Classification

Sewoong Oh

CSE/STAT 416
University of Washington

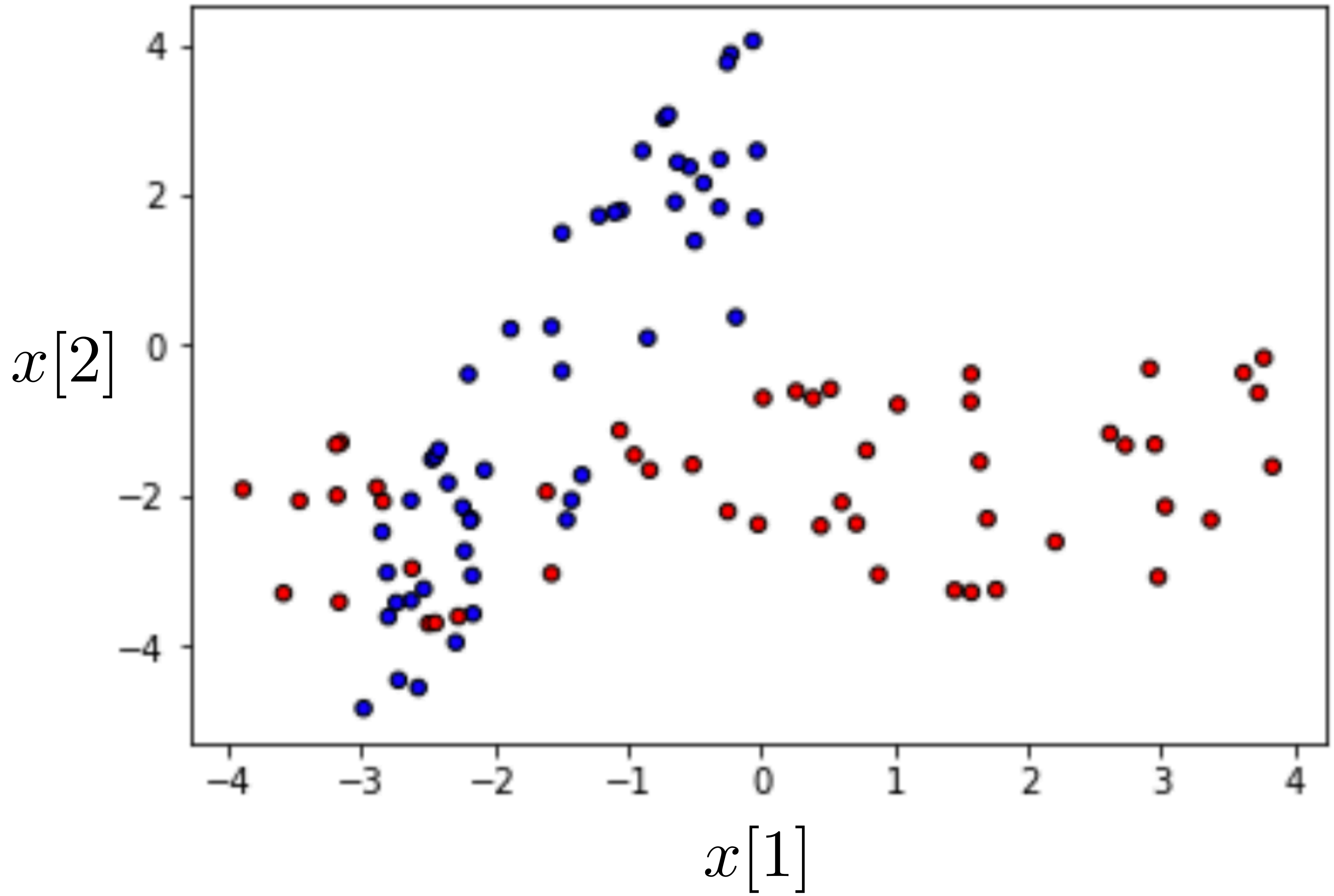
Boolean Classification

Boolean classification

- **Supervised learning** is training a predictor from labelled examples:
- There are two types of supervised learning
  - 1. **Regression**: the output variable ***y*** to be predicted is **real valued** scalar or a vector
  - 2. **Classification**: the output variable ***y*** to be predicted is categorical
    - 2.1 **Boolean classification**: there are two classes
    - 2.2 **Multi-class classification**: multiple classes

- We study Boolean classification in this chapter
- We denote two classes by -1 and 1, often corresponding to {FALSE,TRUE}
- for a data point (x_i, y_i), the value $y_i \in \{-1, 1\}$ is called the **class** or **label**
- A **Boolean classifier** predicts label ***y*** given input ***x***

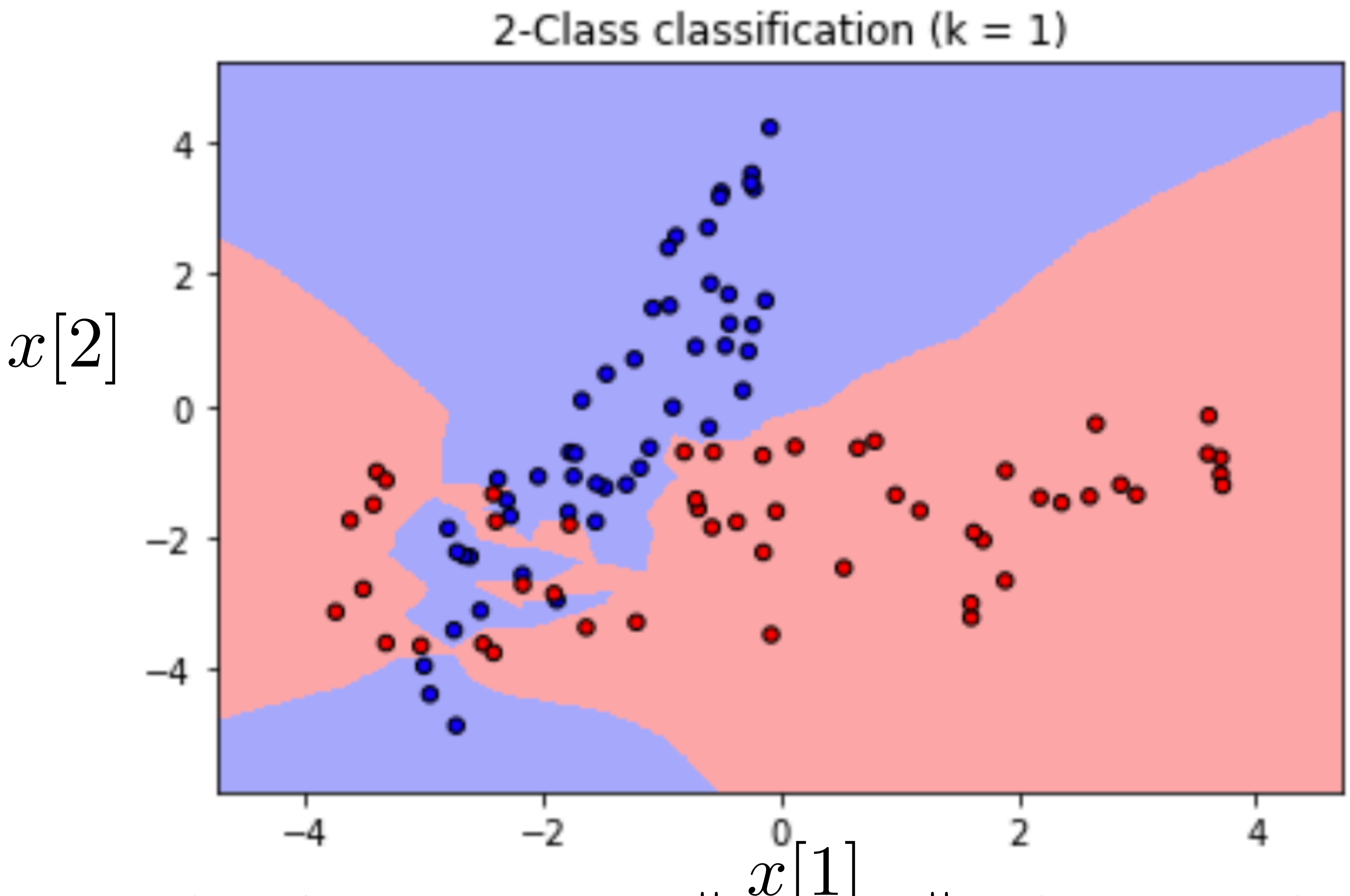
Classification



- in this example $x_i \in \mathbf{R}^2$
- Red points have label y_i=-1, blue points have label y_i=1
- We want a predictor that maps any ***x*** into prediction $\hat{y} \in \{-1, 1\}$

