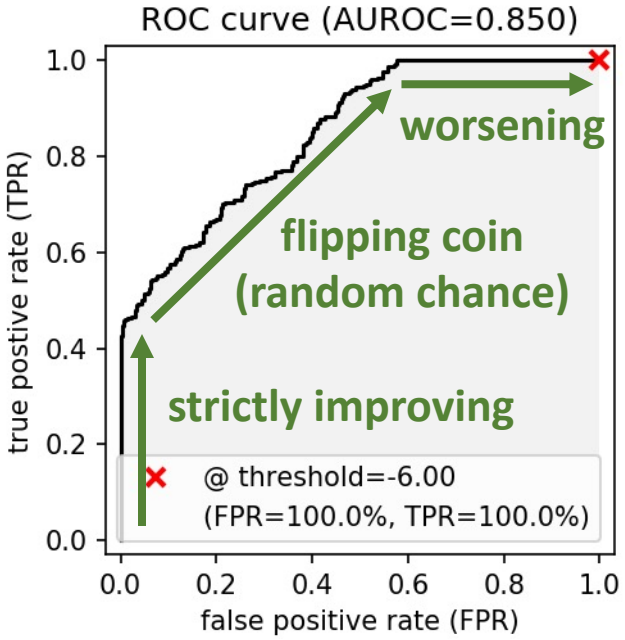


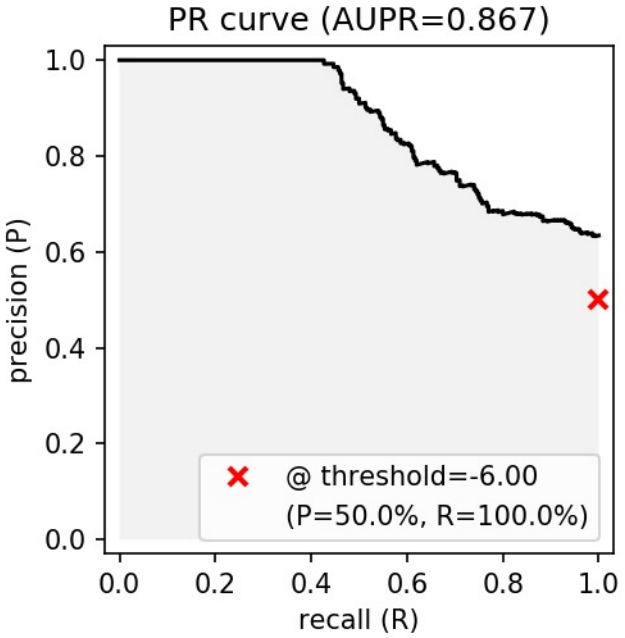




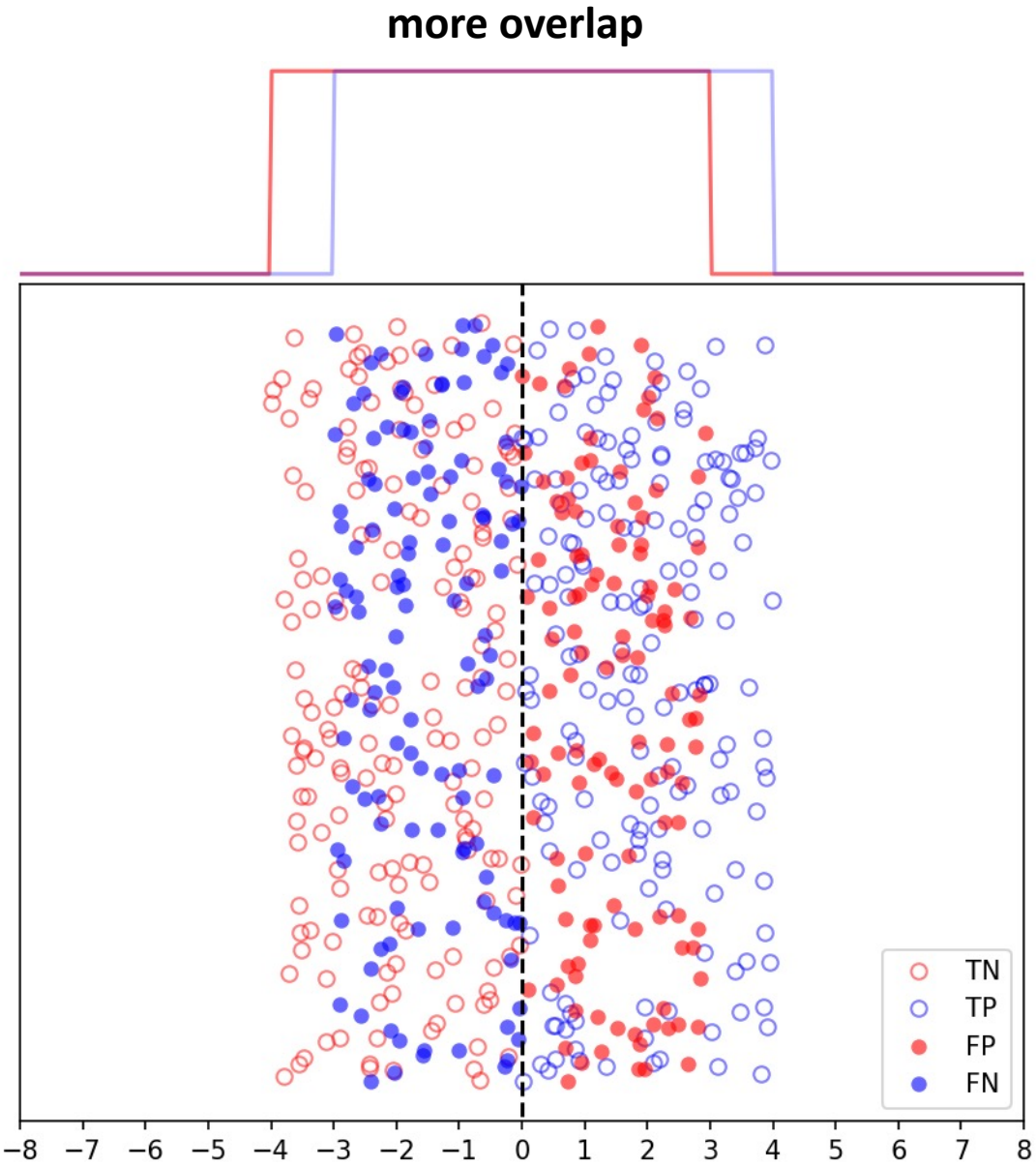
Example of ROC & PR

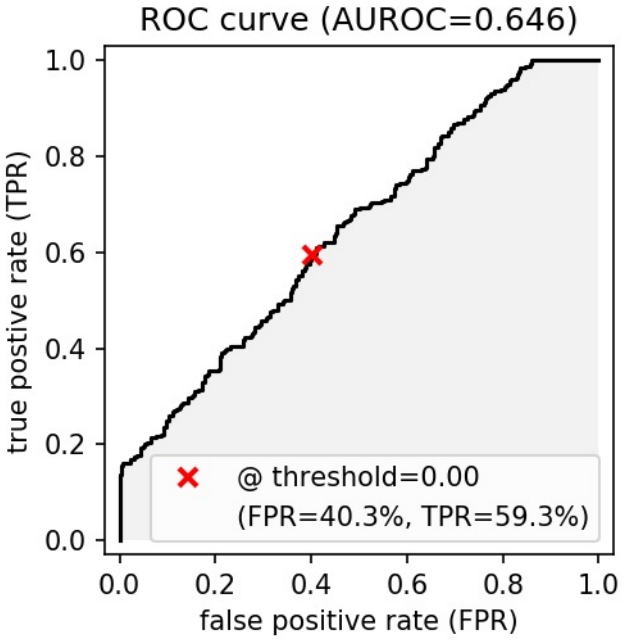


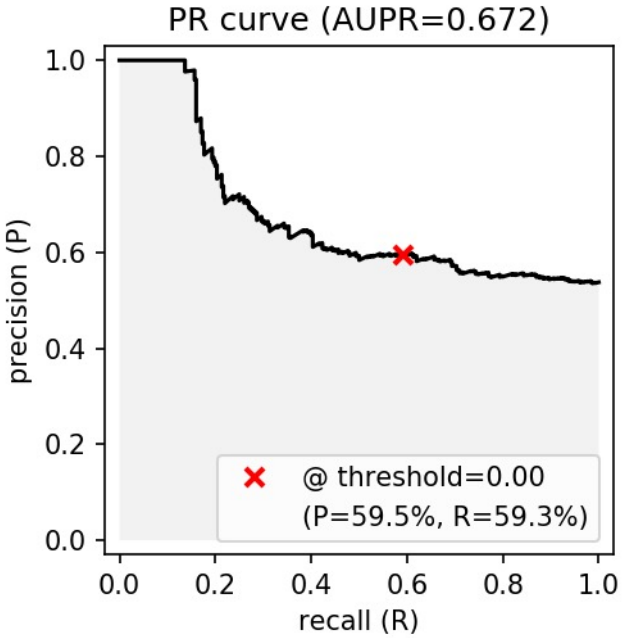




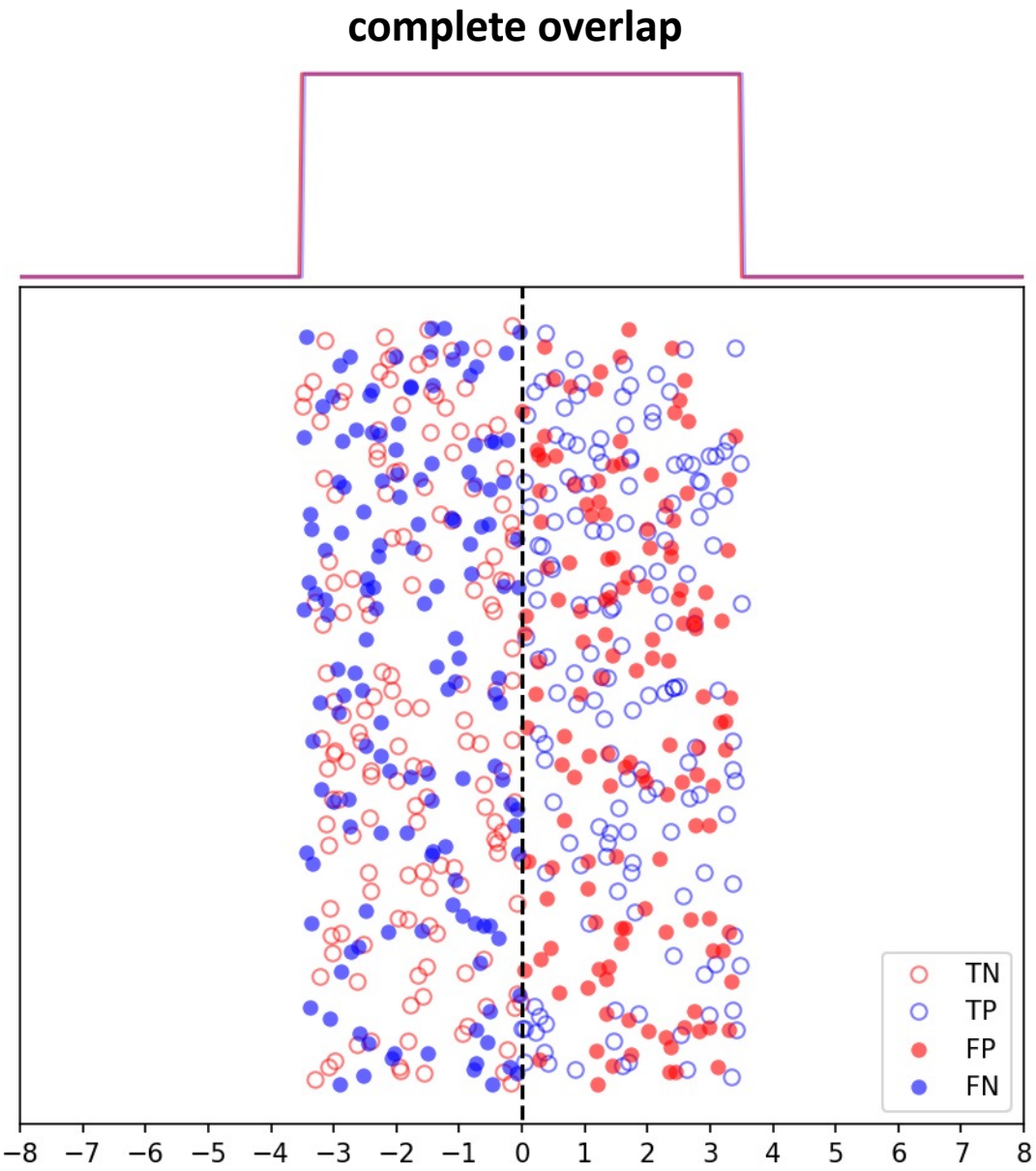
Example of ROC & PR

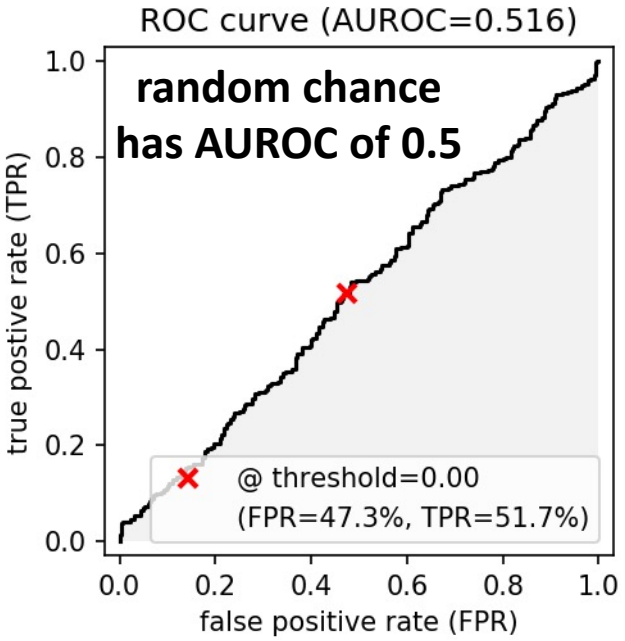


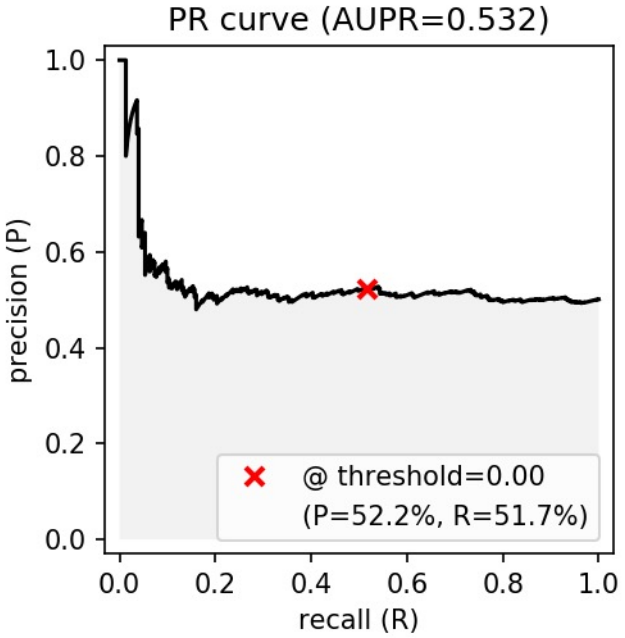




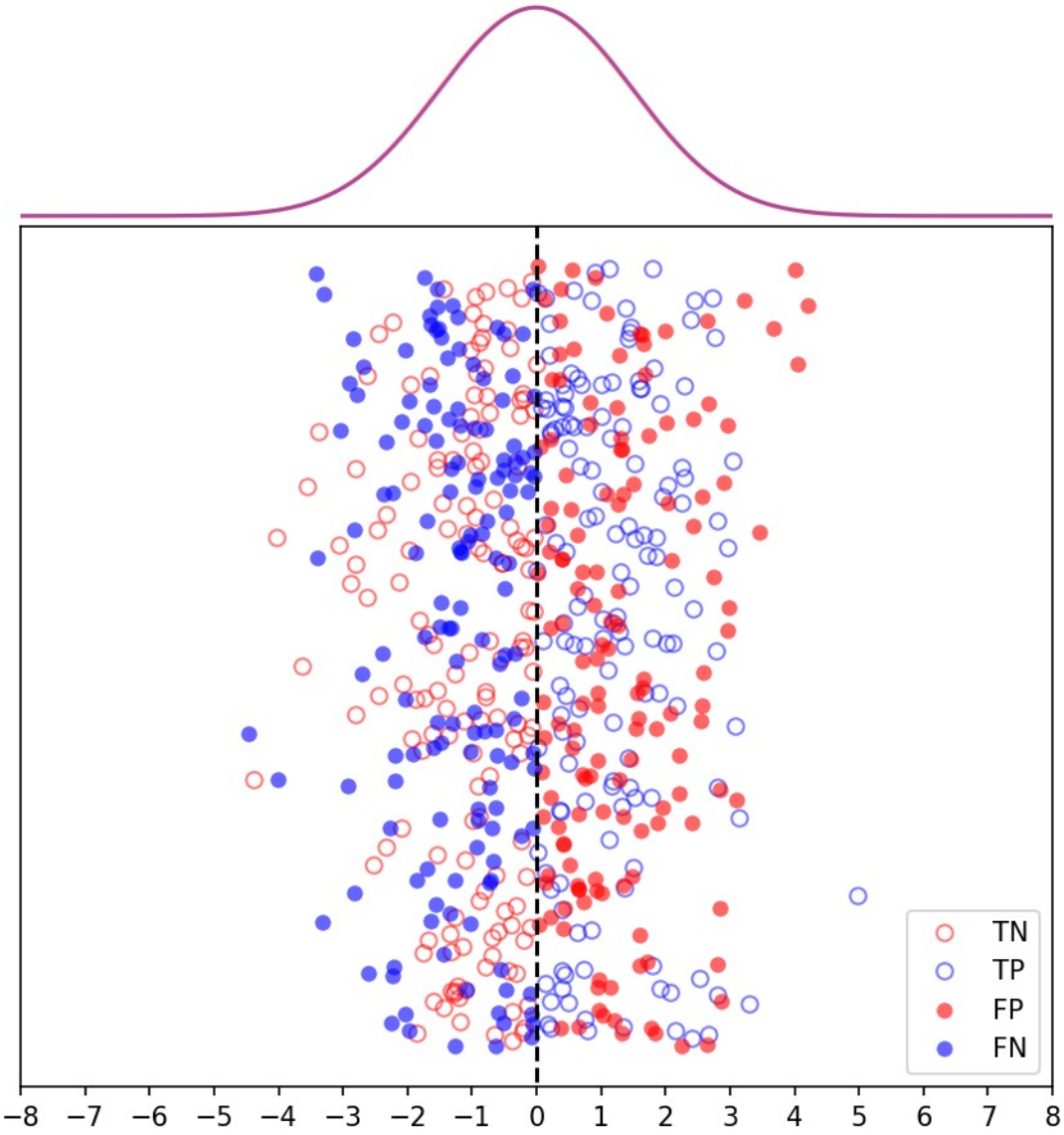
Example of ROC & PR

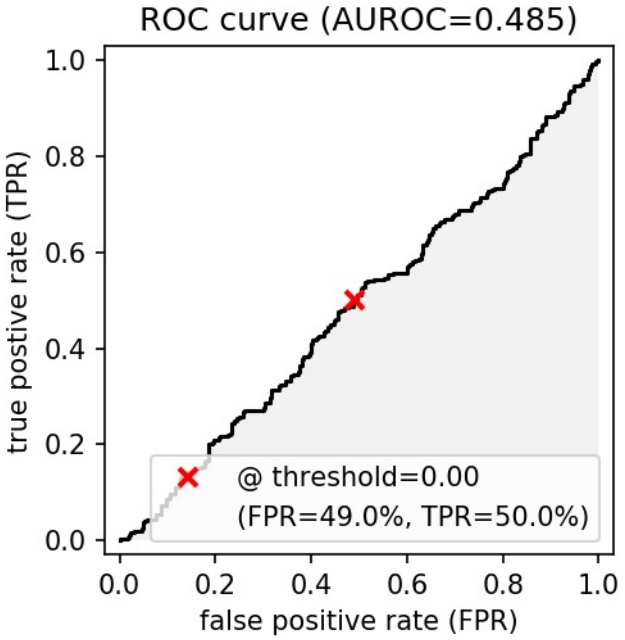


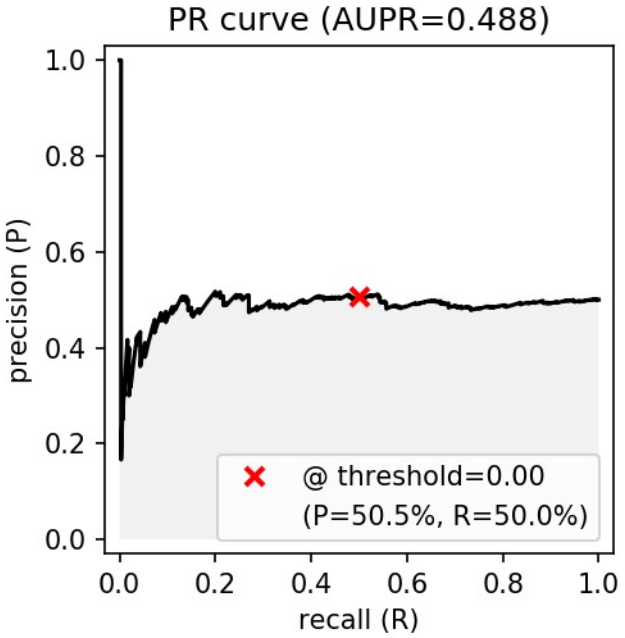




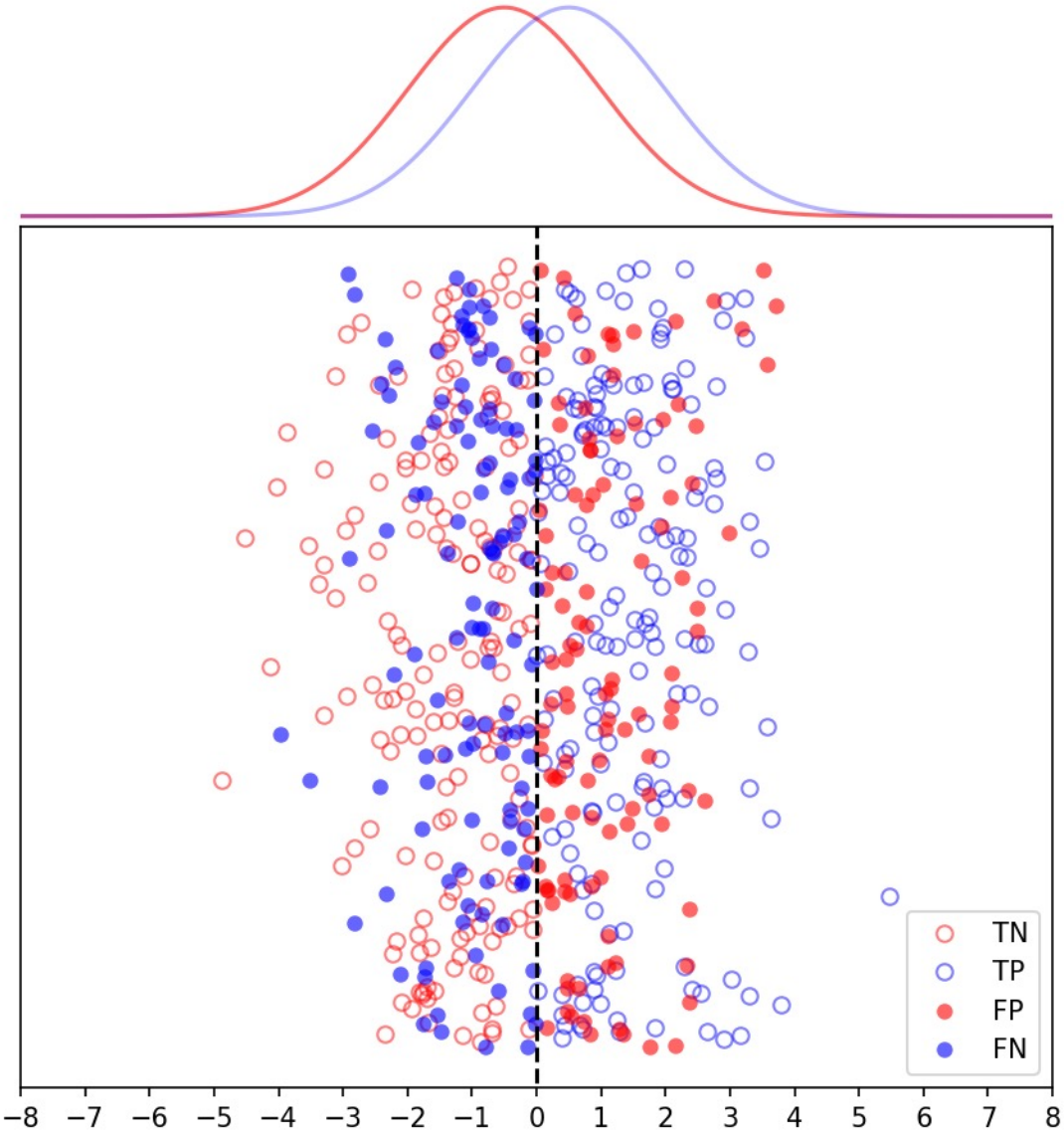
