

Classification, Logistic Regression, Evaluating Classifiers

Data 100, Fall 2019

Suraj Rampure

Agenda

- Logistic Regression
- Classification and Evaluating Classifiers
- Linear Separability

Linear vs. Logistic Regression

In a **linear regression model** with p features, our goal is to predict a quantitative variable (i.e., some real number) from those features. Our output can be any real number.

$$\hat{y} = \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \dots + \hat{\beta}_p x_p = \underbrace{x^T}_{\text{vector}} \underbrace{\hat{\beta}}_{\text{vector}}$$

In a **logistic regression model** with p features, our **goal** is to predict a **categorical variable** from those features.

$$\hat{y} = P(Y=1|x) = \sigma(x^T \hat{\beta})$$

- The output of logistic regression is always between 0 and 1, i.e. it is **quantitative!**
- Gives probability under our model that the category is 1.
- Our goal is to perform binary classification – to predict either 0 or 1.

Logistic Regression and the Logistic Function

In logistic regression, we're modelling the probability that an observation belongs to class 1 (as opposed to class 0).

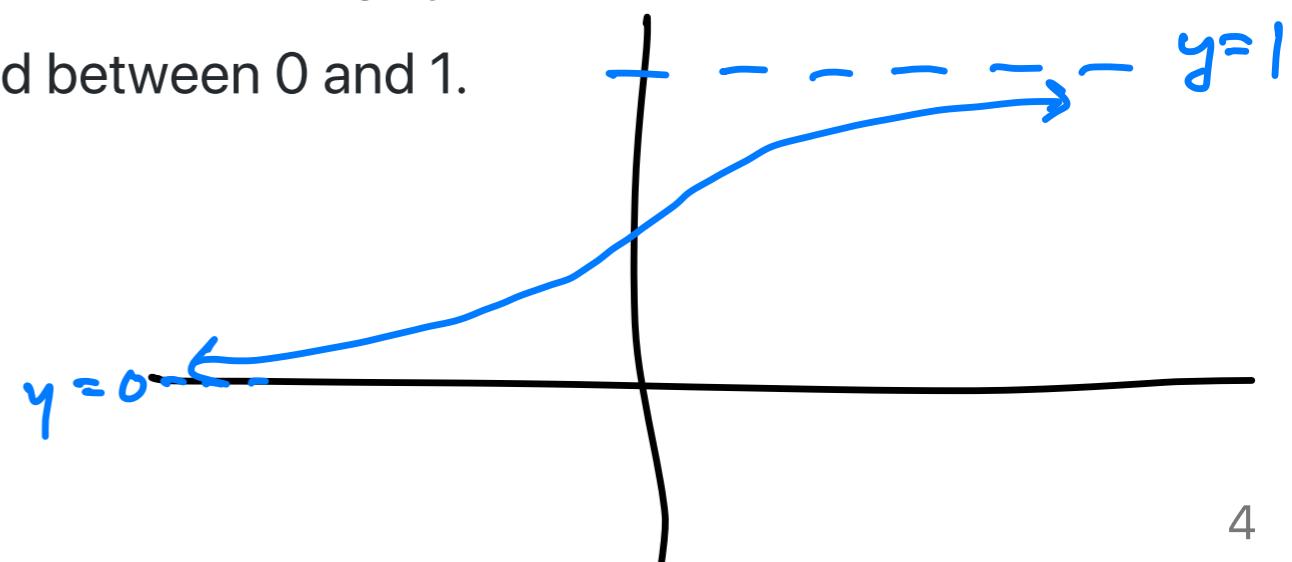
$$\hat{y} = P(Y = 1 | \vec{x}) = \sigma(\vec{x}^T \vec{\beta})$$

Our model looks like the linear regression model, except with a $\sigma(\cdot)$ around it.

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

$$P(Y=1 | x)$$

- $\sigma(t)$ came from the assumption that the log-odds of p is roughly linear.
- Has a nice "s" shape, and has outputs bounded between 0 and 1.