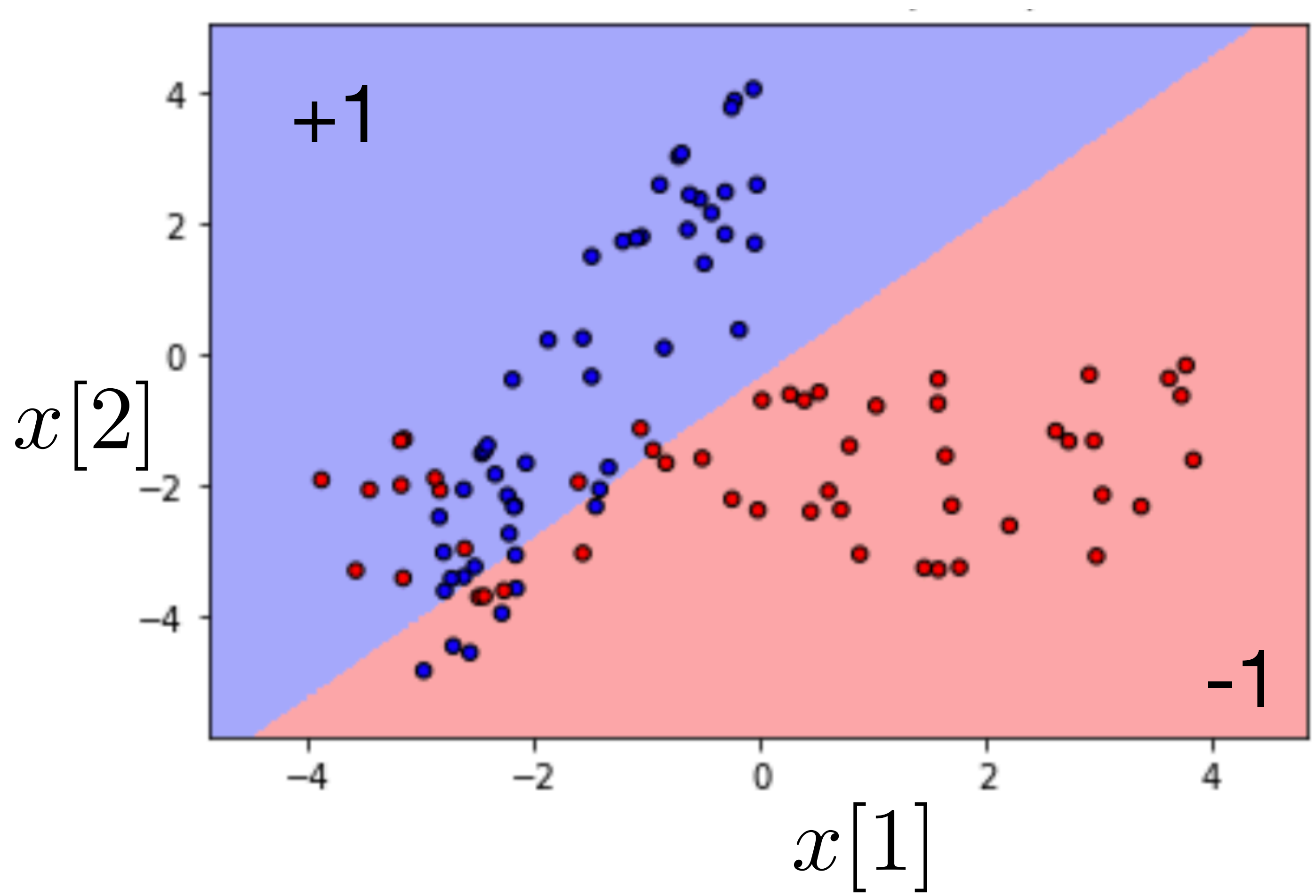
Example: nearest neighbor

- Trained on 100 samples



- given x, let $k = \arg\min_i \|x - x_i\|$, then predict $\hat{y} = y_k$
- Red region is the set of x for which prediction is -1
- Blue region is the set of x for which prediction is 1
- Zero training error, but overfitting

Example: linear classifier



- Trained on 100 samples

- Treat it as linear regression problem on ***x***
- Which trains a linear model: $f(x) = w_0 + w_1 x[1] + w_2 x[2]$ on L2 loss, treating the labels as real values +1 and -1
- Then predicts: $\hat{y} = \mathrm{sign}(f(x))$
- 18% mis-classified in training data
- true positive=42,false negative=8,true negative=38,false negative=12

Example: sentiment analysis

List of positive words	List of negative words
great, awesome, good, amazing,…	bad, terrible, disgusting, sucks,…

x_i

Sushi was <u>great</u>, the food was <u>awesome</u>, but the service was <u>terrible</u>.

$$\hat{y}_i = \text{sign}(\text{number of positive words} - \text{number of negative words})$$

- If we have access to the list of positive and negative words, then we could count them to give a score f(x) and take the sign for estimating the sentiment in {positive,negative}

Example: sentiment analysis

- **Linear classifier**

Sushi was <u>great</u>, the food was <u>awesome</u>, but the service was <u>terrible</u>.

$$h_j(x) = \text{how many times the word appears}$$
$$w_j = \text{how positive is that word}$$

$$\hat{y}_i = \text{sign}(w_0 + w_1 h_1(x) + w_2 h_2(x) + \cdots)$$

Word	Weight
good	1.0
great	1.5
awesome	2.7
bad	-1.0
terrible	-2.1
awful	-3.3
restaurant, the, we, where, …	0.0
…	…

- Without manually constructed list, we can use ML to learn the sentiment of the words (parameters ***w***), and then compute a score f(x)

Confusion Matrix

Two types of error

- When measuring performance of a predictor on Boolean classification
- each input data x_i has a label $y_i \in \{-1, 1\}$
- each corresponding prediction is $\hat{y}_i \in \{-1, 1\}$
- Only four possible combinations of $(\hat{y}_i, y_i)$:
  - **true positive** if $\hat{y}_i = 1$ and $y_i = 1$
  - **true negative** if $\hat{y}_i = -1$ and $y_i = -1$
  - **false negative** of **type II error** if $\hat{y}_i = -1$ and $y_i = 1$
  - **false positive** of **type I error** if $\hat{y}_i = 1$ and $y_i = -1$

We can represent the performance with a confusion matrix

- Define the **confusion matrix**: (some people use the transpose of C)

$$C = \begin{bmatrix} \#\text{ true negatives} & \#\text{ false negatives} \\ \#\text{ false negatives} & \#\text{ true positives} \end{bmatrix} = \begin{bmatrix} C_{tn} & C_{fn} \\ C_{fp} & C_{tp} \end{bmatrix}$$

- $C_{tn} + C_{fn} + C_{fp} + C_{tp} = N$

- $N_n = C_{tn} + C_{fp}$ is the number of negative examples

- $N_p = C_{fn} + C_{tp}$ is the number of positive examples

- Diagonal entries give numbers of correct prediction

- Off-diagonal entries give numbers of incorrect predictions

Some Boolean classification measures

- **Confusion matrix** $\begin{bmatrix} C_{tn} & C_{fn} \\ C_{fp} & C_{tp} \end{bmatrix}$
- The basic error measures are:
  - **False positive** rate is C_{fp}/N
  - **False negative** rate is C_{fn}/N (e.g. medical diagnosis)
  - **Error rate** is $(C_{fn} + C_{fp})/N$
  - **Accuracy** is $(C_{tn} + C_{tp})/N$
- High accuracy does not always mean good classifier
  - For example, 99% population does not have cancer, and predicting always no cancer achieves accuracy 99%

- Error measures also used:
  - **True positive rate** or **sensitivity** or **recall** is C_{tp}/N_p (=0 in the example)
  - **False alarm rate** is C_{fp}/N_n (=0 in the example)
  - **Specificity** or **true negative rate** is C_{tn}/N_n (=1 in the example)
  - **Precision** is $C_{tp}/(C_{tp} + C_{fp})$ (=0 in the example)

Neyman-Pearson error

- **Neyman-Pearson error** over a data set is

$$\kappa C_{fn}/N + C_{fp}/N$$

- A scalarization of our two objectives,
  minimizing false positive and minimizing false negative rates

- A positive real values ***k*** is how much more false negative irritates us than false positives
- When ***k***=1, the Neyman-Pearson error is the **error rate**

- A common and flexible measure of error

Commonly used loss functions for Classification

Linear (Boolean) classifier

- You train a **linear model** of the form

$$f(x) \;=\; w_0 + w_1 h_2(x) + w_2 h_x(x) + \cdots$$

- Prediction is

$$\hat{y} \;=\; \mathrm{sign}\big(\, f(x) \,\big)$$

- Ideally, we would like to find the weights ***w***, that minimizes **error rate** or more generally **Neyman-Pearson error**

$$\frac{\kappa C_{fn} + C_{fp}}{N} \;=$$

$$\frac{1}{N}\sum_{i=1}^{N}\Big\{\kappa \mathbf{I}(\mathrm{sign}(x_i) = -1 \text{ and } y_i = 1) + \mathbf{I}(\mathrm{sign}(x_i) = -1 \text{ and } y_i = 1)\Big\}$$

Notations

- So far we used the notation

$$f(x) \;=\; w_0 + w_1 h_2(x) + w_2 h_x(x) + \cdots$$

  for the linear model, and

$$\hat{y} \;=\; \mathrm{sign}\big(\, f(x) \,\big)$$

  for the (discrete) prediction

- From now on, we will also use $\hat{y}$ to denote the continuous valued model:

$$\hat{y} \;=\; f(x) \;=\; w_0 + w_1 h_2(x) + w_2 h_x(x) + \cdots$$

  to not introduce additional notation
- It should be clear from context which one we mean by $\hat{y}$

Training a linear classifier

- Given a linear model

$$\hat{y} \;=\; f(x) \;=\; w_0 + w_1 h_1(x) + w_2 h_2(x) + \cdots$$

- Neayman-Pearson error cannot be directly optimized (more on this later)

