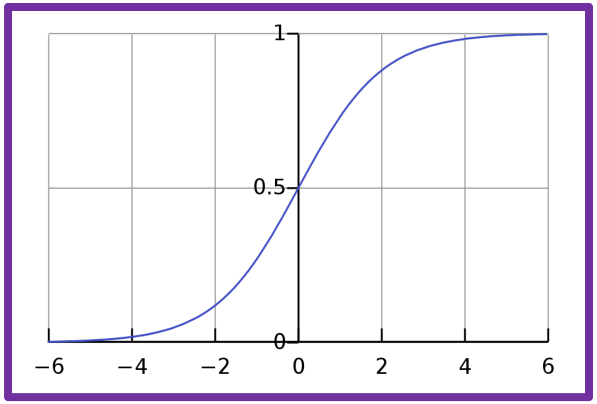
Binary Classification



Many slides attributable to:
Erik Sudderth (UCI)
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Prof. Mike Hughes

Today's objectives (day 07) Binary Classification Basics

- 3 steps of a classification task
  - Prediction
    - Predicting probabilities of each binary class
    - Making hard binary decisions
  - Training
  - Evaluation (much more in next class)

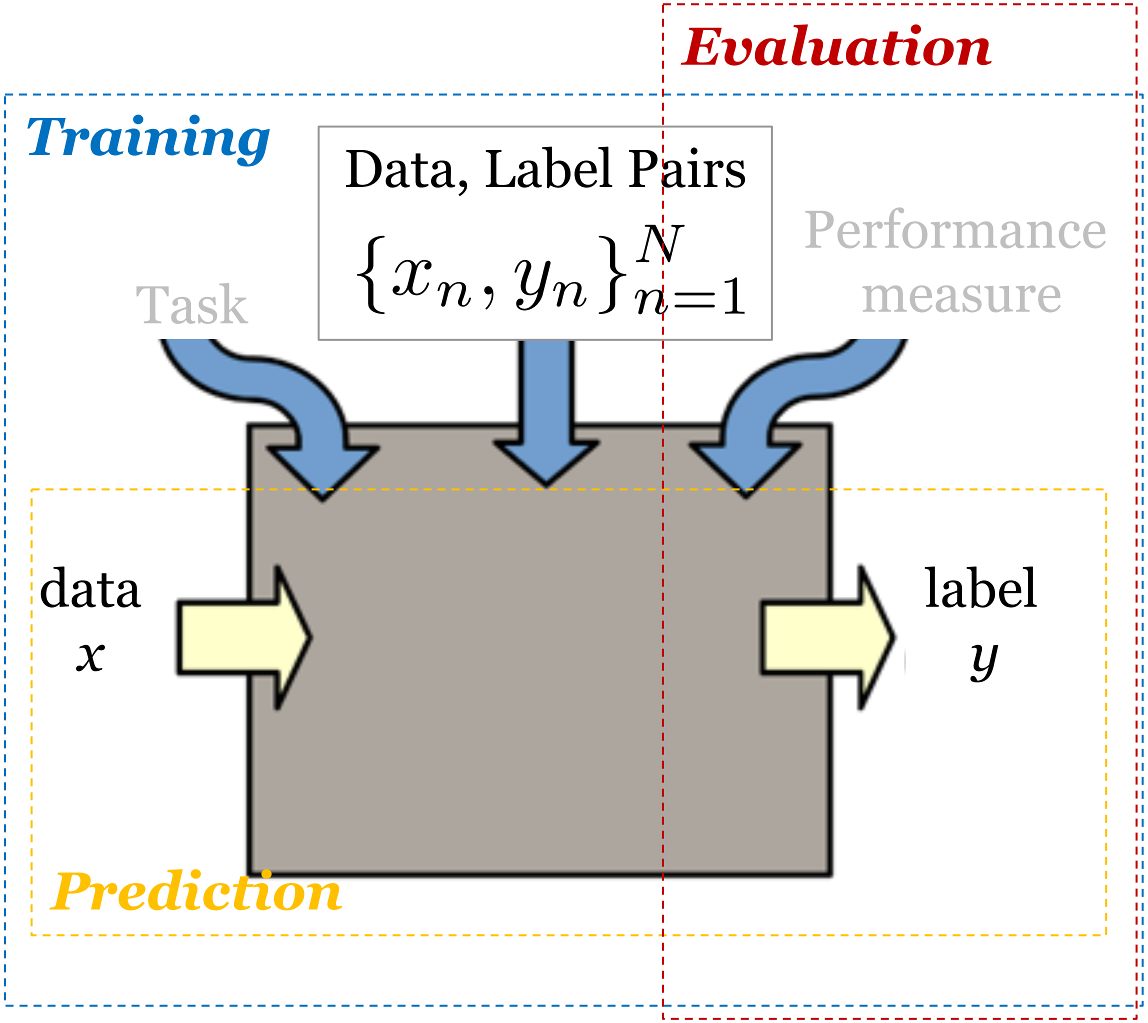
- A "taste" of 2 Methods
  - Logistic Regression
  - K-Nearest Neighbors