Cross Entropy Loss

Cross entropy loss, for a single point, is given by:

$$\text{loss} = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

or, in terms of our observation, and for the logistic model:

$$\hat{y} = \sigma(x^\top \beta)$$

$$R(\vec{\beta}) = -\frac{1}{n} \sum_{i=1}^n \left(y_i \log \sigma(x_i^\top \vec{\beta}) + (1 - y_i) \log(1 - \sigma(x_i^\top \vec{\beta})) \right)$$

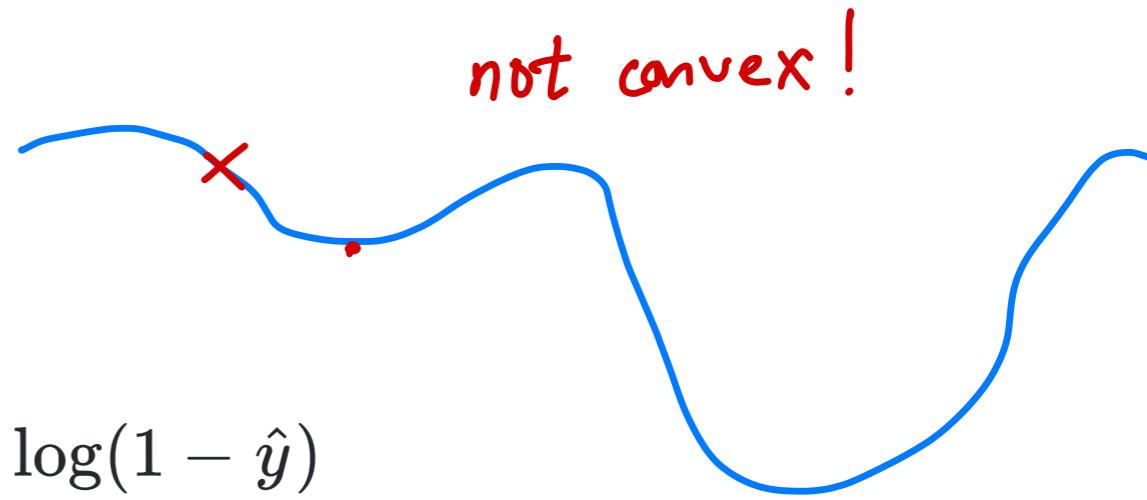
Benefits of cross entropy loss over L_2 loss, for logistic regression:

- The average loss (empirical risk) over all the data is guaranteed to be convex when used with logistic regression, and thus easier to optimize (we won't prove this).
- It more strongly penalizes very bad predictions.
- It has roots in probability and information theory (maximum likelihood estimation, KL divergence).

$y : 0 \text{ or } 1$
 ~~$y \in [0, 1]$~~

$$L_2 : (y - \hat{y})^2$$

Note: We can use L_2 loss for logistic regression, but usually don't.



not convex!

Classification

- Our motivation for performing logistic regression was to predict categorical labels. Specifically, we were looking to perform binary classification, i.e. classification where our outputs are 1 or 0.
- However, the output of logistic regression is a continuous value in the range $[0, 1]$, which we interpret as a probability – specifically,

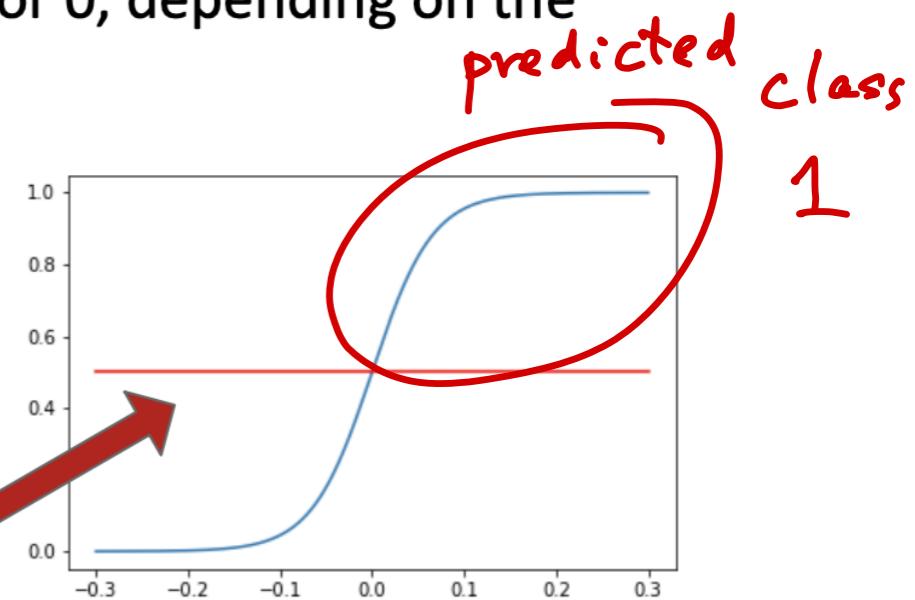
$$P(Y = 1|\vec{x}) = \sigma(\vec{x}^T \hat{\beta})$$