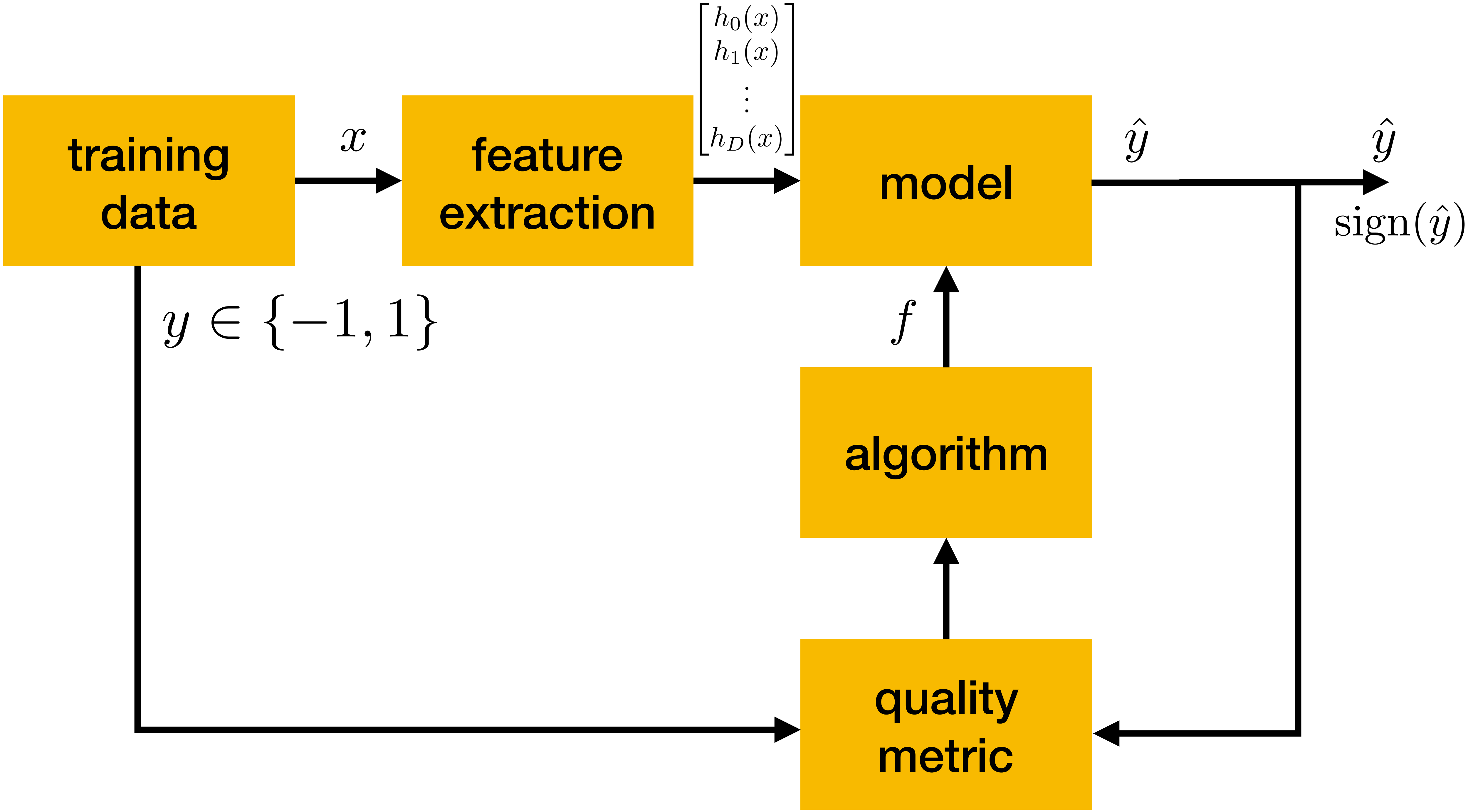
- Instead, in training classifiers, we minimize a loss of the form

$$\mathcal{L}(w) \;=\; \frac{1}{N}\sum_{i=1}^{N} \ell(\underbrace{\hat{y}_i}_{w^T x}, y_i)$$

- And find parameters ***w,*** that minimize a particular choice of loss function

$$\ell(\hat{y}, y)$$

- The choice depends on the application
- One can use regularization: $\text{minimize}_w \;\; \mathcal{L}(w) + \lambda\, r(w)$



- Recipe for training classifiers
  - 1. Train a continuous valued model, as if regression but with special choices of the loss
  - 2. For prediction take sign(f(x))
  - 3. The score f(x) tells us how confident we are in the prediction

Loss function for Boolean classification

- We need to design loss function $\ell(\hat{y}, y)$

- Note that
  - prediction $\hat{y} = w^T x$ can take any values
  - but y can only take $+1$ or -1

- So in order to specify $\ell(\hat{y}, y)$,
  we only need to give two functions (of scalar $\hat{y}$)
  - $\ell(\hat{y}, -1)$ is how much $\hat{y}$ irritates us when $y = -1$
  - $\ell(\hat{y}, 1)$ is how much $\hat{y}$ irritates us when $y = 1$

- typically, one chooses those two functions to be symmetric, but appropriately scaled to reflect that false negatives irritates us factor ***k*** more than false positives:

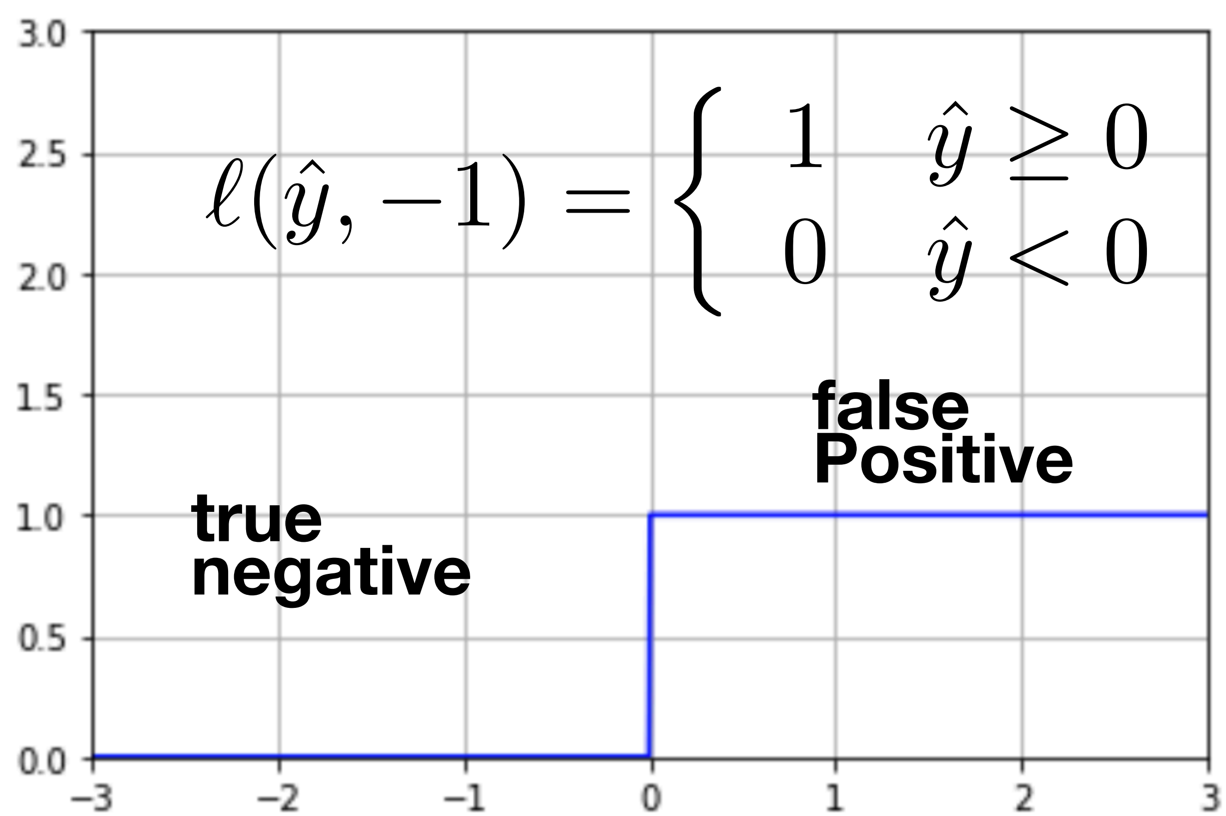
$$\ell(\hat{y}, 1) = \kappa\, \ell(-\hat{y}, -1)$$

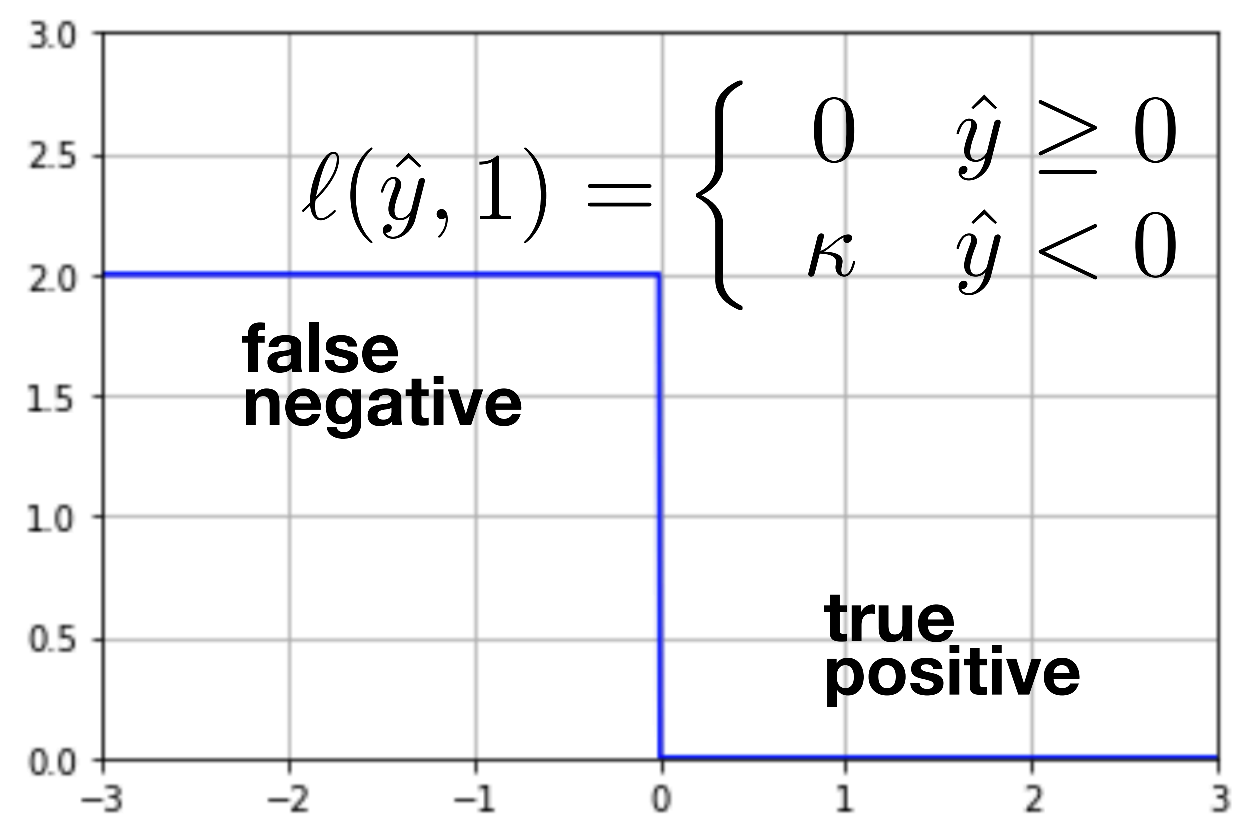
Neyman-Pearson loss

- Neayman-Pearson loss is

$$\ell(\hat{y}, -1) = \begin{cases} 1 & \hat{y} \geq 0 \\ 0 & \hat{y} < 0 \end{cases} \qquad \ell(\hat{y}, 1) = \begin{cases} 0 & \hat{y} \geq 0 \\ \kappa & \hat{y} < 0 \end{cases}$$