Example of ROC & PR

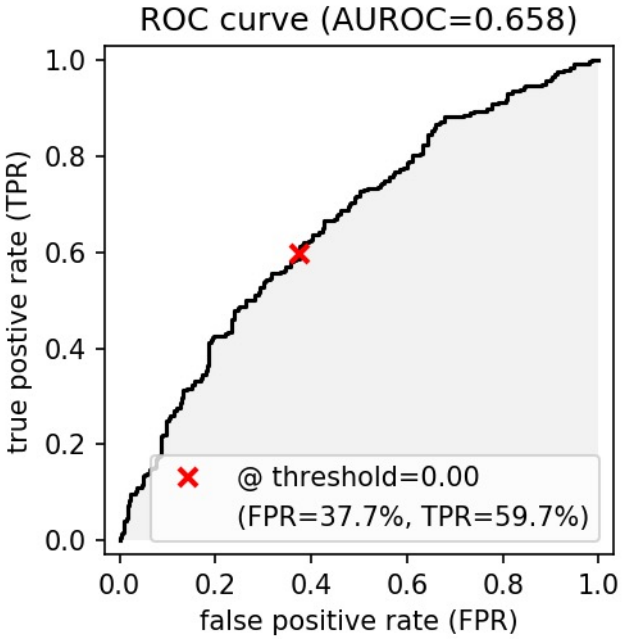


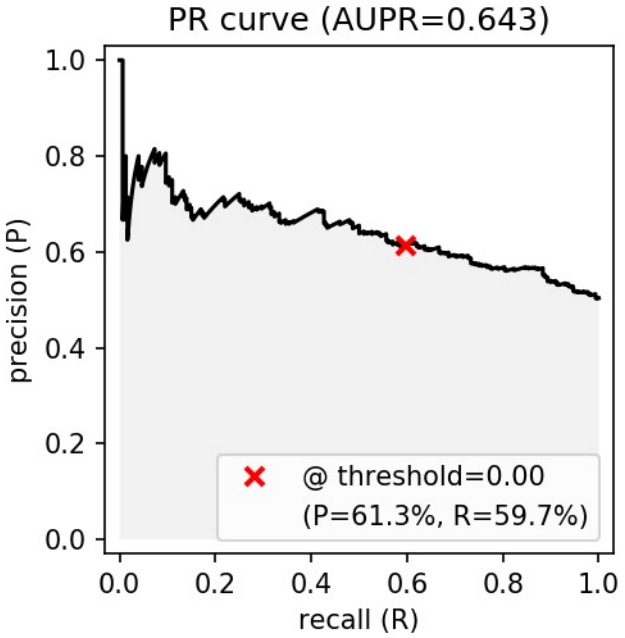




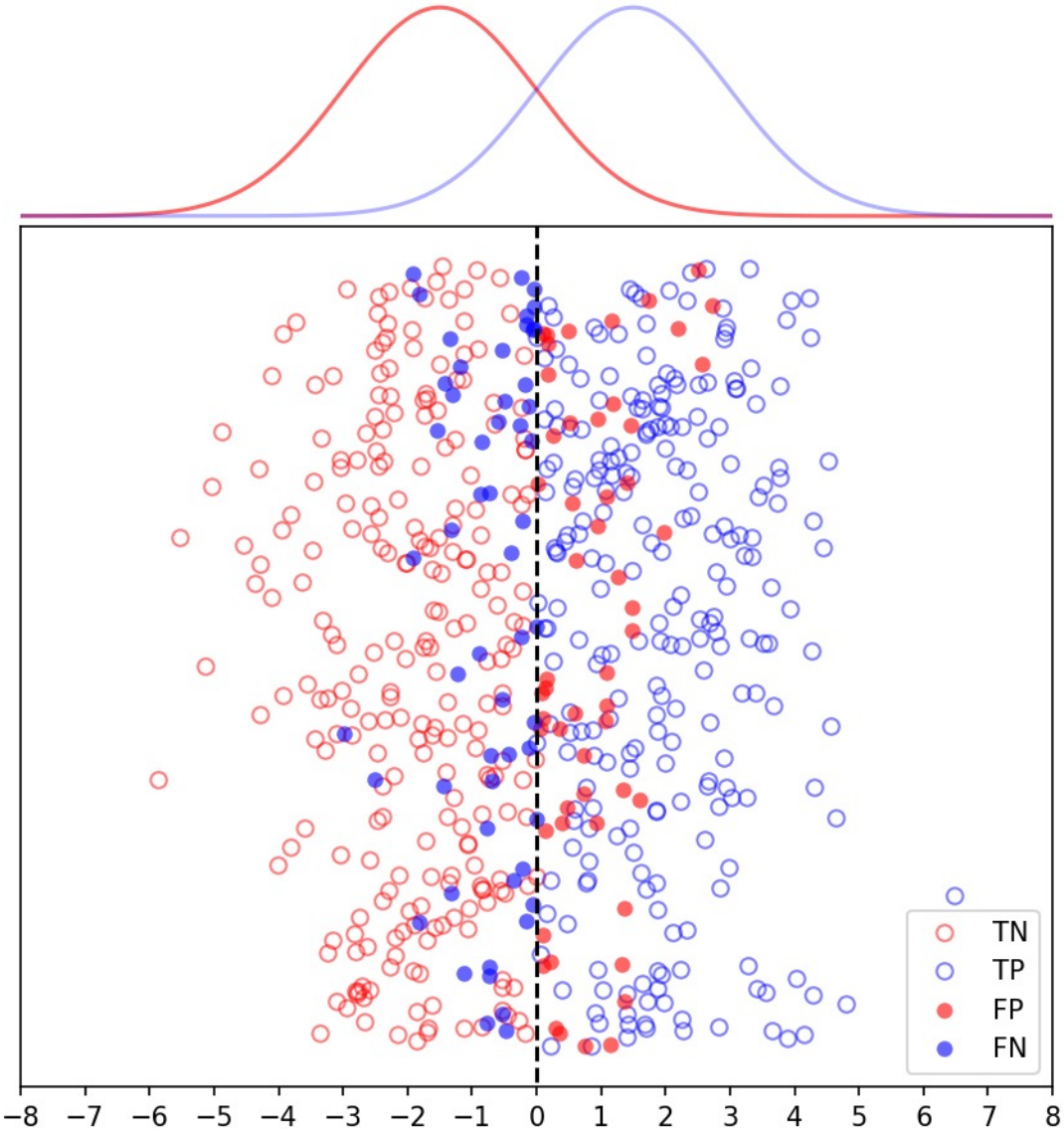
Example of ROC & PR

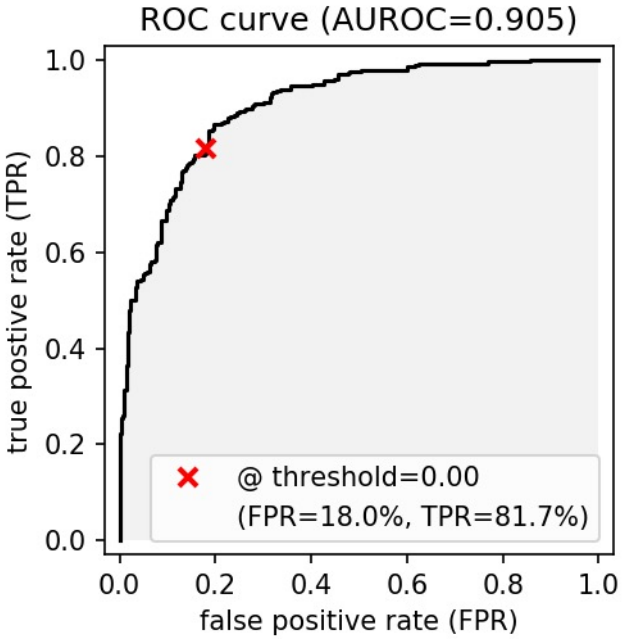


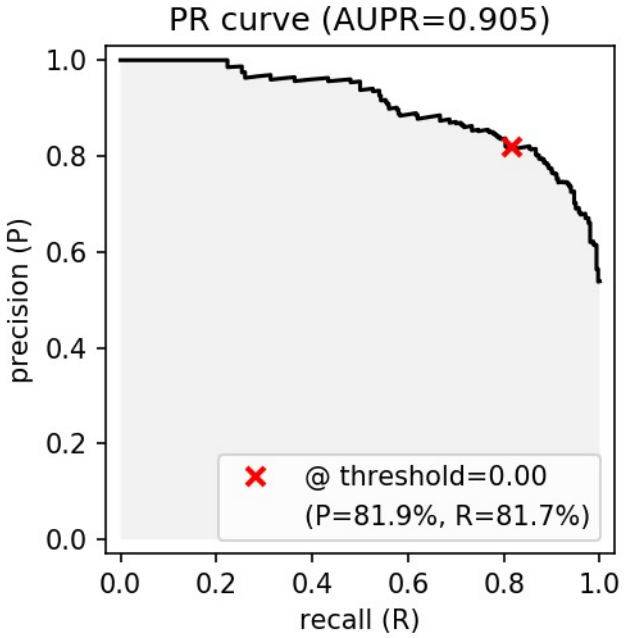




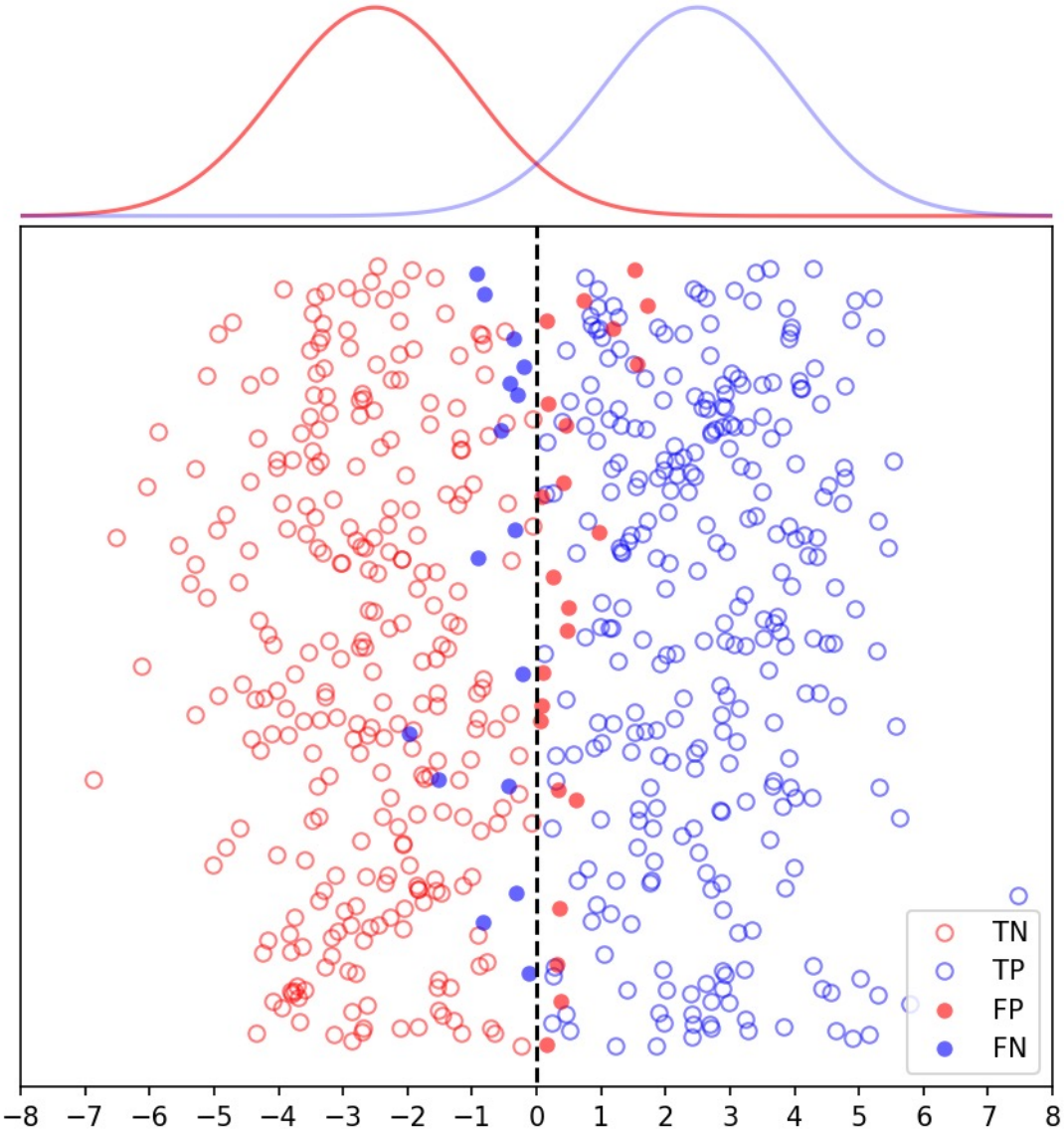
Example of ROC & PR

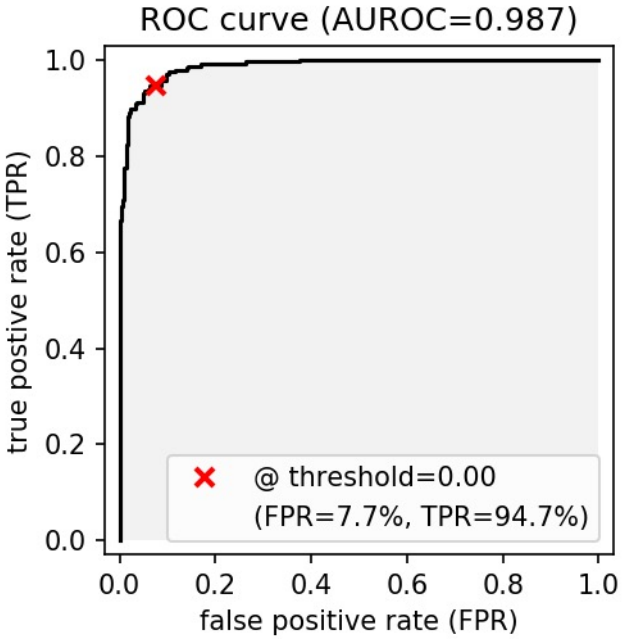


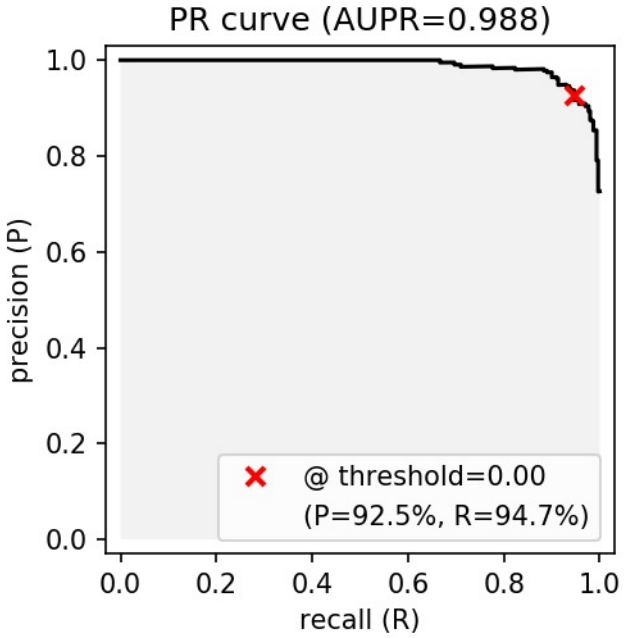




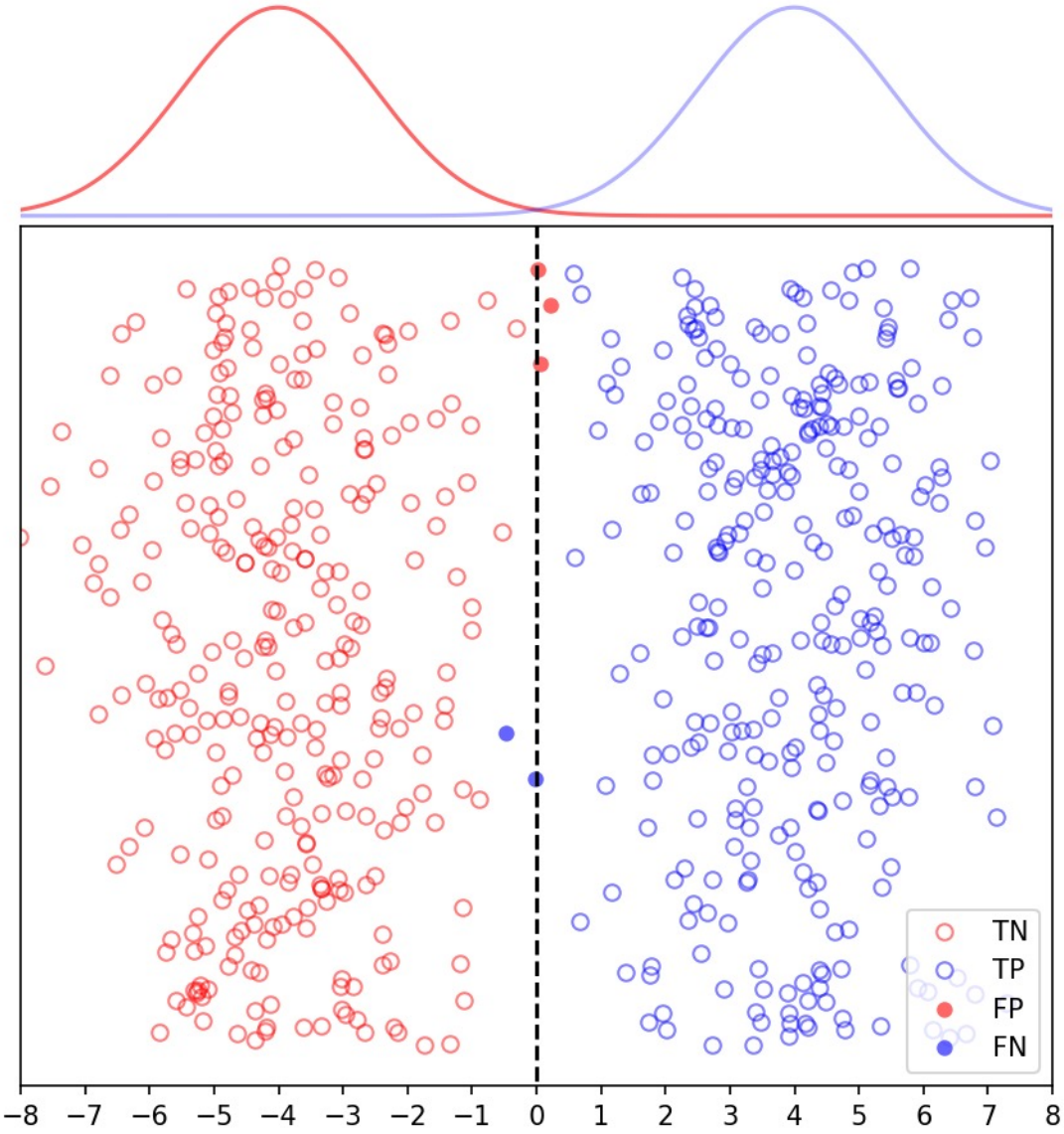
Example of ROC & PR

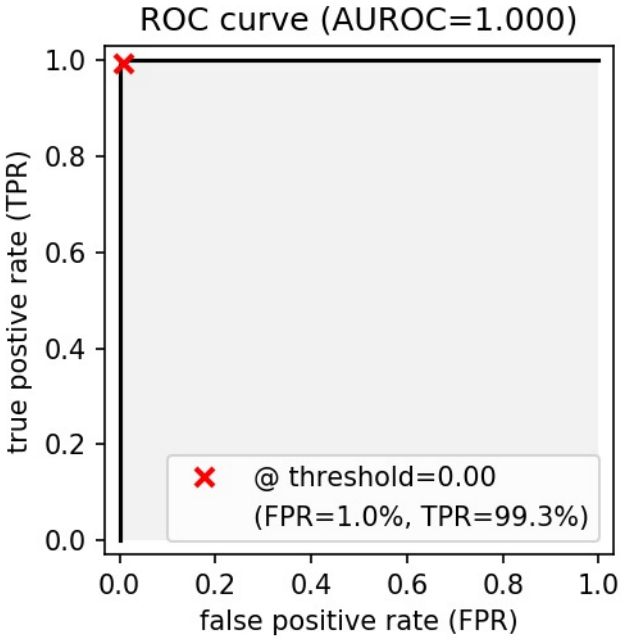


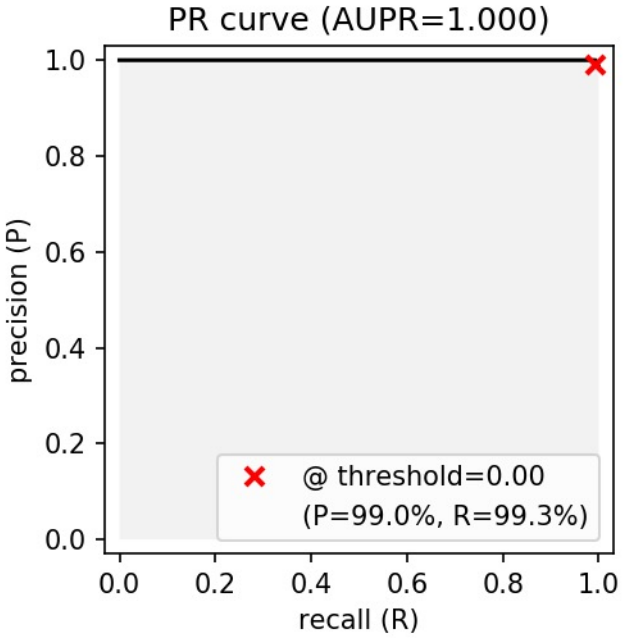