- In order to predict a 1 or 0, we pair our logistic model with a decision rule, or classification rule.

Decision Rules

Given an observation \vec{x} , the following **decision rule** outputs 1 or 0, depending on the probability that our model assigns to \vec{x} belonging to class 1:

$$\text{classify}(\vec{x}) = \begin{cases} 1, & P(Y = 1|\vec{x}) \geq 0.5 \\ 0, & P(Y = 1|\vec{x}) < 0.5 \end{cases}$$



- Note: We **don't need** to set our **threshold to 0.5**. Depending on the type of errors we want to minimize, we can increase or decrease it.
- Furthermore, depending on the domain, we may want to choose not to provide a classification at all, if our probability of being in class 1 is too close to 0.5.

Evaluating Classifiers

Evaluating the Effectiveness of a Classifier

Suppose we train a binary classifier, and suppose `y` represents actual values, and `y_pred` represents predicted values.

Recall (no pun intended) the following definitions:

- True Positives: `TP = np.count_nonzero((y == y_pred) & (y_pred == 1))`
- True Negatives: `TN = np.count_nonzero((y == y_pred) & (y_pred == 0))`
- False Positives: `FP = np.count_nonzero((y != y_pred) & (y_pred == 1))`
- False Negatives: `FN = np.count_nonzero((y != y_pred) & (y_pred == 0))`

Then, we have the following definitions:

- Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$
- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$

confusion matrix

		Actual	
		1	0
Predicted	1	TP	FP
	0	FN	TN

as threshold T

confusion matrix

		1 Actual	0
Predicted	1	TP	FP
	0	FN	TN

Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

eg "of all people we predicted to have cancer, what proportion actually does?"