- Neayman-Pearson loss computed on the training data is (training) Neayman-Pearson error





Problem with Neyman-Pearson loss

- Neyman-Pearson loss is not differentiable, or even continuous (And certainly not convex)

- Its gradient is zero or does not exist

- Gradient based optimizer does not know how to improve the model

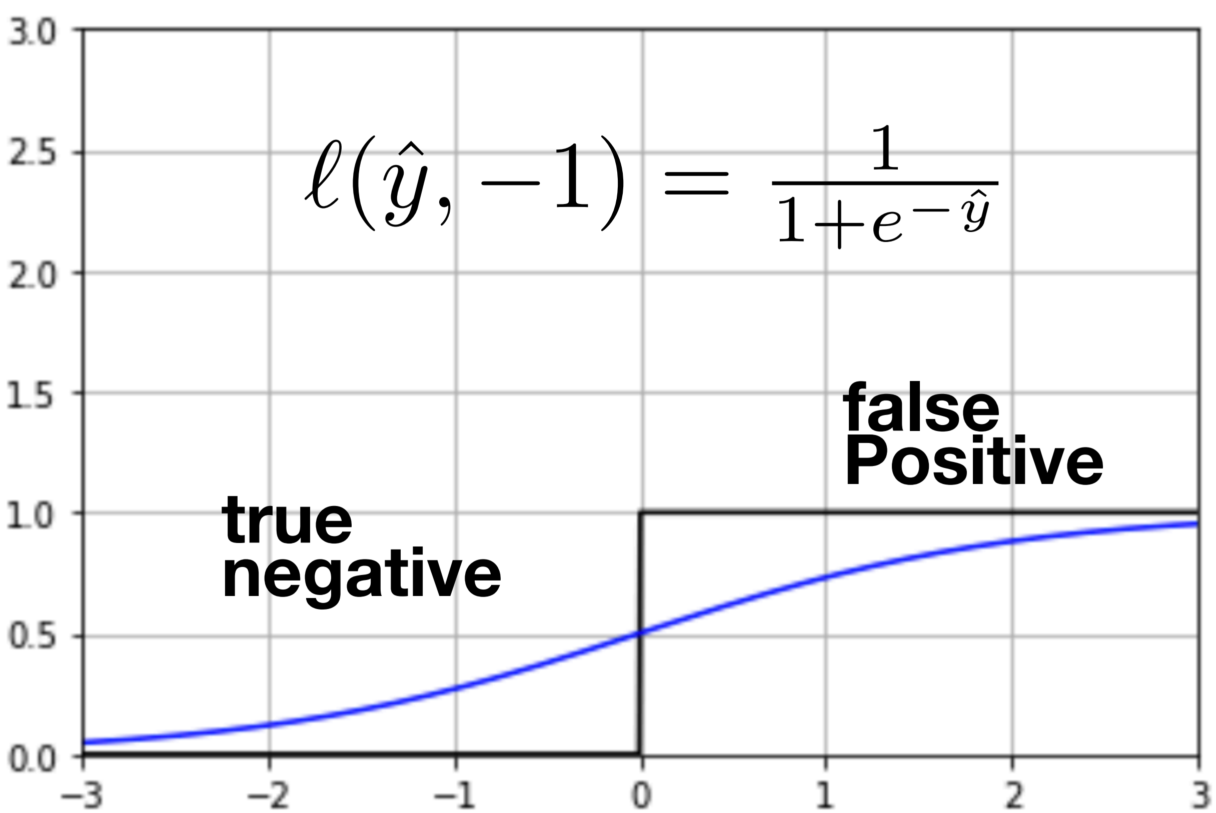
Ideas of proxy loss

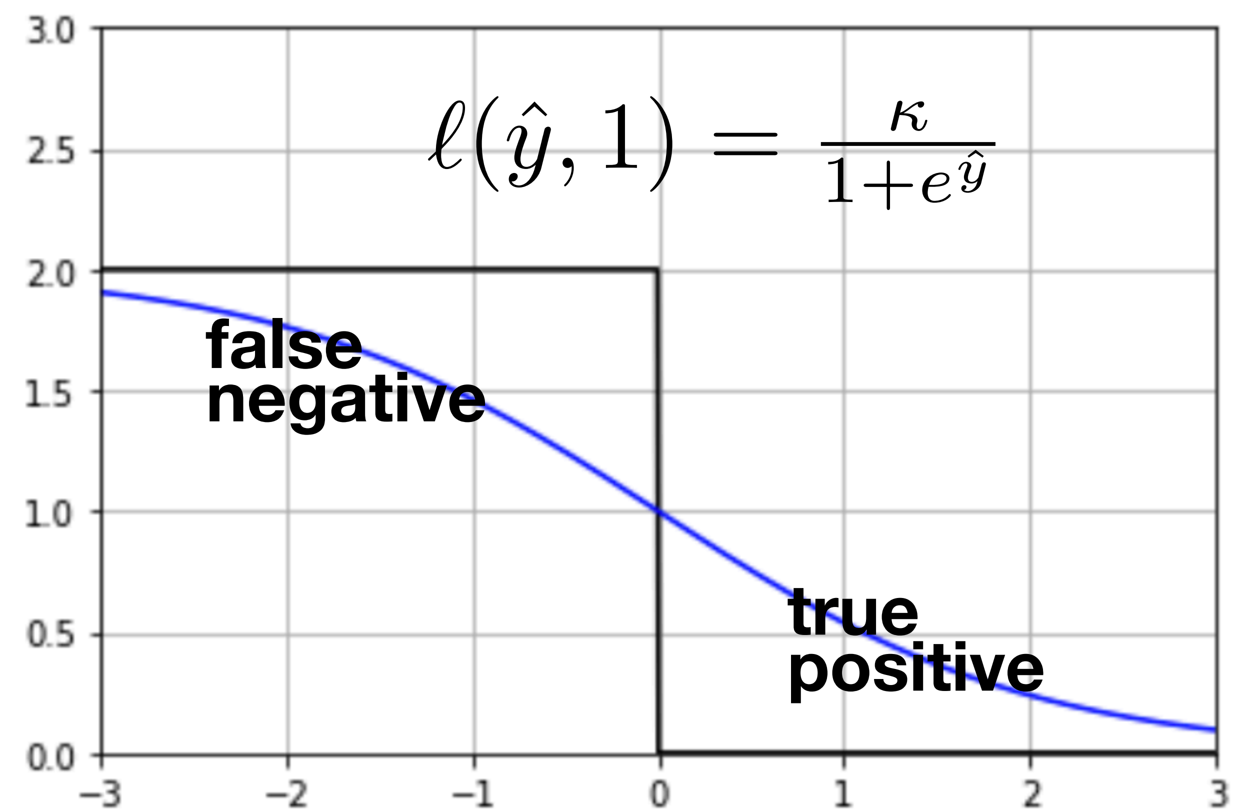
- We get better results using proxy losses that
  - approximate, or captures the flavor of, the Neyman-Pearson loss
  - Is more easily optimized (e.g. convex or non-zero derivatives)

- concretely, we want **proxy loss function**
  - with $\ell(\hat{y}, -1)$ small when $\hat{y} < 0$ and larger when $\hat{y} > 0$
  - with $\ell(\hat{y}, 1)$ small when $\hat{y} > 0$ and larger when $\hat{y} < 0$
  - Which has other nice characteristics, e.g., differentiable or convex

Sigmoid loss

- Differentiable approximation of Neyman-Pearson loss
- But not convex in $\hat{y}$
- The two losses sum to one, if ***k***=1
- Softer (or smoothed) version of the N-P loss