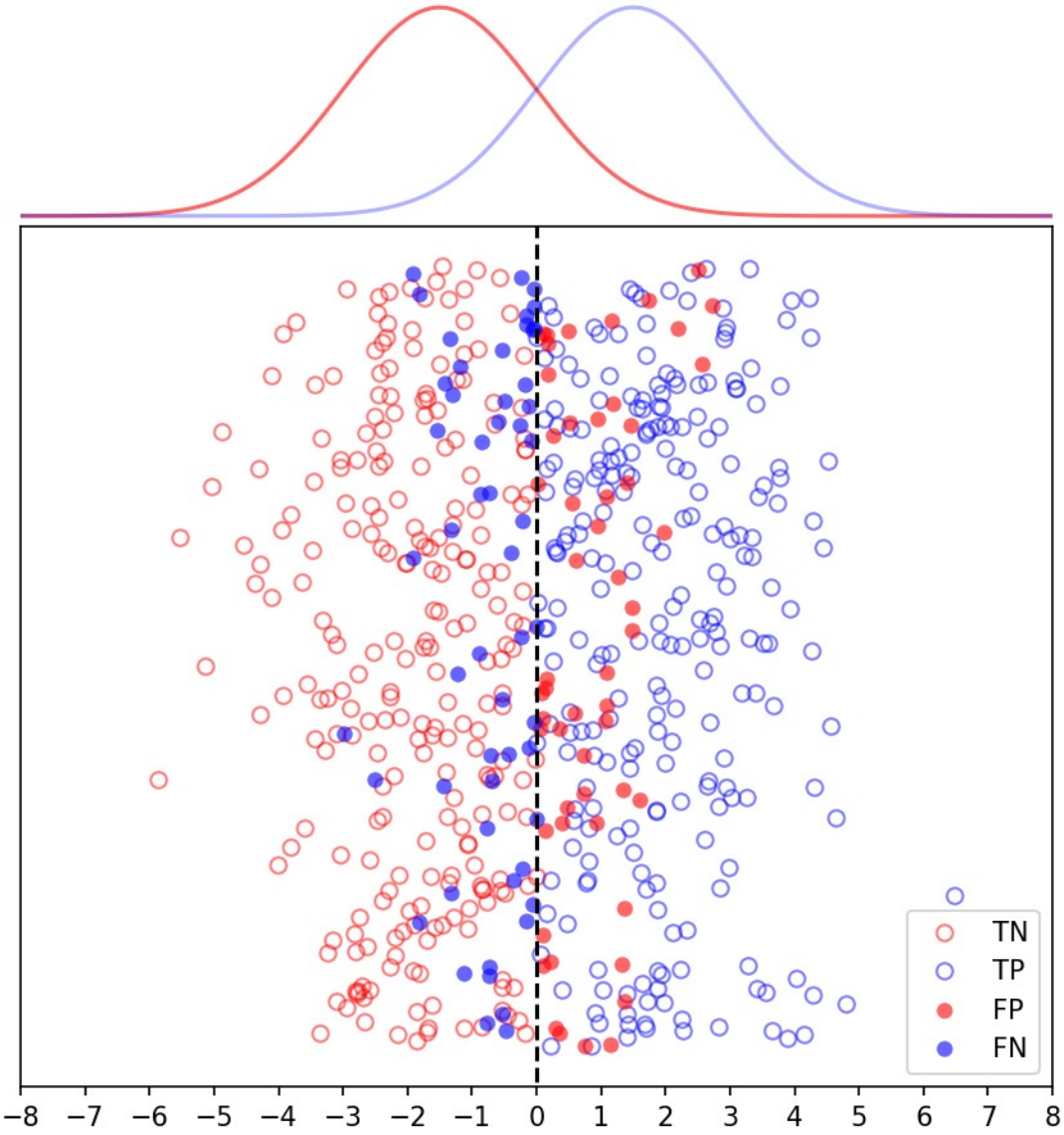
Example of ROC & PR

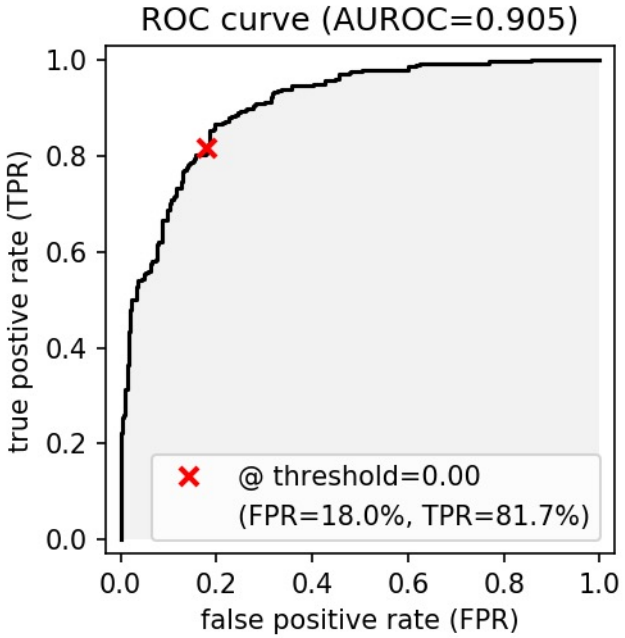


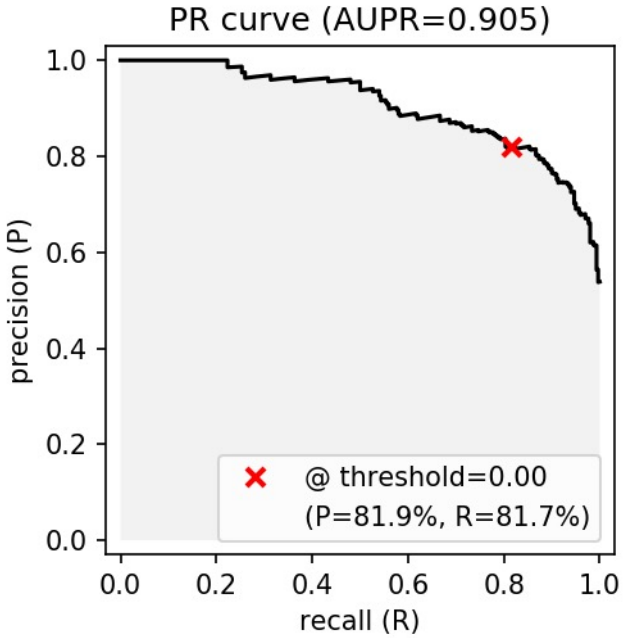




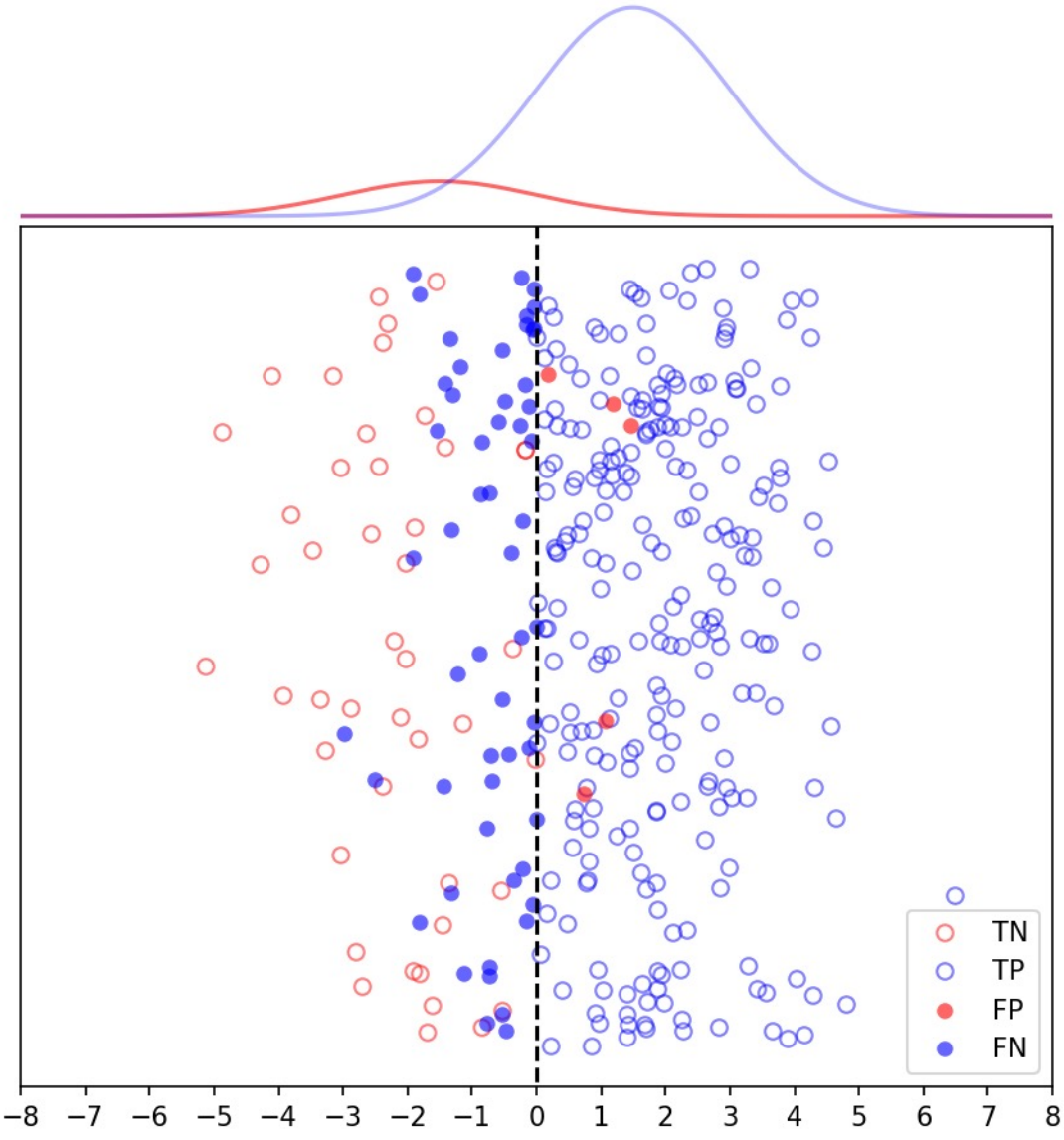
Example of ROC & PR

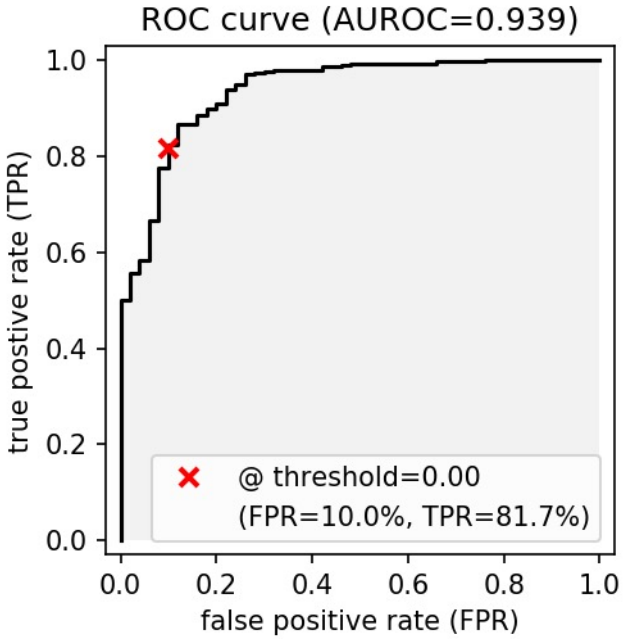


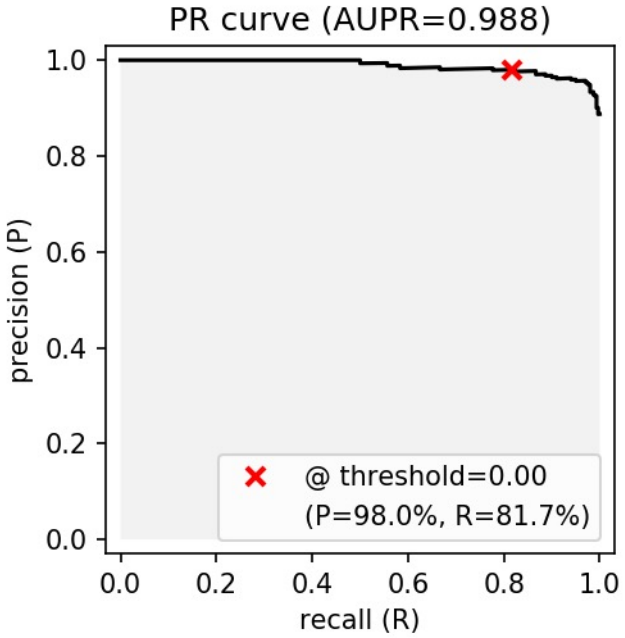




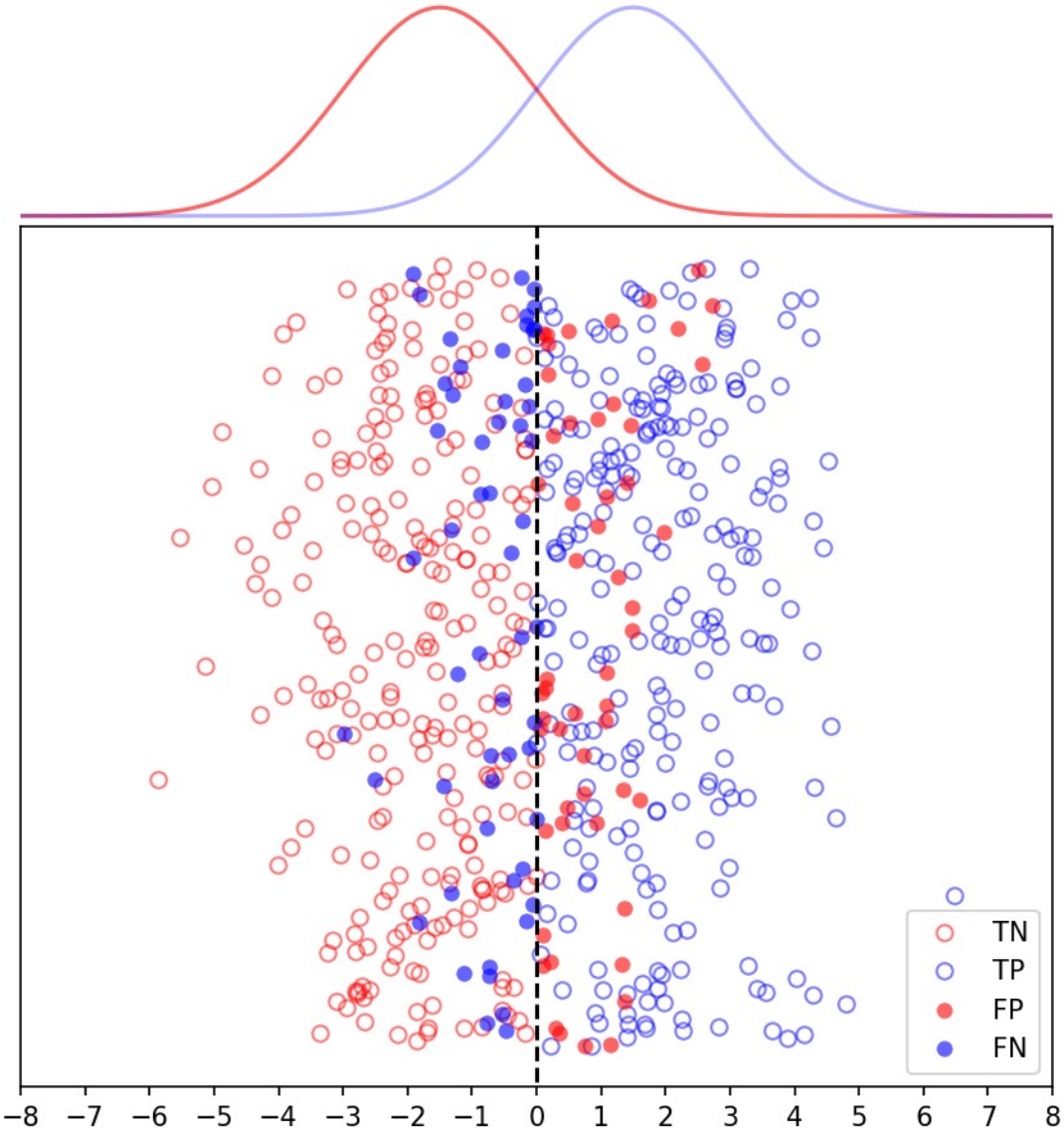
Example of ROC & PR

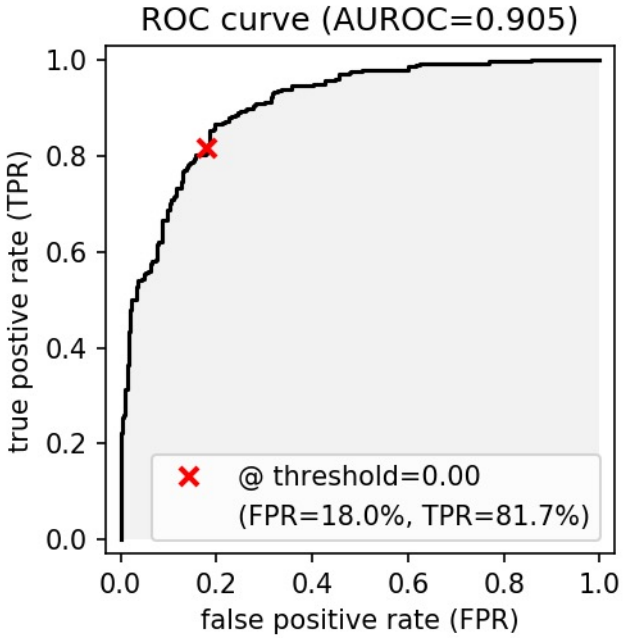


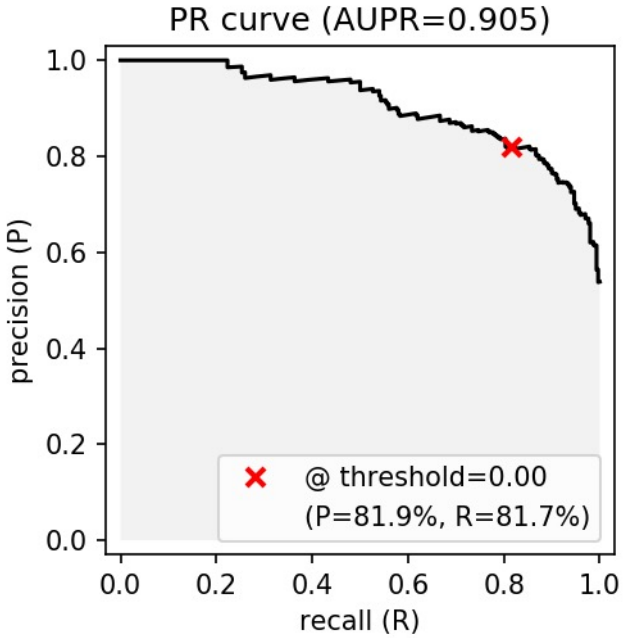




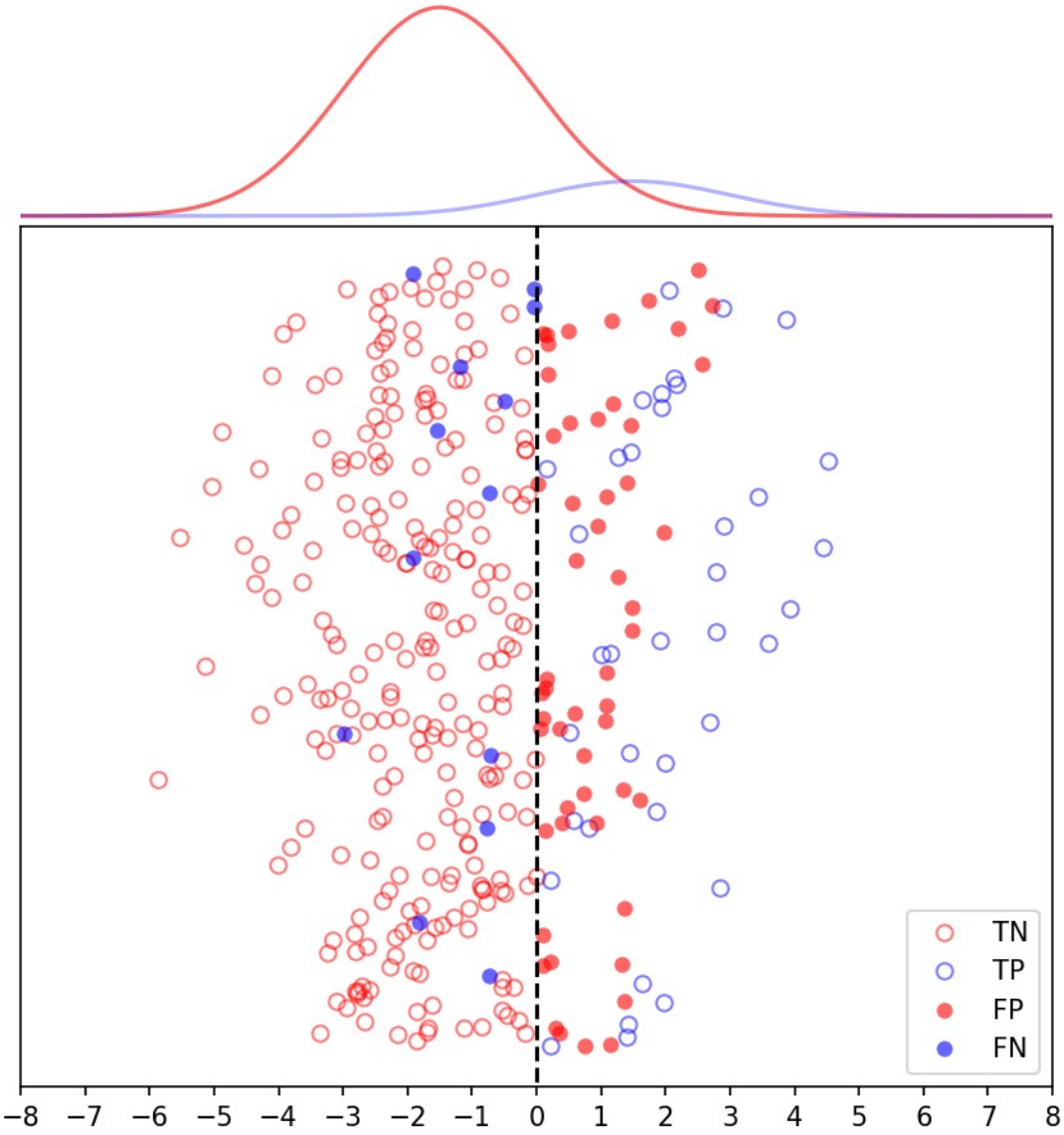
Example of ROC & PR

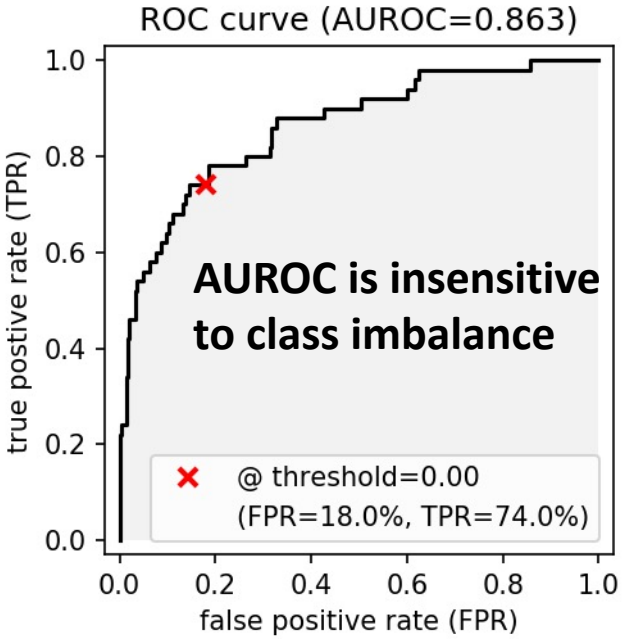