Recall = $\frac{\text{TP}}{\text{TP} + \text{FN}}$

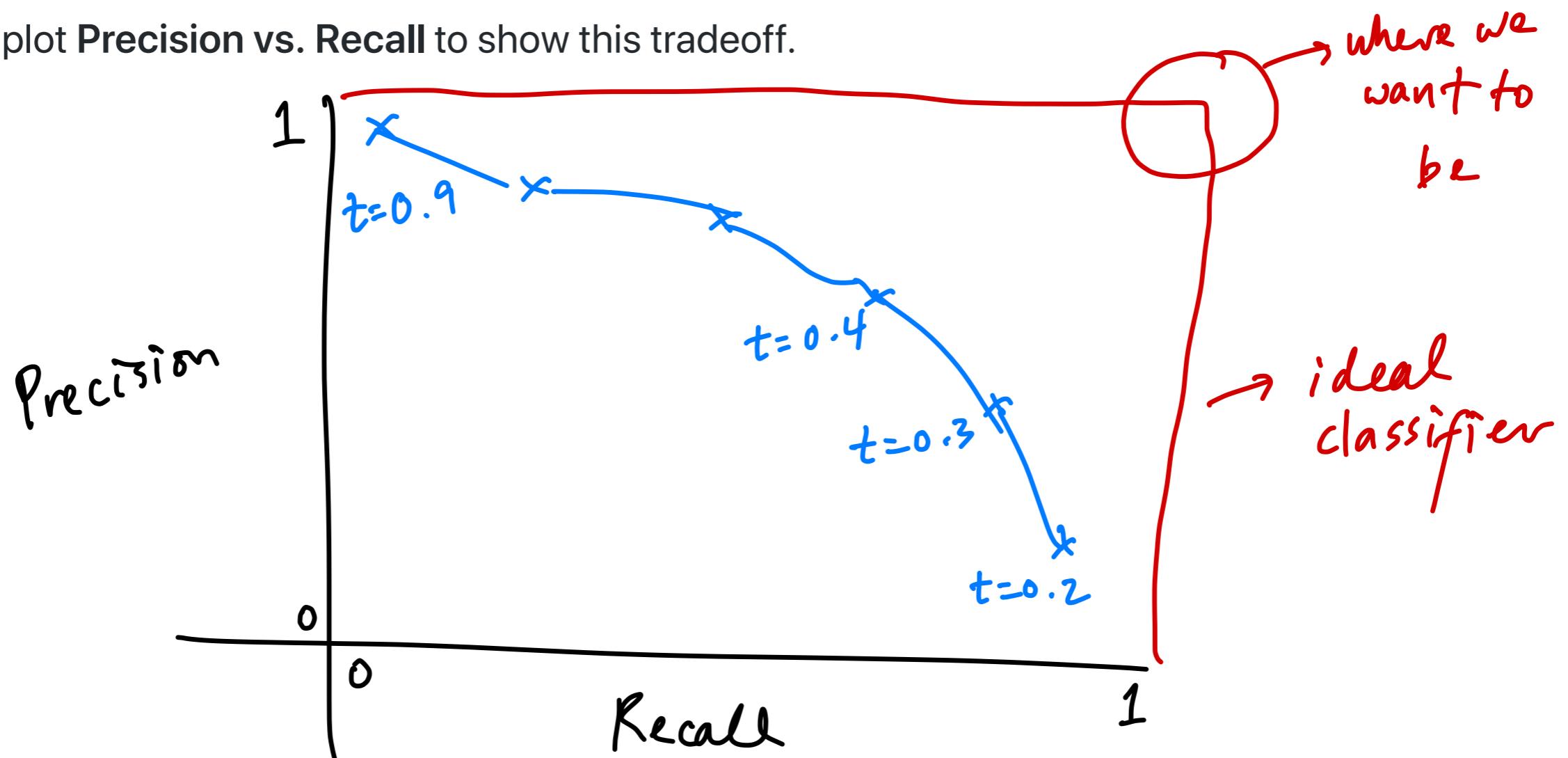
denominator
stays constant

eg "of all people that have cancer, what proportion did we detect?"

PR Curves

- Ideally, we would have a precision of 1 and recall of 1.
- As our threshold increases, precision tends to increase, while recall decreases.

We can plot Precision vs. Recall to show this tradeoff.



Evaluating the Effectiveness of a Classifier

We now introduce a new metric:

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN}$$



This tells us **the proportion of observations that were actually negative that we classified as positive**. You can think of this as being the "false alarm rate."

Similarly, we have

$$\text{True Positive Rate (TPR)} = \text{Recall} = \frac{TP}{TP + FN}$$



which is **the proportion of observations that were actually positive that we classified as positive**. You can think of this as being the "miss rate". (Note, this is the same as recall.)

Question: As our classification threshold increases, how do both of these quantities change?