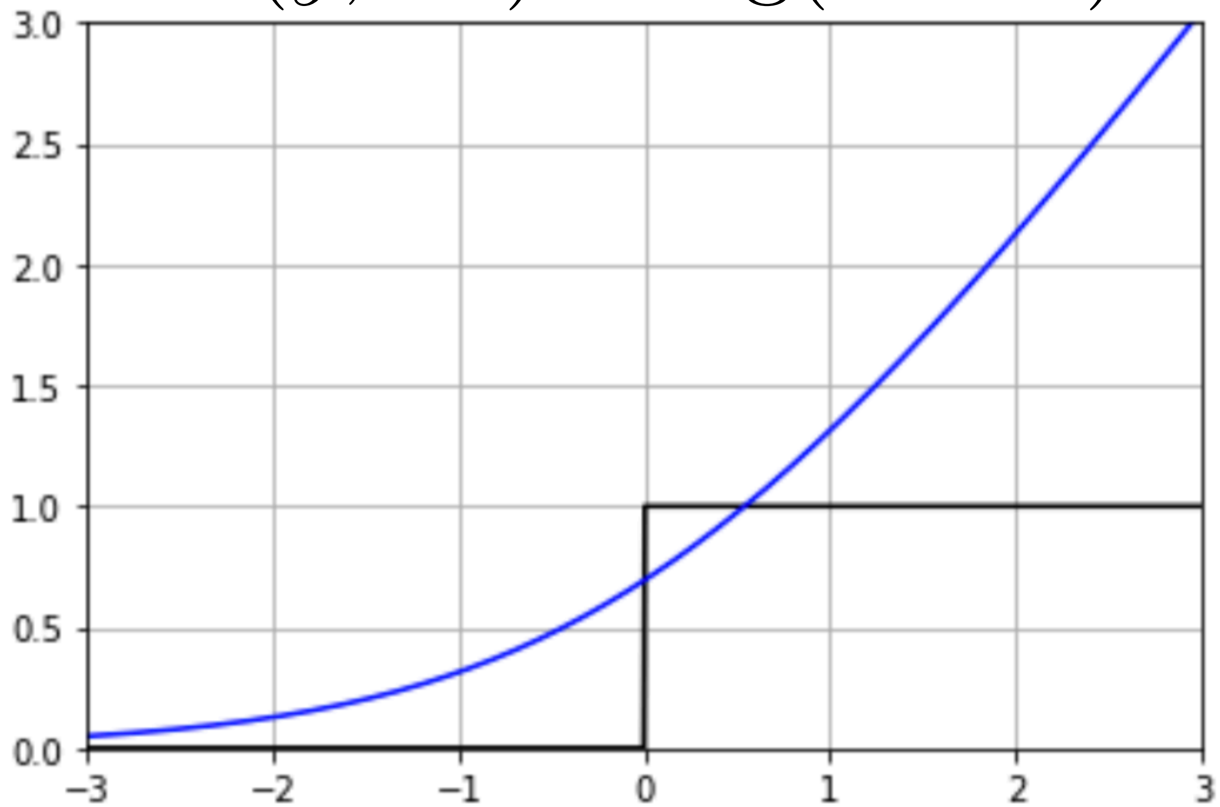




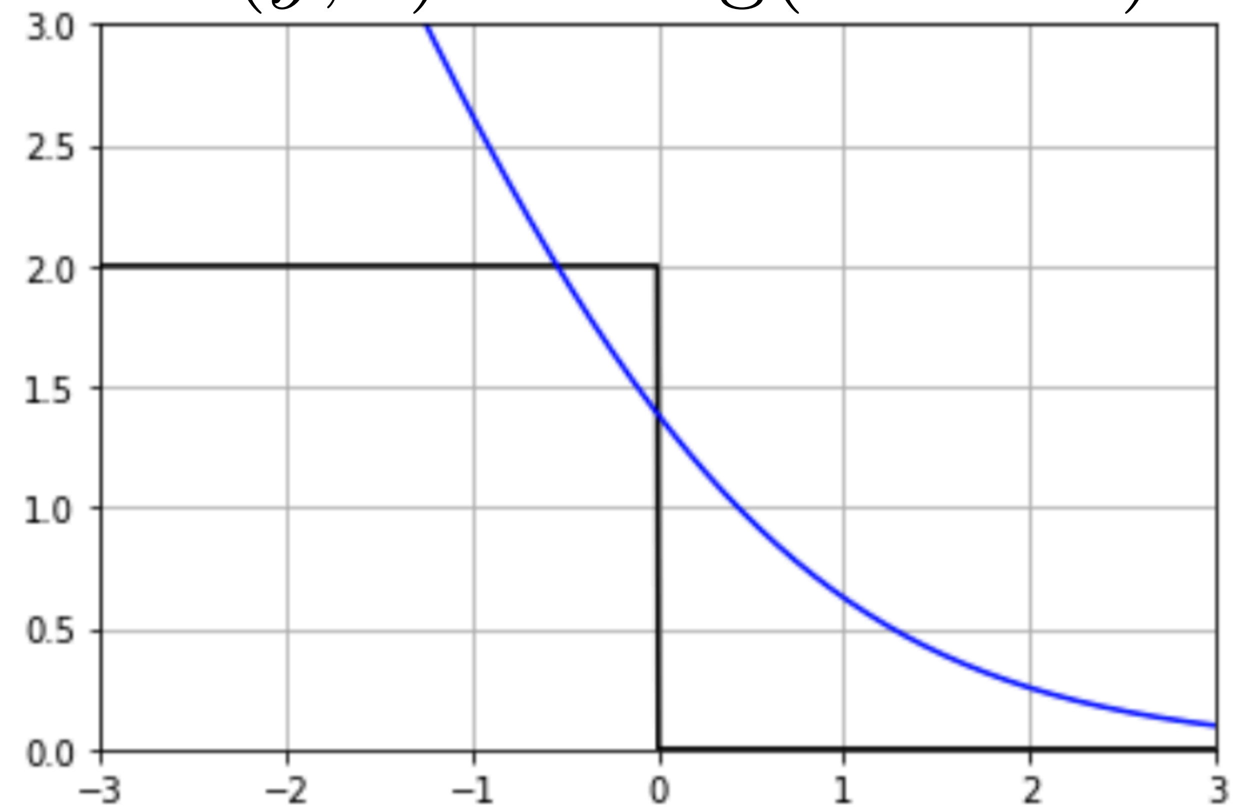
Logistic loss

- Differentiable and convex in $\hat{y}$
- approximation of Neyman-Pearson

$$\ell(\hat{y}, -1) = \log(1 + e^{\hat{y}})$$



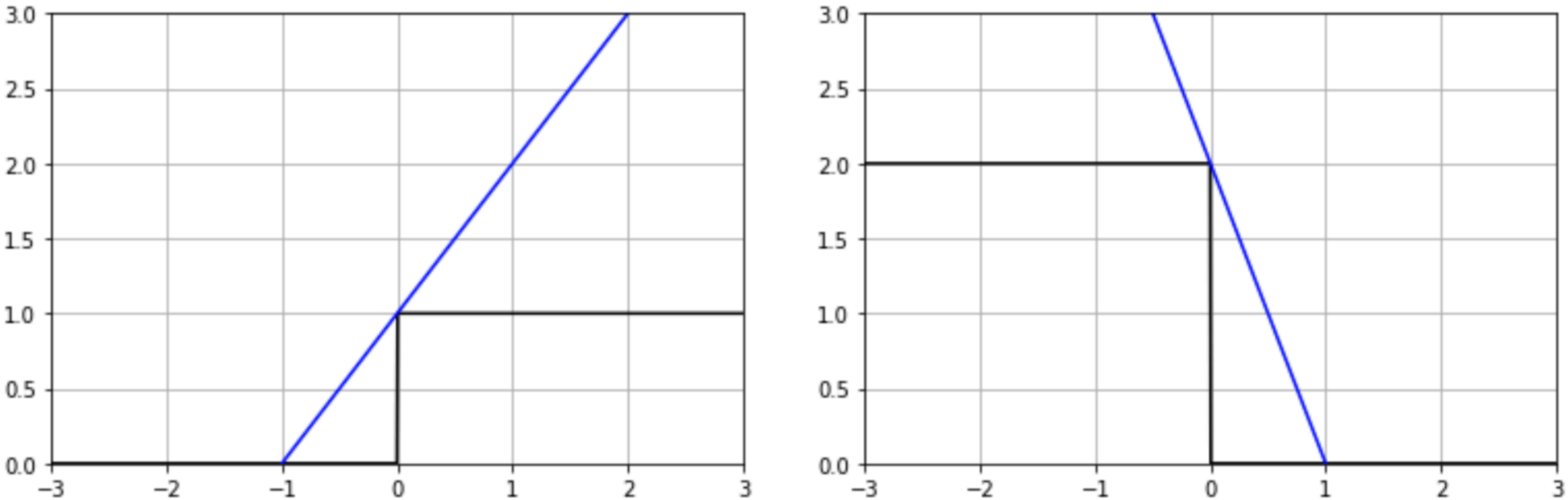
$$\ell(\hat{y}, 1) = \kappa \log(1 + e^{-\hat{y}})$$



Hinge loss

- Non-differentiable but convex approximation of Neyman-Pearson loss

$$\ell(\hat{y}, -1) = [1 + \hat{y}]^+ \qquad \ell(\hat{y}, 1) = \kappa[1 - \hat{y}]^+$$