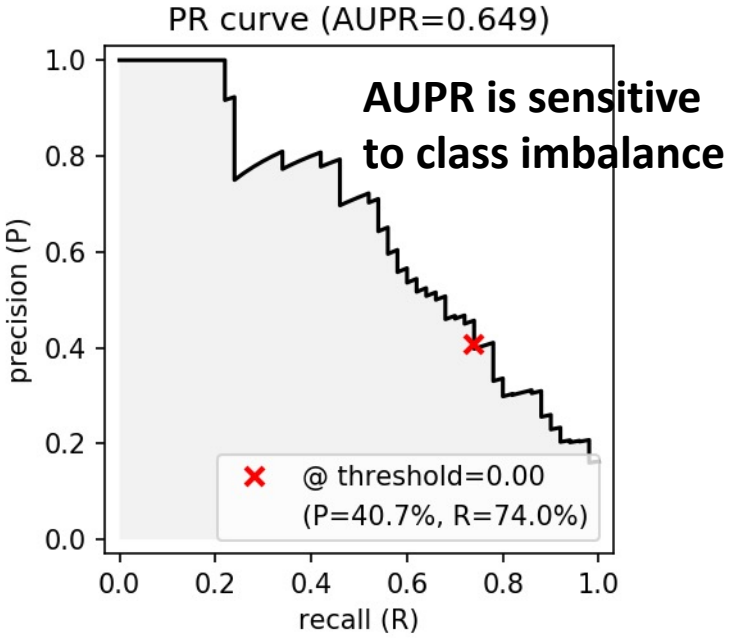




Example of ROC & PR







Comparing binary classifiers

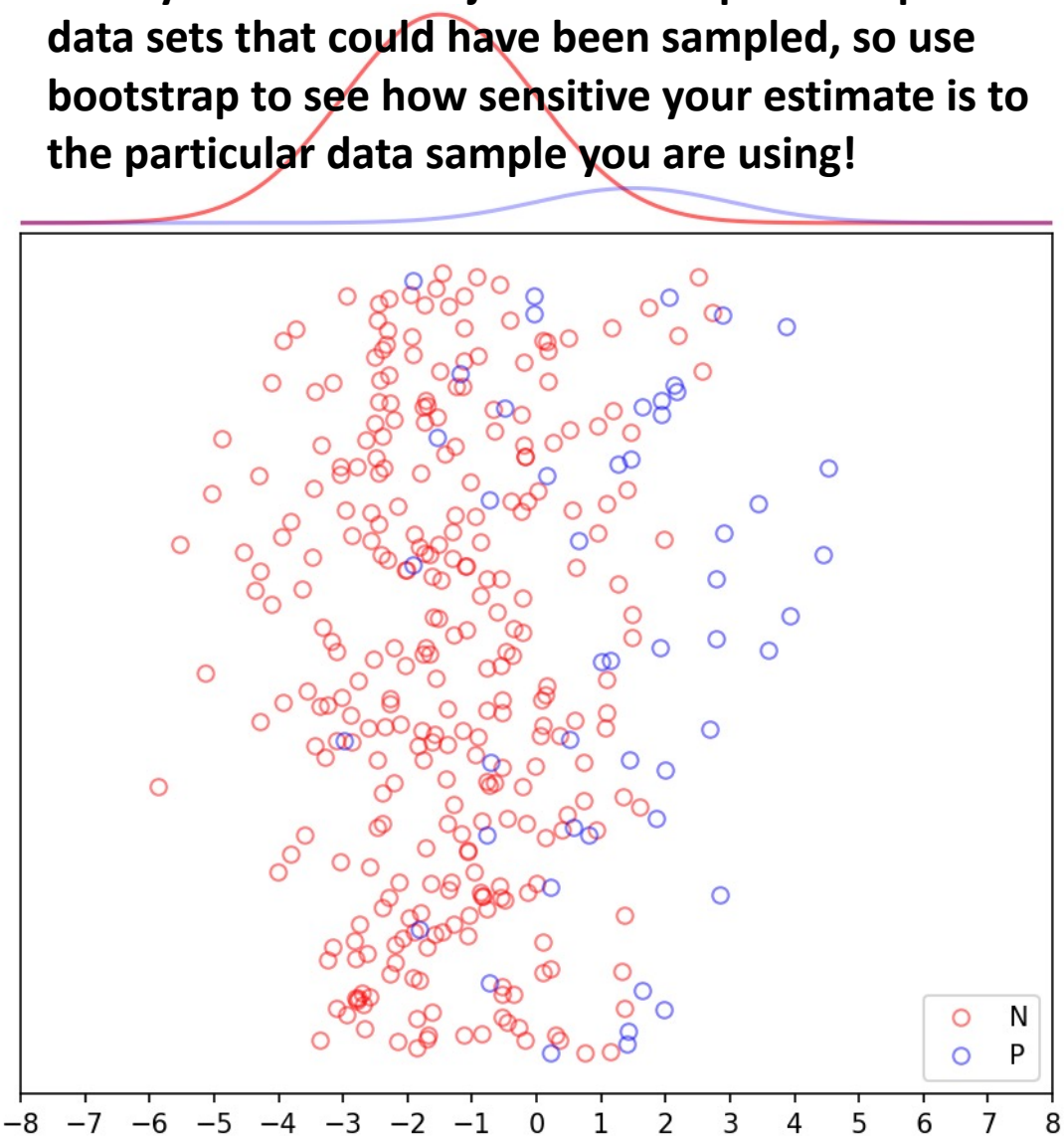
- But how do we compare classifiers *A* and *B* when each classifier has *two* numbers?
  - If stakeholders want a particular *TPR* or *FPR* or *P*, easy! e.g. "choose best cancer diagnostic @ 20% *FPR*"
  - Otherwise, no strict ordering over pairs (*FPR*, *TPR*) or (*R*, *P*)
- 1st idea: ***directly compare* the curves**
- Example for ROC curves:
  - If for each *FPR* the corresponding *TPR* for classifier *A* is always greater than the corresponding *TPR* for classifier *B*, then "the *TPR* of *A* dominates *TPR* of *B.*"
  - This does **not** mean that *A* and *B* are compared using the *same threshold*. It means that separate thresholds are chosen to give ***same FPR*** for each classifier.

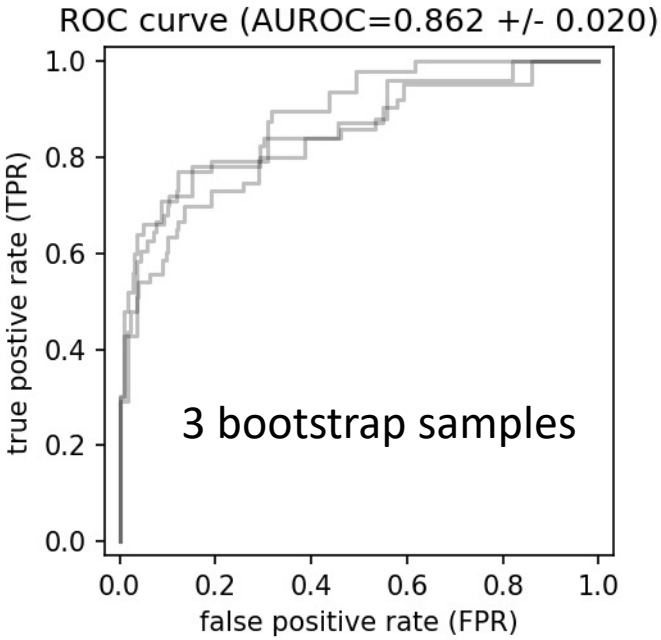
Comparing binary classifiers

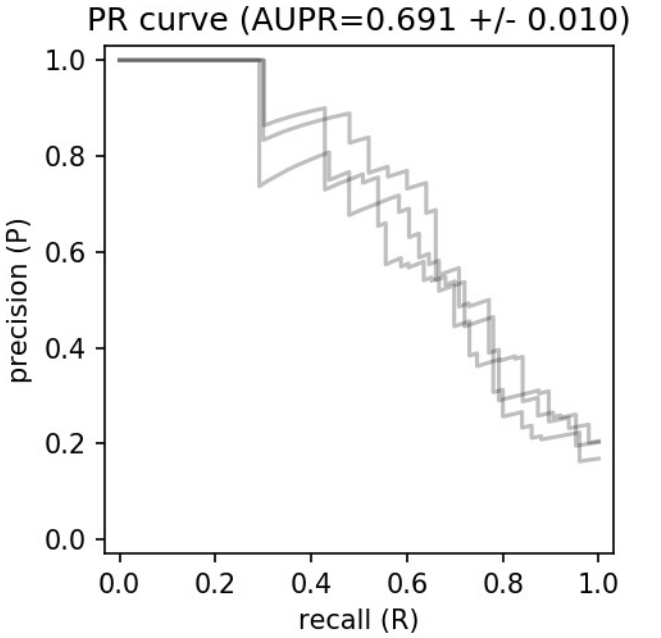
- What if the curves cross, and neither dominates?
- 2nd idea: **compare the *area* under each curve**
- If **AUROC** of *A* is greater than AUROC of *B*, then we can expect *A* to have greater *TPR* on average across all choices of threshold.
- If **AUPR** of *A* is greater than AUPR of *B*, then we can expect *A* to have greater precision *P* on average across all choices of recall (all choices of TPR).
- For highly imbalanced classes, consider using AUPR to assess performance, as it is sensitive to imbalance

Can **bootstrap** ROC & PR

Idea: your data set is just one sample from possible data sets that could have been sampled, so use bootstrap to see how sensitive your estimate is to the particular data sample you are using!