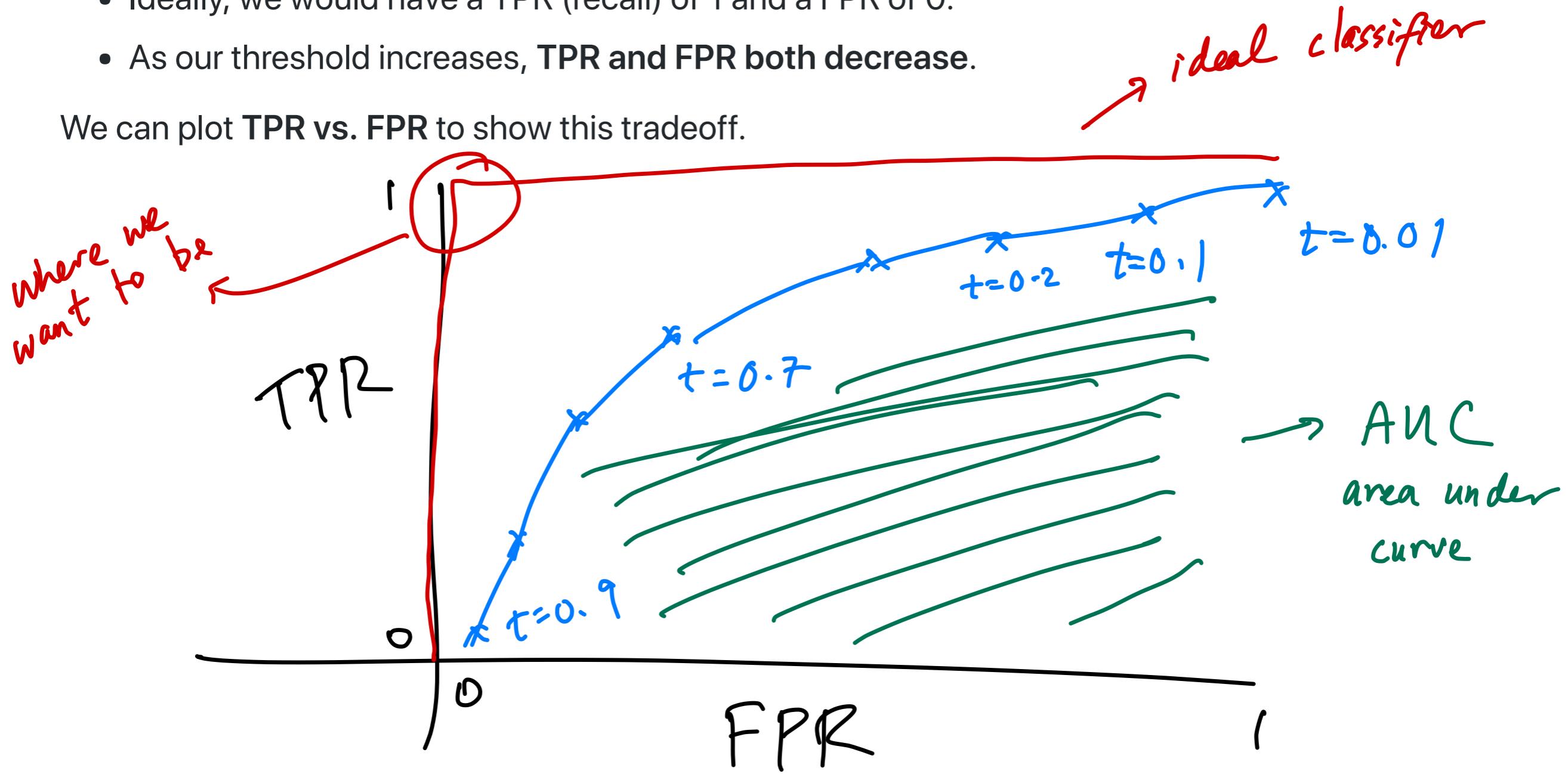
both denominators are constant, while #predicted positives decrease



ROC Curves

- Ideally, we would have a TPR (recall) of 1 and a FPR of 0.
- As our threshold increases, TPR and FPR both decrease.

We can plot TPR vs. FPR to show this tradeoff.



Area Under Curve

- The AUC (area under curve) metric for a specific curve (ROC or PR) gives you a single number to evaluate your classifier's performance.
- It **does not** tell you how to choose your threshold. To do that, you should look at the ROC / PR curve and make a decision based off of your classification goal.

Linear Separability

Linear Separability

logistic

The goal of ~~linear~~ regression is to model the probability of a point belonging to a class. Typically, we do this when there is some uncertainty, i.e. overlap, in our training set.

In some cases, we are able to draw a *linear decision boundary* to separate our data.