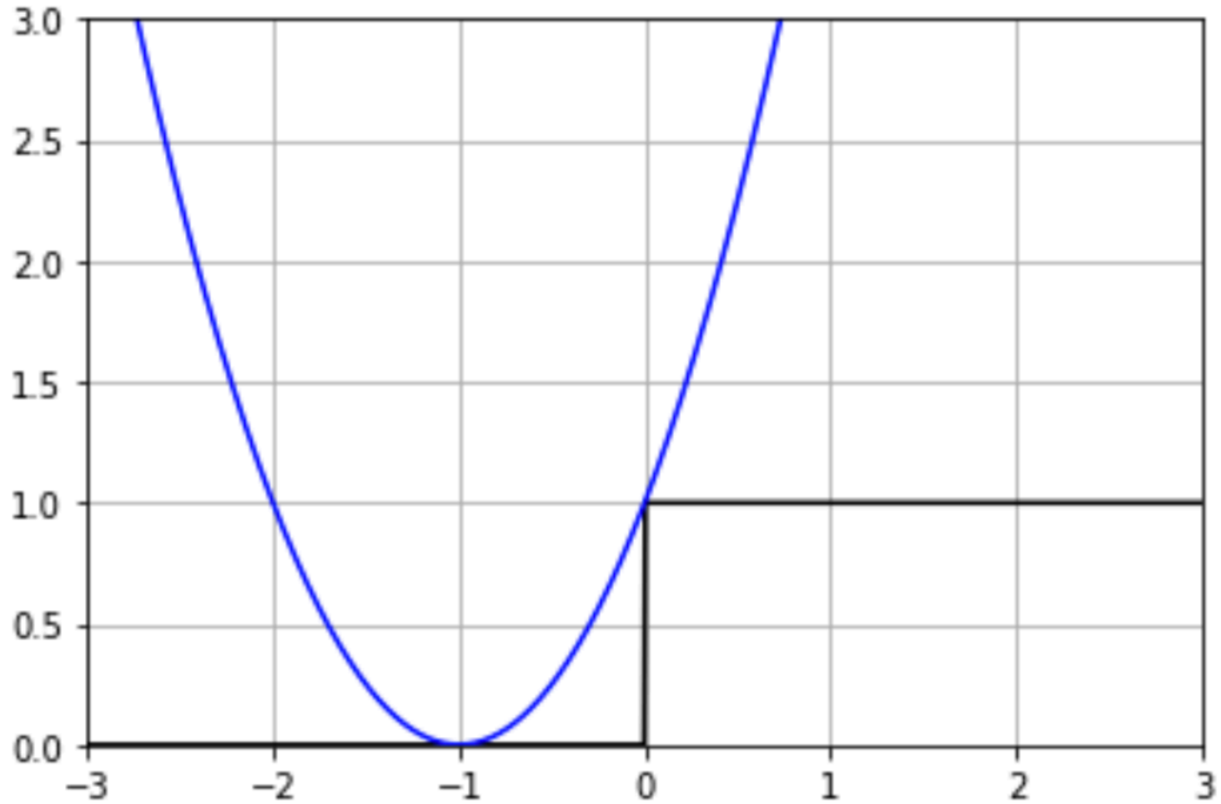


$$\text{where } [x]^+ \ = \ \max\{0, x\}$$

Square loss

- Not only is it convex, square loss is easy to minimize (has a closed form solution)

$$\ell(\hat{y}, -1) = (1 + \hat{y})^2$$



$$\ell(\hat{y}, 1) = \kappa(1 - \hat{y})^2$$

Commonly used Boolean classifiers

Squared loss classifier

- Uses sum of squares loss (a.k.a. L2 loss, Mean Squared Error (MSE), Residual Sum of Squares (RSS))

$$\text{minimize}_w \ \mathcal{L}(w) \ = \ \frac{1}{N}\Big(\sum_{i:y_i=-1} (1+\hat{y}_i)^2 + \kappa \sum_{i:y_i=1} (1-\hat{y}_i)^2 \Big)$$

together with a choice of your regularizer

- This is particularly easy to optimize, if the regularizer is also L2 regularizer

Logistic regression

- Uses logistic loss

$$\text{minimize}_w \ \mathcal{L}(w) \ = \ \frac{1}{N}\Big(\sum_{i:y_i=-1} \log(1+e^{\hat{y}_i}) + \kappa \sum_{i:y_i=1} \log(1+e^{-\hat{y}_i}) \Big)$$

  with a choice of a regularizer

- Is a convex optimization if the regularizer is convex, and the minimizer can be found efficiently

Support vector machine (SVM)

- Uses hinge loss

$$\text{minimize}_w \; \mathcal{L}(w) \;=\; \frac{1}{N}\Big(\sum_{i:y_i=-1} [1+\hat{y}_i]^+ + \kappa \sum_{i:y_i=1} [1-\hat{y}_i]^+ \Big)$$

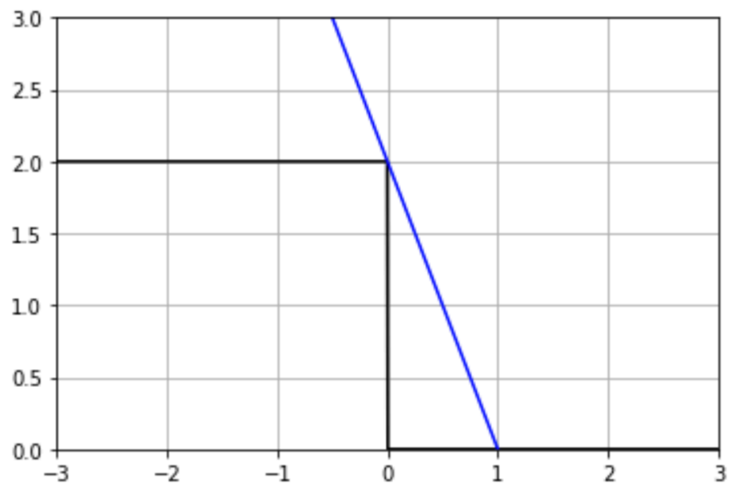
with sum of squares regularizer

$$\text{where } [x]^+ \;=\; \max\{0, x\}$$

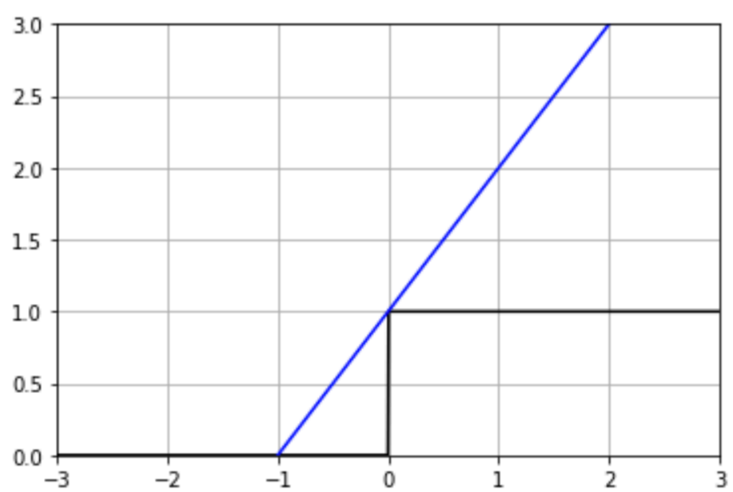
- It is a convex minimization

Support vector machine (SVM)

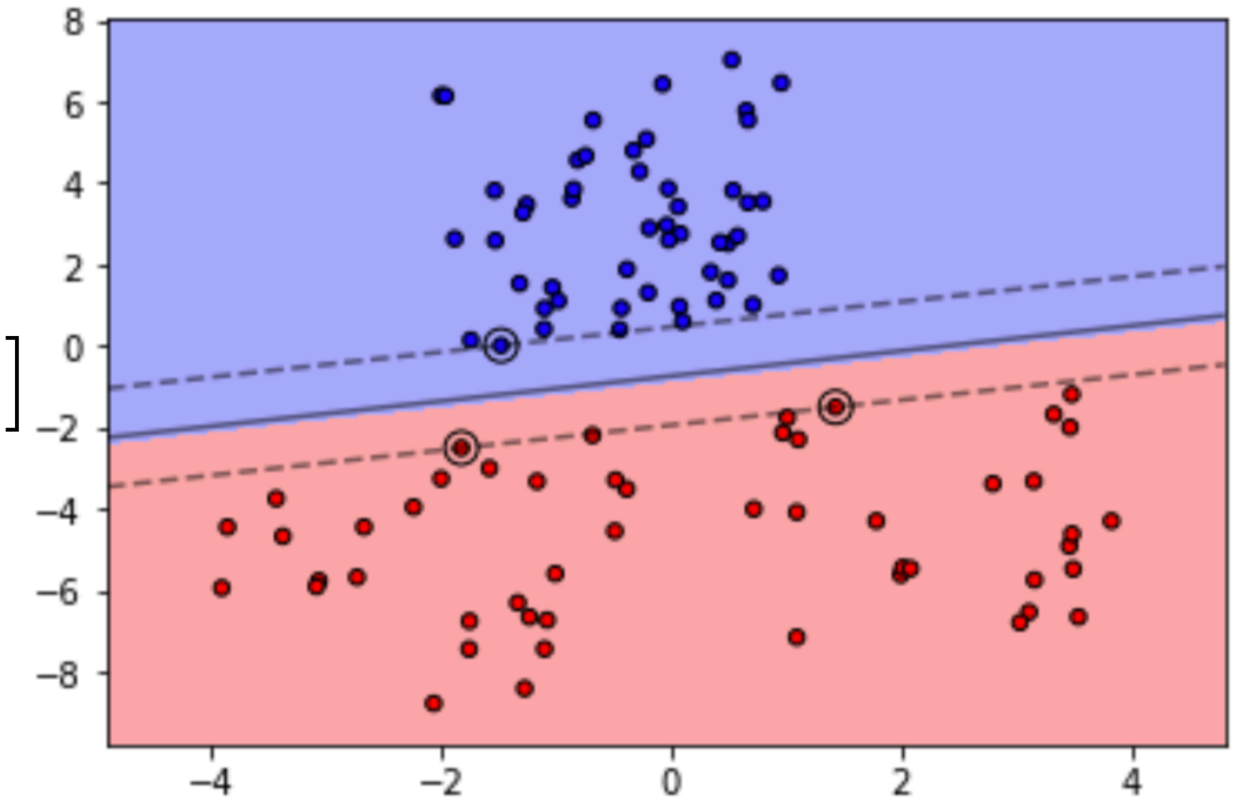
$$\ell(\hat{y}, 1) = \kappa[1 - \hat{y}]^+$$



$$\ell(\hat{y}, -1) = [1 + \hat{y}]^+$$



- Linear model is trained on the hinge loss shown on the left with ***k***=1
- Resulting prediction is shown below



- As we predict with $\mathrm{sign}(\hat{y})$, the decision boundary is at $w^T x = 0$
- black lines show the points where $w^T x = \pm 1$
- What is the training error?