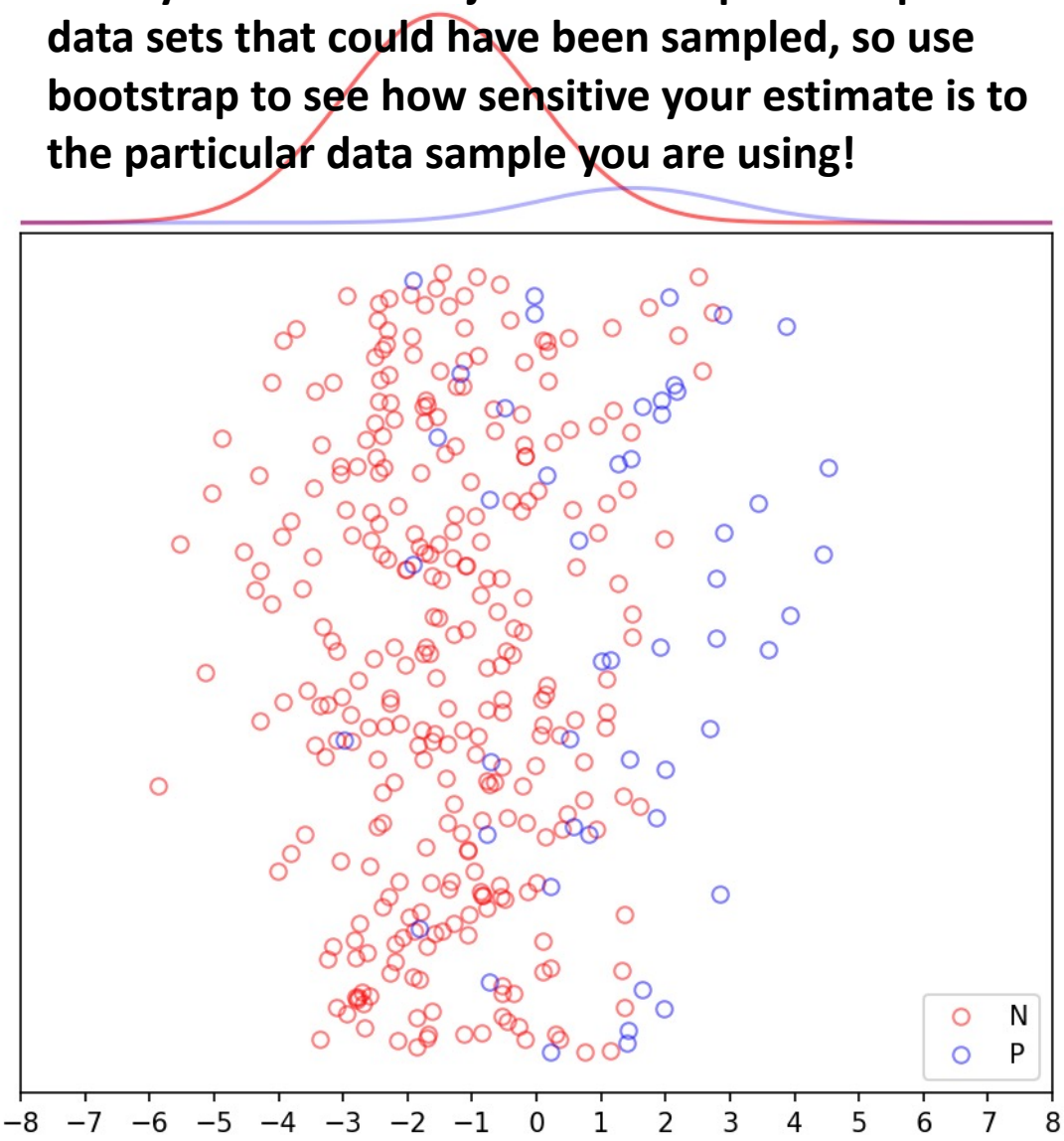


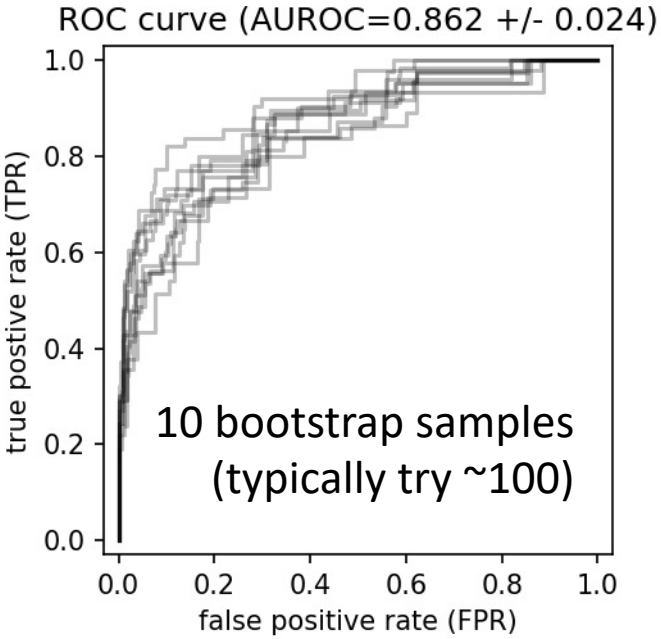


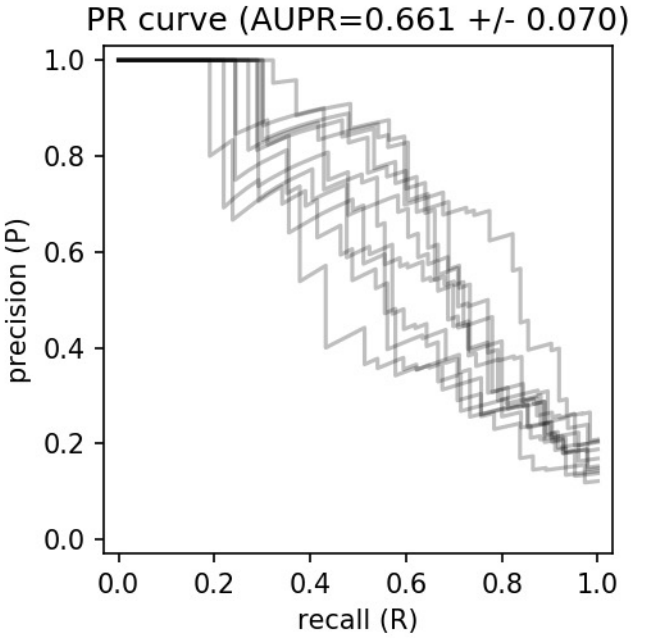


Can **bootstrap** ROC & PR

Idea: your data set is just one sample from possible data sets that could have been sampled, so use bootstrap to see how sensitive your estimate is to the particular data sample you are using!







Viola-Jones cascade: Classic paper using asymmetric loss for *training*

20,000 citations!

ACCEPTED CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION 2001

Rapid Object Detection using a Boosted Cascade of Simple Features

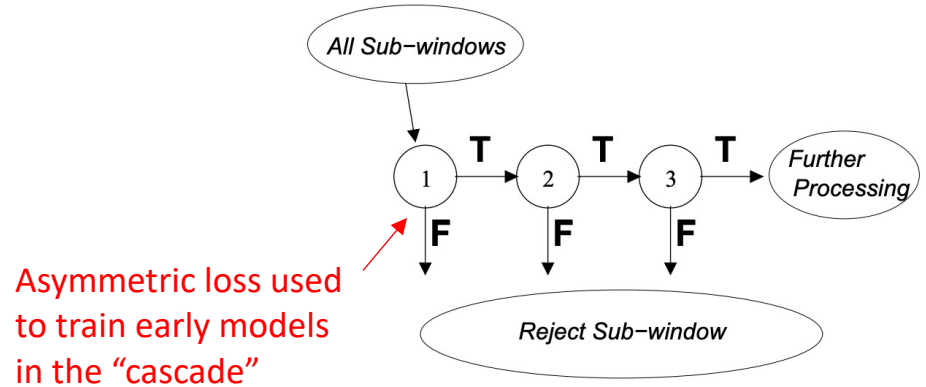
Paul Viola
viola@merl.com
Mitsubishi Electric Research Labs
201 Broadway, 8th FL
Cambridge, MA 02139

Michael Jones
mjones@crl.dec.com
Compaq CRL
One Cambridge Center
Cambridge, MA 02142

- Trains an early-stage detector to quickly reject image patches that "obviously not a face"
  - Loss penalizes false-negatives heavily, false positives only lightly
- Later stage classifier only sees the "hard" cases, trained on a more balanced loss function

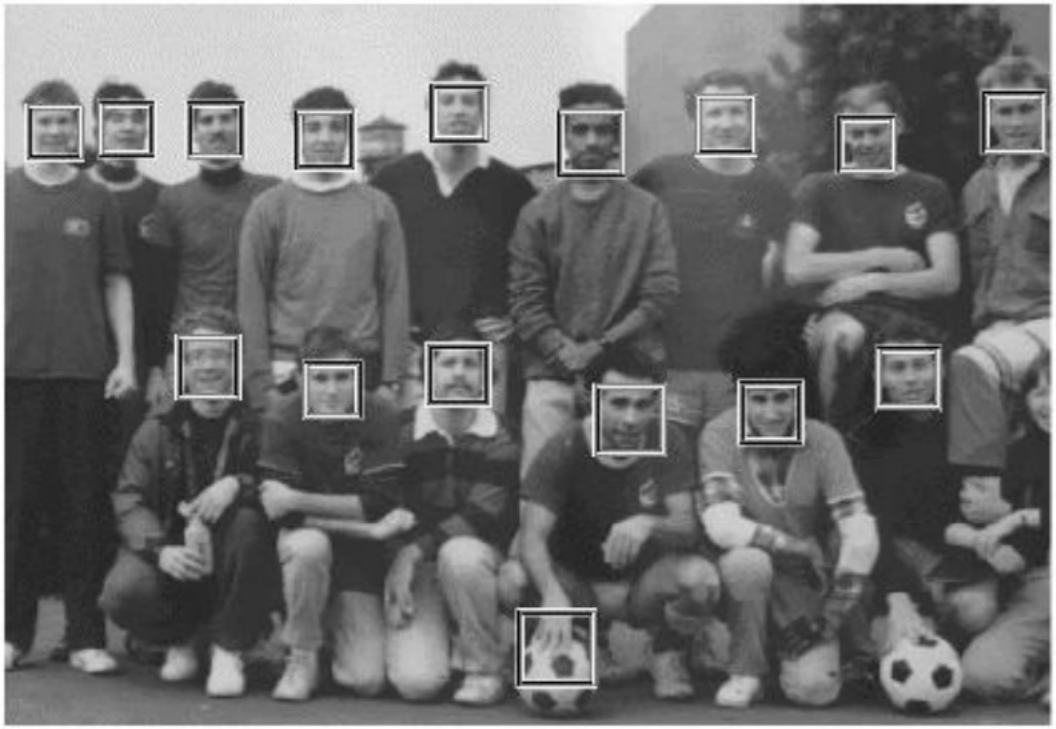
https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf

Viola-Jones face detector



Asymmetric loss used to train early models in the "cascade"

Figure 4: Schematic depiction of a the detection cascade.