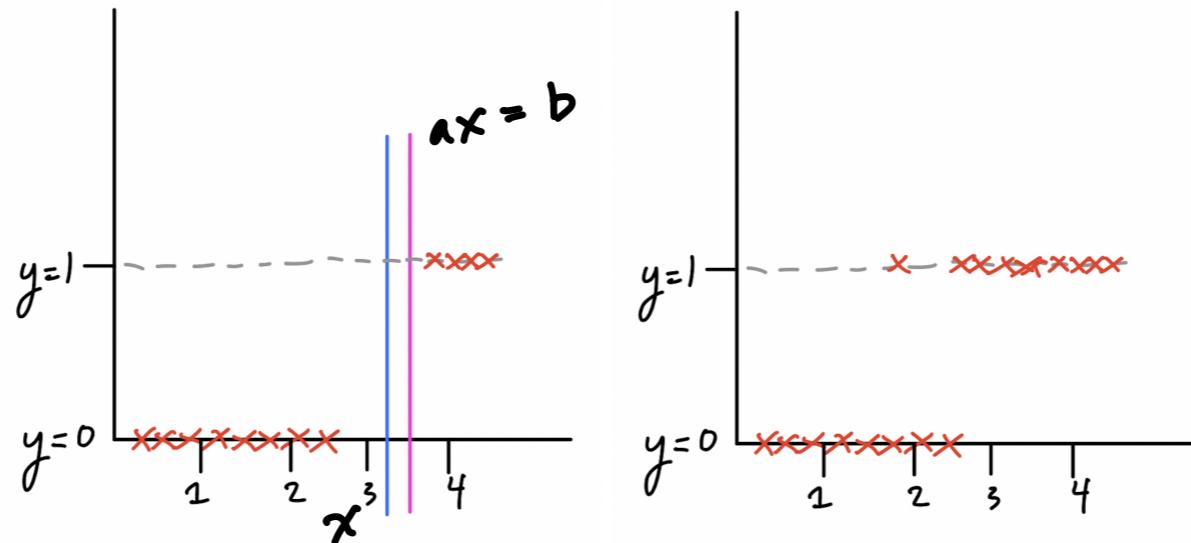


Above, we see that our data is **linearly separable**. We can draw a line (infinitely many lines, in fact) between the clusters of red dots and green dots.



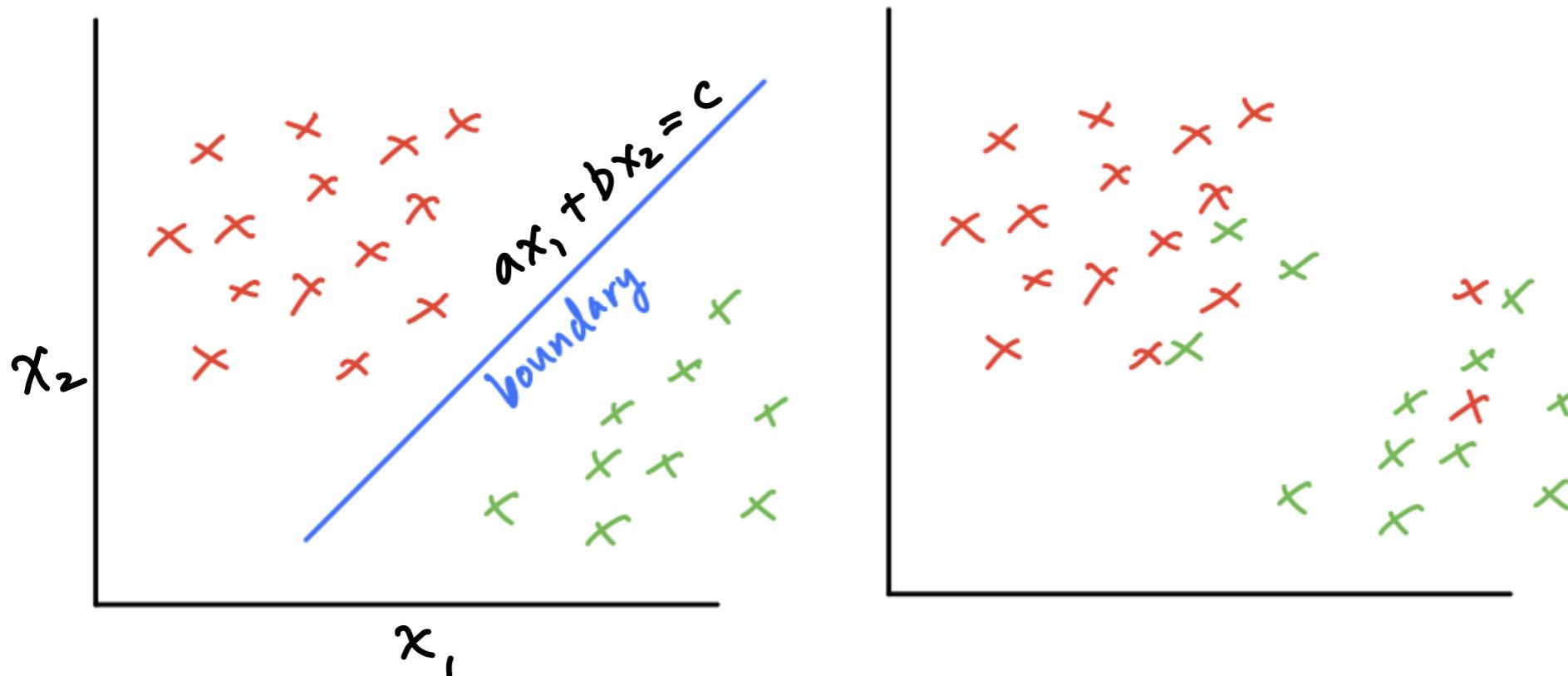
This is not the case in the second example.