Receiver Operating Characteristic (ROC)

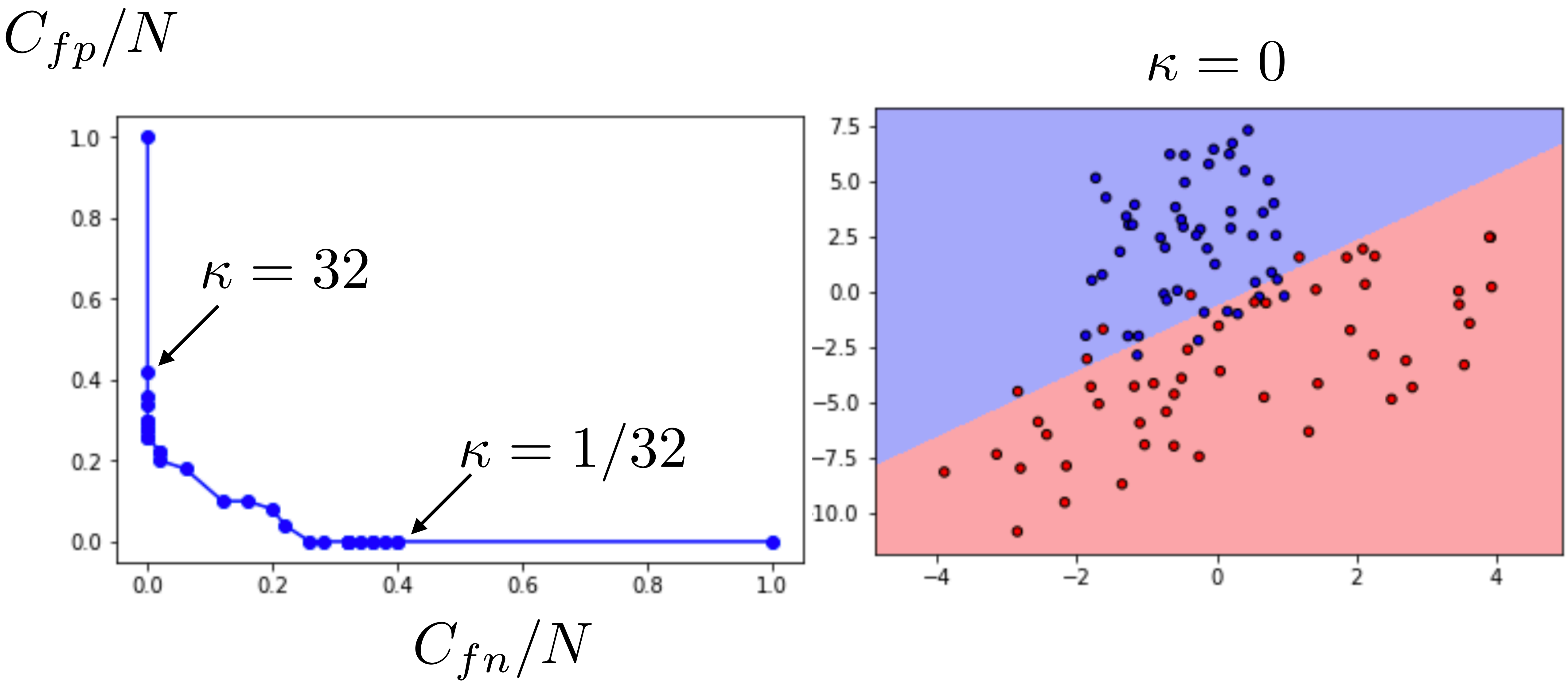
Receiver Operating Characteristic (ROC)

- Always abbreviated as **ROC,** comes from WWII
- Explores the tradeoff between false negative and false positive rates

- Typical recipe for evaluating performance of a classifier
  - 1. Construct a classifier for many values of ***k***
    - For each ***k***, select the regularization hyper-parameter via cross-validation, that minimizes Neyman-Pearson loss on test data set
  - 2. Plot the computed pair (false negative rate, false positive rate) on a 2-D plot.

- Connecting all the dots gives you **ROC curve** (when viewed upside-down)

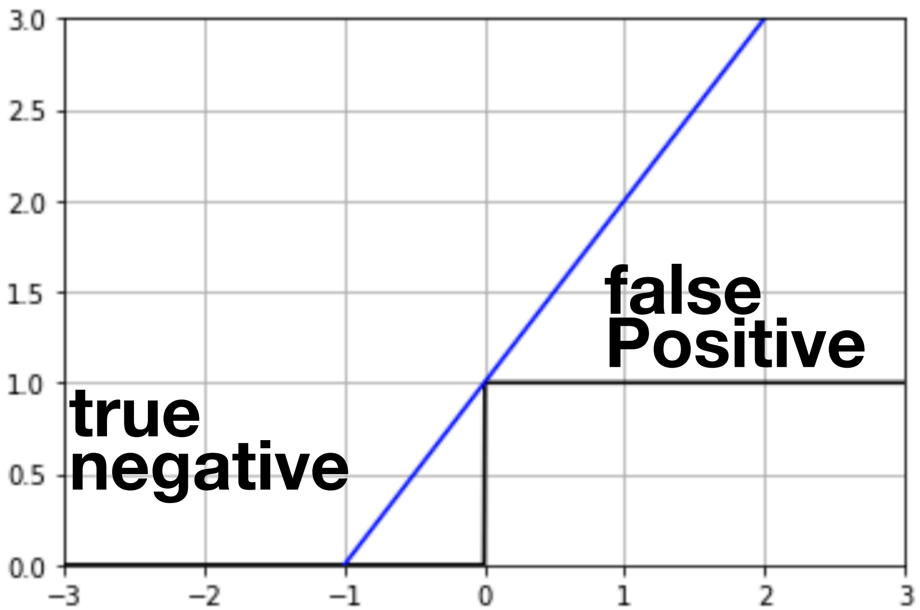
Example: ROC curve

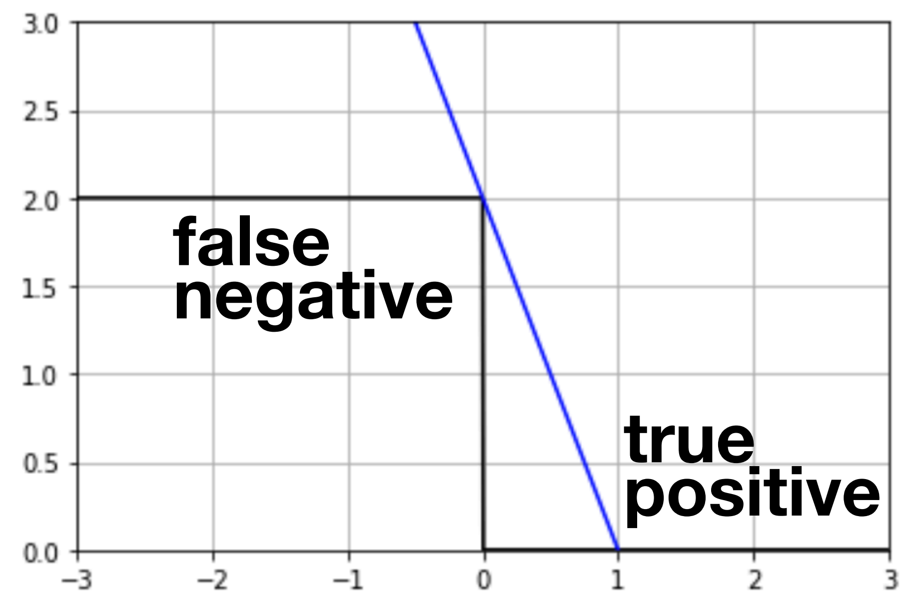


- SVM with various k
- Left hand plot shows training error pairs (C_{fn}/N ,C_{fp}/N)
- Right hand plot shows minimum error classifier (i.e. k=1)