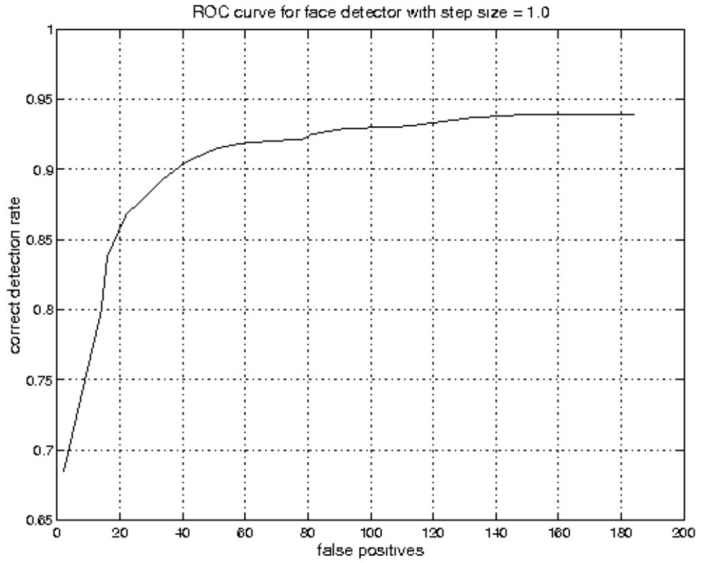
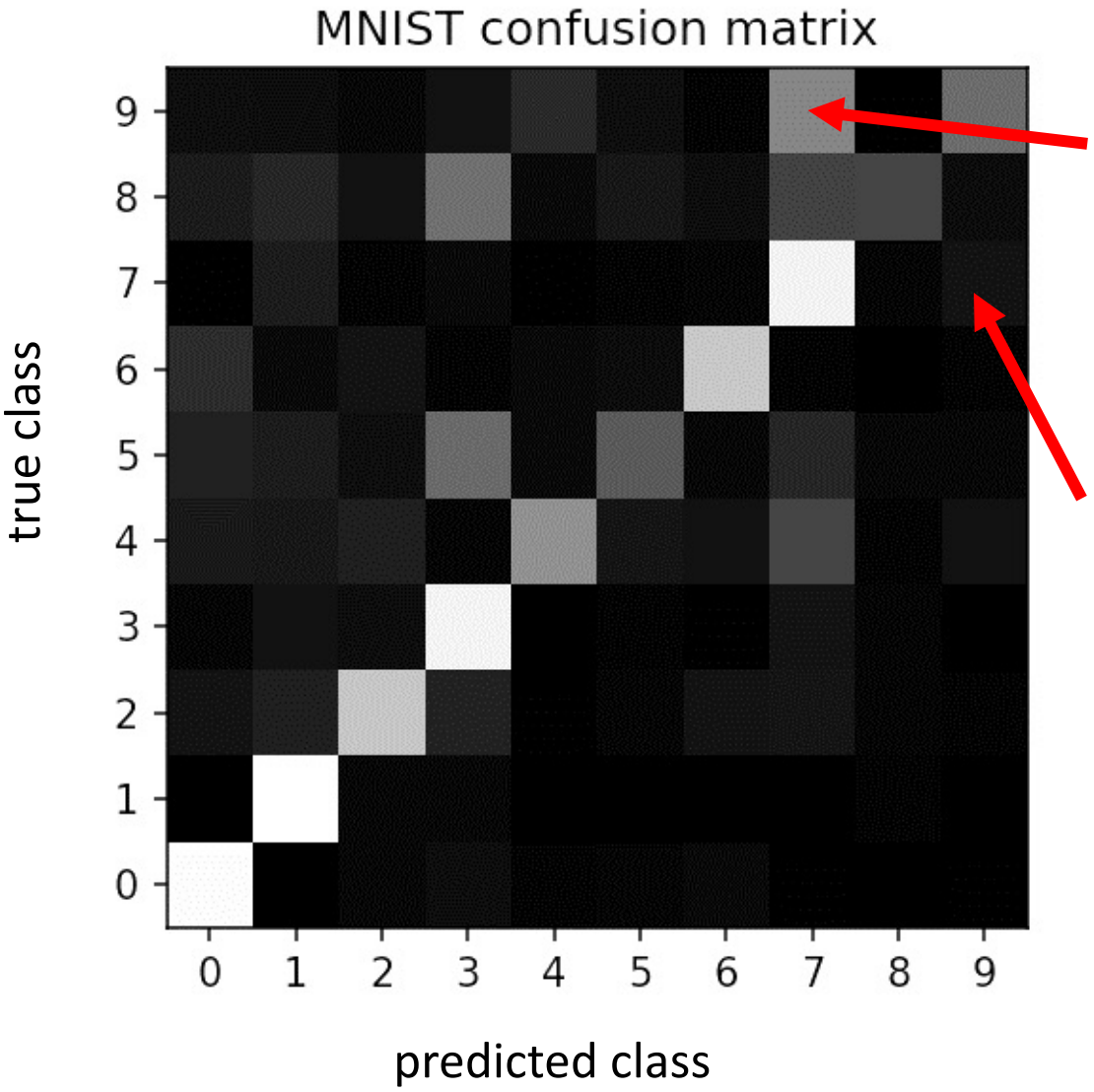


Figure 6: ROC curve for our face detector on the MIT+CMU test set. The detector was run using a step size of 1.0 and starting scale of 1.0 (75,081,800 sub-windows scanned).

Confusion matrix



Number of times a “9” digit was mistakenly predicted to be a “7”

Number of times a “7” digit was mistakenly predicted to be a “9”

sklearn.metrics.confusion_matrix

```
sklearn.metrics. confusion_matrix (y_true, y_pred, labels=None, sample_weight=None)
```
[source]

Compute confusion matrix to evaluate the accuracy of a classification

By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i but predicted to be in group j.

Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.

Read more in the User Guide.

Parameters:

y_true : *array, shape = [n_samples]*

Ground truth (correct) target values.

y_pred : *array, shape = [n_samples]*

Estimated targets as returned by a classifier.

labels : *array, shape = [n_classes], optional*

List of labels to index the matrix. This may be used to reorder or select a subset of labels. If none is given, those that appear at least once in `y_true` or `y_pred` are used in sorted order.

sample_weight : *array-like of shape = [n_samples], optional*

Sample weights.

Returns:

C : *array, shape = [n_classes, n_classes]*

Classification metrics

See the Classification metrics section of the user guide for further details.

metrics.accuracy_score (y_true, y_pred[, …])	Accuracy classification score.
metrics.auc (x, y[, reorder])	Compute Area Under the Curve (AUC) usin
metrics.average_precision_score (y_true, y_score)	Compute average precision (AP) from predi
metrics.balanced_accuracy_score (y_true, y_pred)	Compute the balanced accuracy
metrics.brier_score_loss (y_true, y_prob[, …])	Compute the Brier score.
metrics.classification_report (y_true, y_pred)	Build a text report showing the main classifi
metrics.cohen_kappa_score (y1, y2[, labels, …])	Cohen's kappa: a statistic that measures int
metrics.confusion_matrix (y_true, y_pred[, …])	Compute confusion matrix to evaluate the a
metrics.f1_score (y_true, y_pred[, labels, …])	Compute the F1 score, also known as balar
metrics.fbeta_score (y_true, y_pred, beta[, …])	Compute the F-beta score
metrics.hamming_loss (y_true, y_pred[, …])	Compute the average Hamming loss.
metrics.hinge_loss (y_true, pred_decision[, …])	Average hinge loss (non-regularized)
metrics.jaccard_score (y_true, y_pred[, …])	Jaccard similarity coefficient score
metrics.log_loss (y_true, y_pred[, eps, …])	Log loss, aka logistic loss or cross-entropy l
metrics.matthews_corrcoef (y_true, y_pred[, …])	Compute the Matthews correlation coefficie
metrics.multilabel_confusion_matrix (y_true, …)	Compute a confusion matrix for each class
metrics.precision_recall_curve (y_true, …)	Compute precision-recall pairs for different
metrics.precision_recall_fscore_support (…)	Compute precision, recall, F-measure and s
metrics.precision_score (y_true, y_pred[, …])	Compute the precision
metrics.recall_score (y_true, y_pred[, …])	Compute the recall
metrics.roc_auc_score (y_true, y_score[, …])	Compute Area Under the Receiver Operatin scores.
metrics.roc_curve (y_true, y_score[, …])	Compute Receiver operating characteristic
metrics.zero_one_loss (y_true, y_pred[, …])	Zero-one classification loss.

Surrogate loss functions

- When training a classifier to be cost-sensitive, we tend to use *weighted* version of its "natural" loss function
  - Logistic regression / neural nets: weighted version of NLL

$$-\sum_{i=1}^{N} C_1 y_i \ln \hat{y}_i + C_0 (1 - y_i) \ln(1 - \hat{y}_i)$$

  - SVM: weighted version of margin violation penalty

$$+ \quad C_1 \sum_{i\,:\,t_i=+1} \xi_i \quad + \quad C_0 \sum_{i\,:\,t_i=-1} \xi_i$$

- Used *even if test performance measured by AUC/AUPR*
  - When our training loss doesn't match the test metric we plan to use, it is called a ***surrogate loss*** or ***proxy loss***.

Surrogate loss functions

- If AUC or AUPR is what we care about, why not train to maximize it *directly*? Why a surrogate?

AUC Optimization vs. Error Rate Minimization

Corinna Cortes* **and Mehryar Mohri**
AT&T Labs – Research

Abstract

The *area under an ROC curve* (AUC) is a criterion used in many applications to measure the quality of a classification algorithm. However, the objective function optimized in most of these algorithms is the error rate and not the AUC value. We give a detailed statistical analysis of the relationship between the AUC and the error rate, including the first exact expression of the expected value and the variance of the AUC for a fixed error rate. Our results show that the average AUC is monotonically increasing as a function of the classification accuracy, but that the standard deviation for uneven distributions and higher error rates is noticeable. Thus, algorithms designed to minimize the error rate may not lead to the best possible AUC values. We show that, under certain conditions, the global function optimized by the RankBoost algorithm is exactly the AUC. We report the results of our experiments with RankBoost in several datasets demonstrating the benefits of an algorithm specifically designed to globally optimize the AUC over other existing algorithms optimizing an approximation of the AUC or only locally optimizing the AUC.

Because optimization of an AUC loss is very hard

- Gradients hard to compute
- Non-convex optimization

Still, smart people have tried and with some success!

http://papers.nips.cc/paper/2518-auc-optimization-vs-error-rate-minimization.pdf (Cortes & Mohri)
http://jmlr.csail.mit.edu/papers/volume4/freund03a/freund03a.pdf (RankBoost)

ROC and PR curves for multi-class?

- Multiclass ROC/PR an area of active research, e.g. http://proceedings.mlr.press/v97/kleiman19a/kleiman19a.pdf
- Simplest multiclass extensions treat each of the *K* predictions as "One-vs-Rest" binary, then compute an ROC or PR curve from *K*N* binary predictions
  - ***Micro average***: treat each binary prediction $\hat{y}_{ik}$ and binary target y_{ik} as independent, and compute standard ROC or PR curve (and AUROC or AUPR).
  - ***Macro average***: compute *K* separate curves (ROC or PR) and average their *y*-axis value at each unique *x*-axis point (TPR for ROC, P for PR) to get one 'average' curve. (Note: class imbalances now averaged out even for PR.)

Example of *3*-class ROC averaging

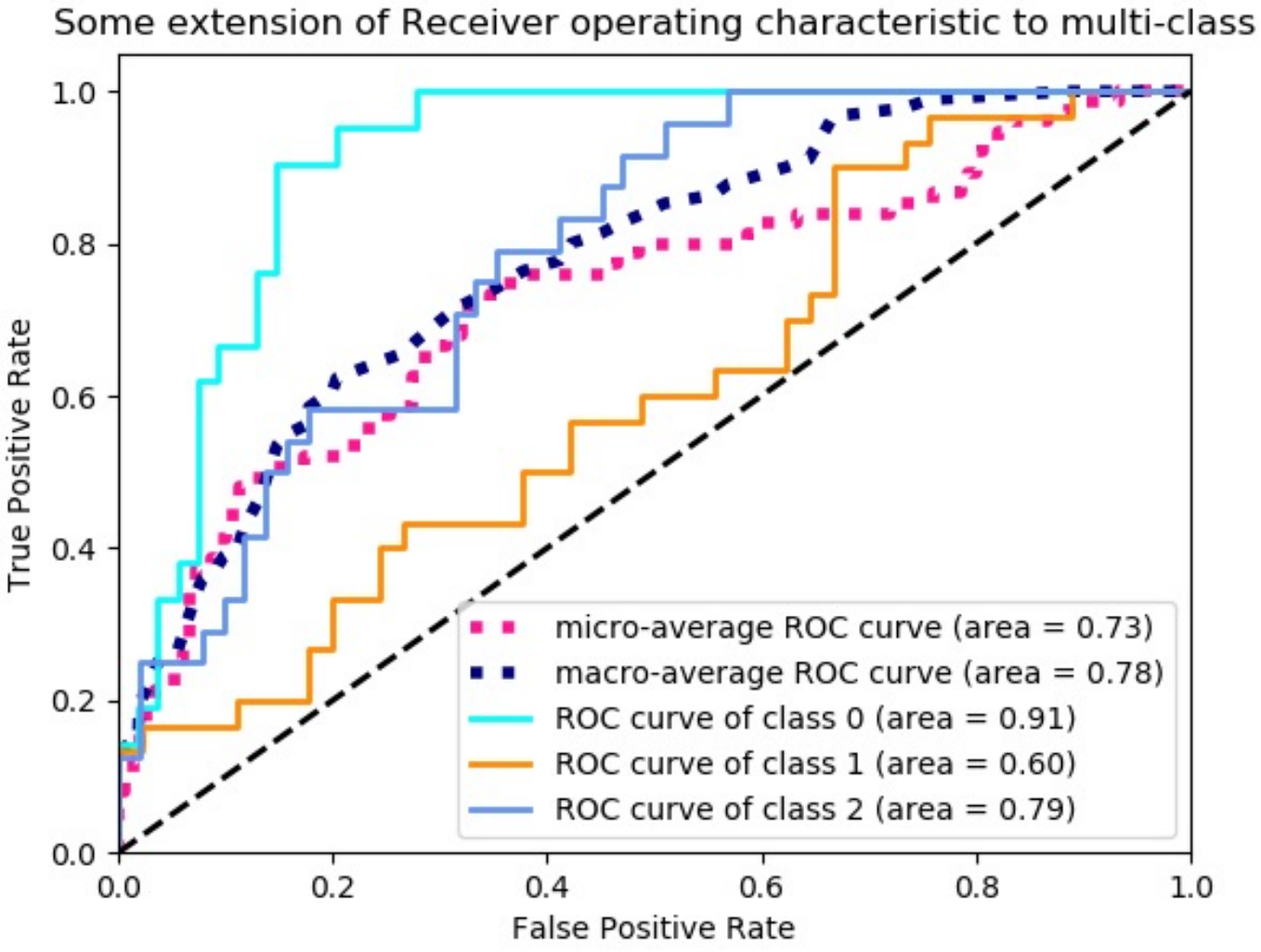


Image from https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

PRML Readings

§1.5.2 Minimizing the expected loss

§1.5.5 Equation (1.91) and Figure 1.29 ONLY