Again, let's look at data in 1D, but plotted in 2D (one dimension is the value of our variable, the second dimension is the label, 0 or 1 --- this is equivalent to the drawings on the previous slide).



- On the left, we have an example of linearly separable data. In the blue and purple are two possible hyperplanes that can separate our data.
- However, on the right, we have non-linearly separable data. In this case we would use a tool like logistic regression to model probabilities.

Linear Separability in Two Dimensions



Here, we have examples of both linearly separable and non-linearly separable data in two dimensions. Here, our data is truly two dimensional, as our feature space has two components --- an x_1 and a x_2 . The class is represented by the color (Y).

Formal Definition of Linear Separability

We say data with d dimensions is linearly separable iff you can draw a degree $d - 1$ hyperplane that completely separates the points.

An equivalent definition: our data is linearly separable iff the following inequality perfectly classifies it (i.e. with 100% accuracy), for some c :

$$x^T \beta \geq c$$

For example...

$$\beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p = c$$

Note: $\alpha x_1 + b x_2 = c$
is a deg-1 hyperplane

→ rearrange to
 $x_2 = \alpha x_1 + \beta$
(deg-1 line)

- If we have a single feature x , our data is linearly separable iff we can find some β, c s.t. $\beta x \geq c$ is a perfect classifier
- If we have three features x_1, x_2, x_3 , our data is linearly separable iff we can find some $\beta_1, \beta_2, \beta_3, c$ s.t. $\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \geq c$ is a perfect classifier
- Note: One or more of these β may be 0!

Cross Entropy Loss and Linear Separability

Recall, when using cross entropy loss, our empirical risk is of the form

$$R(\vec{\beta}) = - \left(y \log \sigma(x^T \vec{\beta}) + (1 - y) \log(1 - \sigma(x^T \vec{\beta})) \right)$$

If our data is linearly separable, we don't really need to perform logistic regression, since there's no "uncertainty" --- we can just skip straight to finding a separating hyperplane. However, we don't typically know this beforehand, and so we may end up trying to perform logistic regression on linearly separable data.

Questions:

- If our data is linearly separable, what's the minimum value of our average cross entropy loss?
- What value of $\vec{\beta}$ provides this? $\pm \infty$
- What problems does this cause? How can we account for this? *large $\beta \rightarrow$ overfitting!*

Regularized Cross Entropy Loss

In order to prevent our $\vec{\beta}$ from diverging, we can regularize our empirical cross entropy loss.

$$R(\vec{\beta}) = -\frac{1}{n} \sum_{i=1}^n \left(y \log \sigma(x^T \vec{\beta}) + (1 - y) \log(1 - \sigma(x^T \vec{\beta})) \right) + \lambda S(\vec{\beta})$$

tinyurl.com/surajfeedback