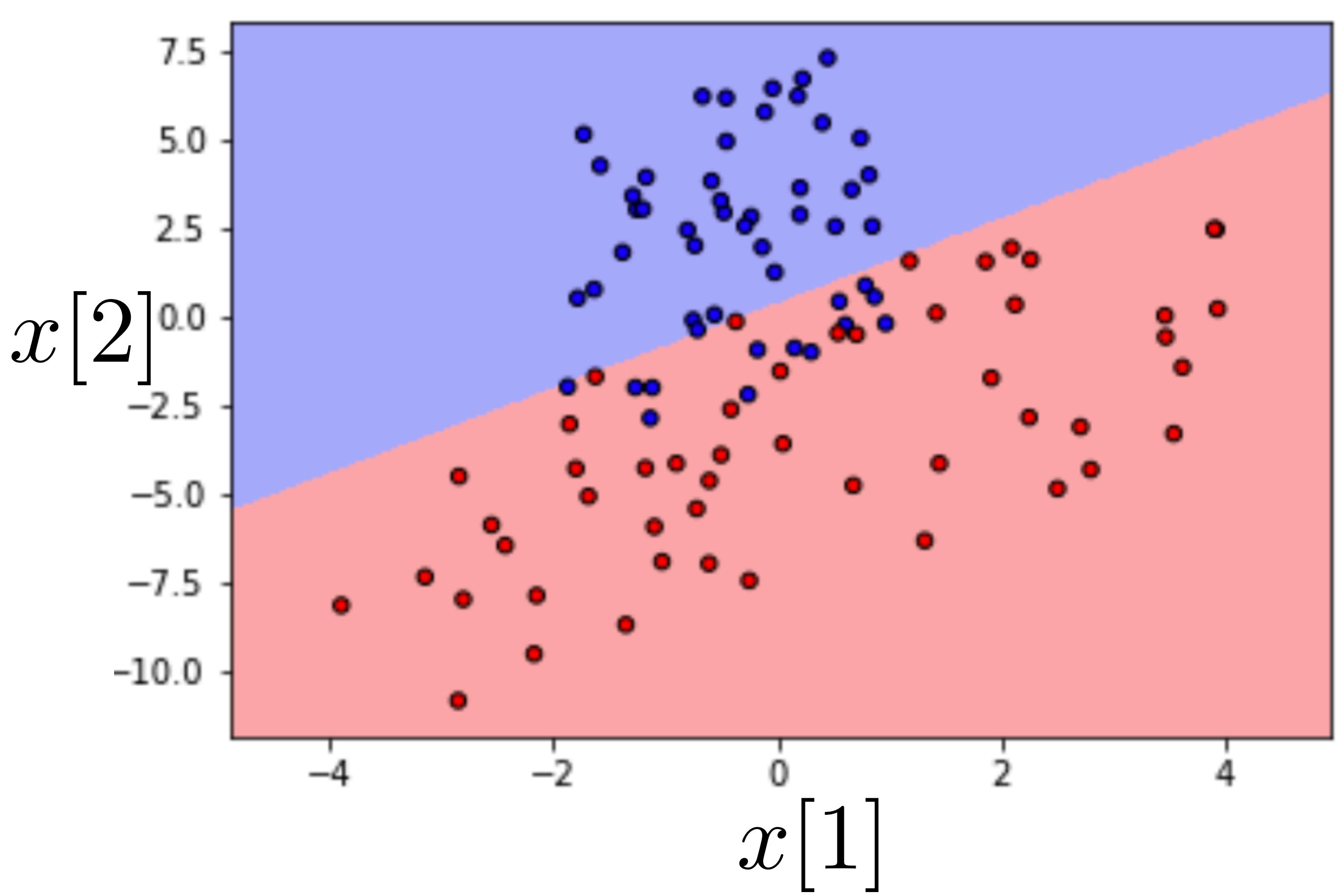
Example

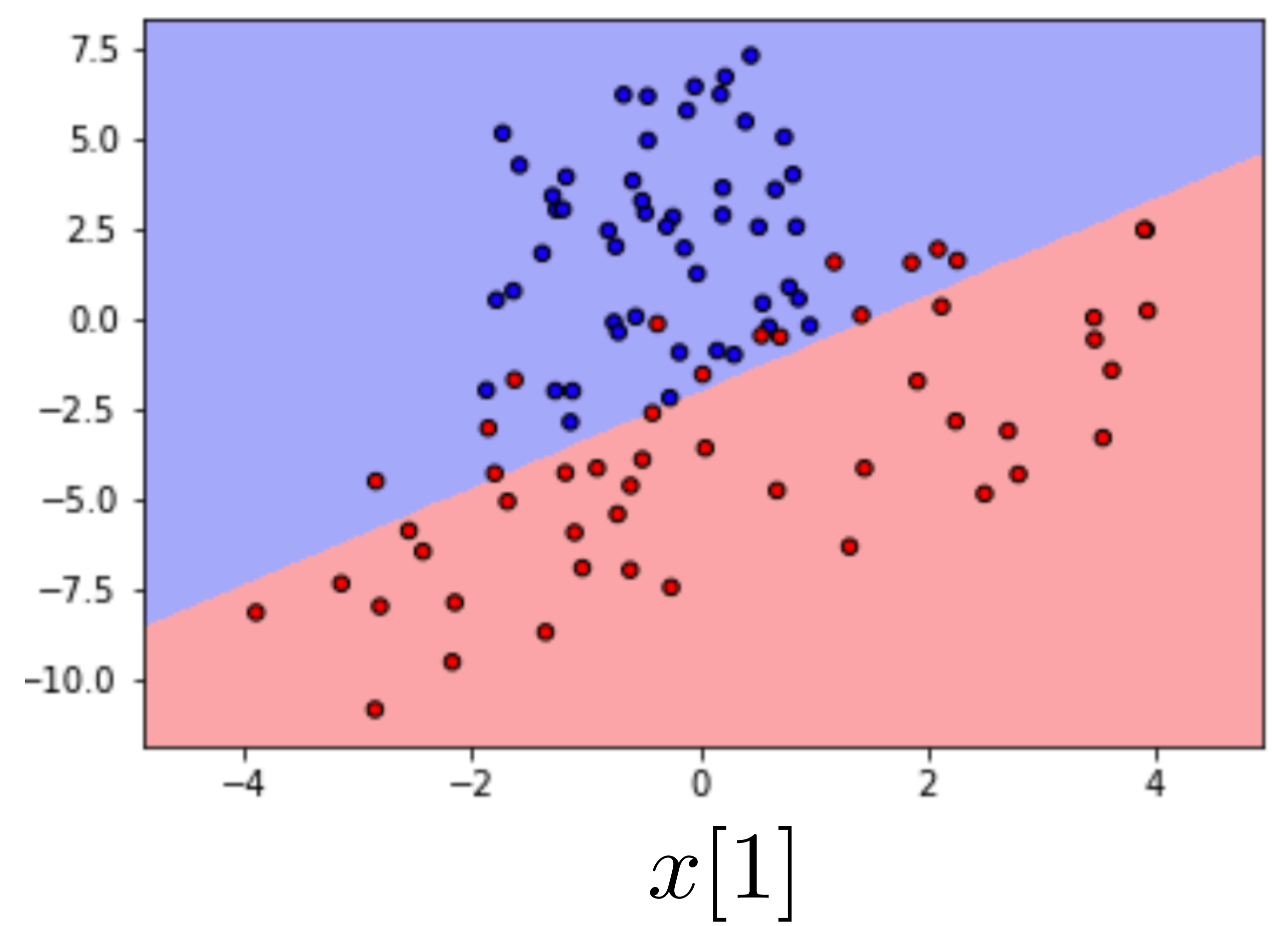
$$\ell(\hat{y}, -1) = [1 + \hat{y}]^+ \qquad \ell(\hat{y}, 1) = \kappa[1 - \hat{y}]^+$$





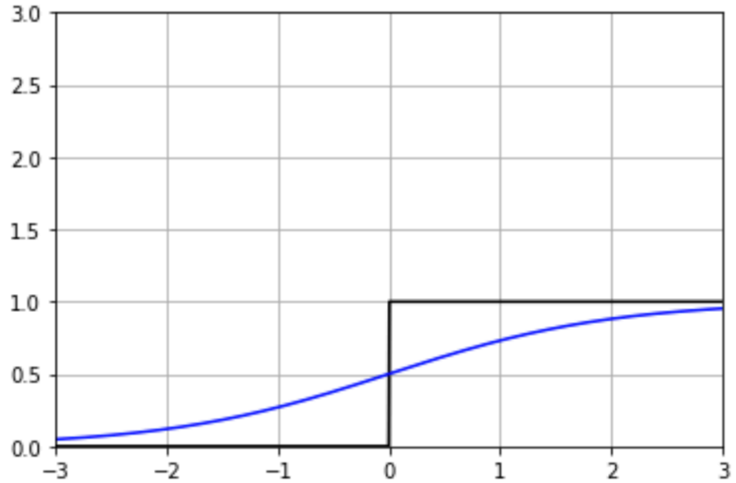
$\kappa = 0.4$ $\qquad$ $\kappa = 4$





Probabilistic interpretation of logistic regression

- When $\kappa = 1$, we get the following losses for each data point x



$$(\underbrace{\frac{1}{1+e^{-w^T x}}}_{\ell(\hat{y},-1)}, \underbrace{\frac{1}{1+e^{w^T x}}}_{\ell(\hat{y},1)})$$

  when using **sigmoid loss** that is trained with a linear model
- They are
  - Non-negative
  - sum to one, and
  - they measure how likely it is that the point ***x*** has label +1 (or -1) respectively
- One can view it as an estimation of the probability

$$(\mathbb{P}(y_i = +1|x), \mathbb{P}(y_i = -1|x))$$

-

Probabilistic interpretation of logistic regression

- Then taking the sign of the linear predictor to make final decision is simply taking a label that is more likely:

$$\hat{\hat{y}} = \mathrm{sign}(w^T x) \quad \Longleftrightarrow \quad \hat{\hat{y}} = \begin{cases} +1 & \frac{1}{1+e^{-w^T x}} > \frac{1}{1+e^{w^T x}} \\ -1 & \text{otherwise} \end{cases}$$

- and **logistic regression** can be interpreted as **Maximum Likelihood Estimator** under the probabilistic model with sigmoid function:

- $$(\underbrace{\frac{1}{1+e^{-w^T x}}}_{\mathbb{P}(y_i=+1|x_i)}, \underbrace{\frac{1}{1+e^{w^T x}}}_{\mathbb{P}(y_i=-1|x_i)})$$

Maximum Likelihood Estimator (MLE)

- model: $$(\underbrace{\frac{1}{1+e^{-w^Tx}}}_{\mathbb{P}(y_i=+1|x_i)}, \underbrace{\frac{1}{1+e^{w^Tx}}}_{\mathbb{P}(y_i=-1|x_i)})$$
- log-likelihood on a data point (xi,yi):

$$\text{log-likelihood} = \log\Big(\mathbb{P}(y_i|x_i)\Big) = \begin{cases} \log\Big(\frac{1}{1+e^{-w^Tx_i}}\Big) & \text{if } y_i = +1 \\ \log\Big(\frac{1}{1+e^{w^Tx_i}}\Big) & \text{if } y_i = -1 \end{cases}$$

- Maximum Likelihood Estimator is the one that maximizes the sum of all likelihoods on training data points

$$\text{maximize}_w \textstyle\sum_{i:y_i=-1} \log\Big(\frac{1}{1+e^{\hat{y}_i}}\Big) + \sum_{i:y_i=1} \log\Big(\frac{1}{1+e^{-\hat{y}_i}}\Big)$$

- Notice that this is exactly the **logistic regression** without any regularizers and with ***k***=1

$$\text{minimize}_w\ \mathcal{L}(w) \ = \ \frac{1}{N}\Big(\sum_{i:y_i=-1} \log(1+e^{\hat{y}_i}) + \kappa \sum_{i:y_i=1} \log(1+e^{-\hat{y}_i})\Big)$$