What will we learn?

Supervised Learning

Unsupervised Learning

Reinforcement Learning



Before: Regression

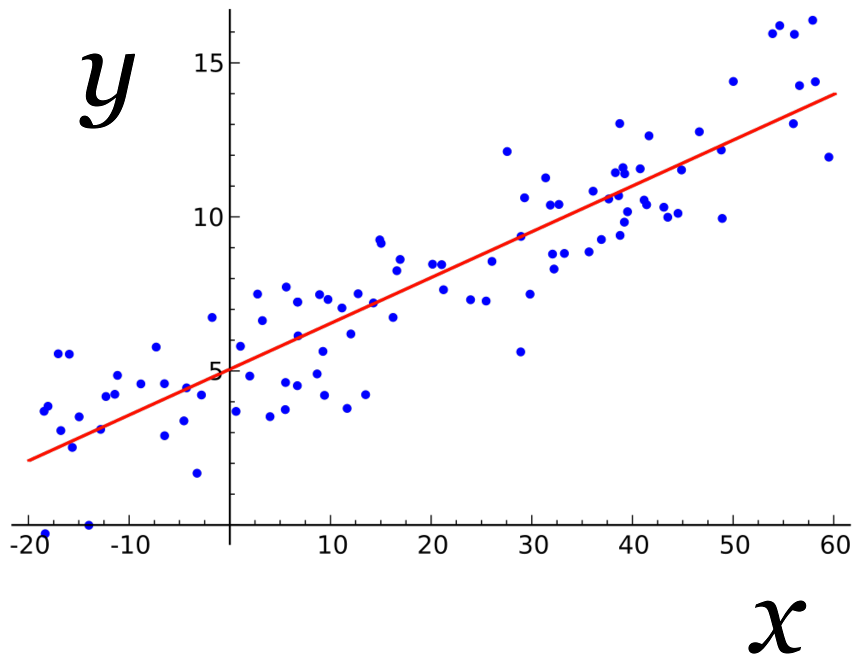
Supervised Learning

regression

Unsupervised Learning

Reinforcement Learning

y is a numeric variable
e.g. sales in $$



Task: Binary Classification

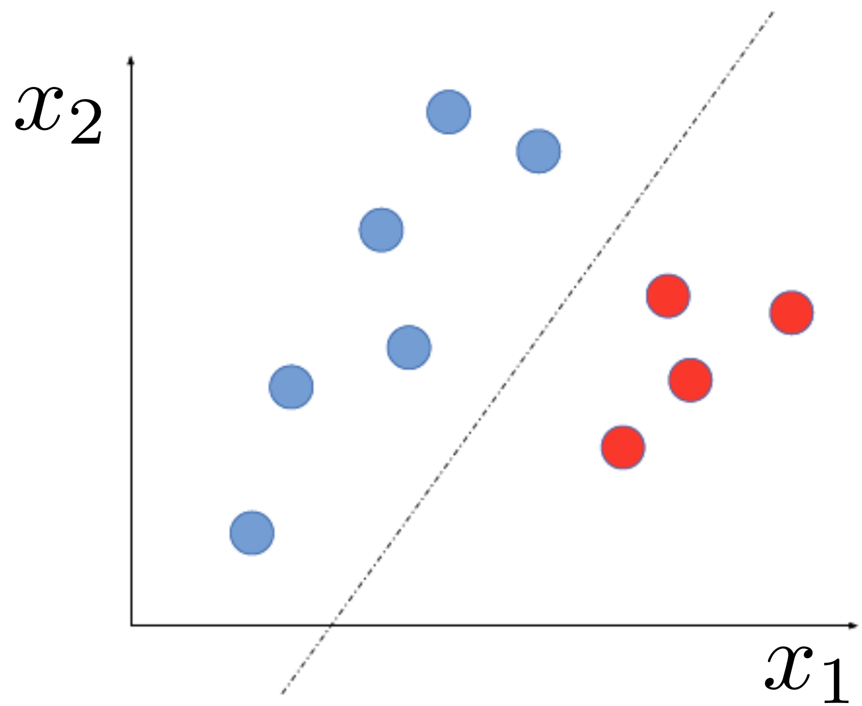
Supervised Learning

binary classification

Unsupervised Learning

Reinforcement Learning

y is a binary variable (red or blue)



Example: Hotdog or Not

https://www.theverge.com/tldr/2017/5/14/15639784/hbo-silicon-valley-not-hotdog-app-download

Task: Multi-class Classification

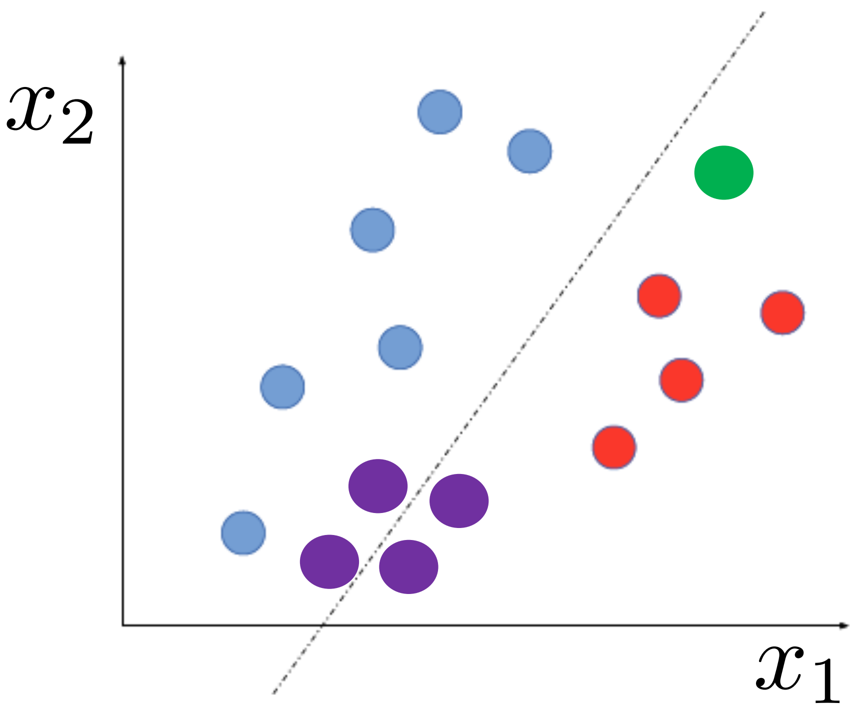
Supervised Learning

multi-class classification

Unsupervised Learning

Reinforcement Learning

y is a discrete variable
(red or blue or green or purple)



Binary Prediction Step

Goal: Predict label (0 or 1) given features x

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

  "features"
  "covariates"
  "predictors"
  "attributes"

  Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $y_i \in \{0, 1\}$

  "responses"
  "labels"

  Binary label (0 or 1)

Binary Prediction Step

```
>>> # Given: pretrained binary classifier model
>>> # Given: 2D array of features x_NF

>>> x_NF.shape
(N, F)

>>> yhat_N = model.predict(x_NF)
>>> yhat_N[:5] # peek at predictions
[0, 0, 1, 0, 1]

>>> yhat_N.shape
(N,)
```

Types of binary predictions

TN : true negative
FN : false negative

FP : false positive
TP : true positive

		classifier calls	
		"negative" C=0	"positive" C=1
true outcome	Y=0	TN	FP
	Y=1	FN	TP

Probability Prediction Step

Goal: Predict probability of event y=1 given features x

- Input: $x_i \triangleq \left[x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF} \right]$

"features"
"covariates"
"predictors"
"attributes"

Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $\hat{p}_i$

"probabilities"

Probability between 0 and 1
e.g. 0.001, 0.513, 0.987

Probability Prediction Step

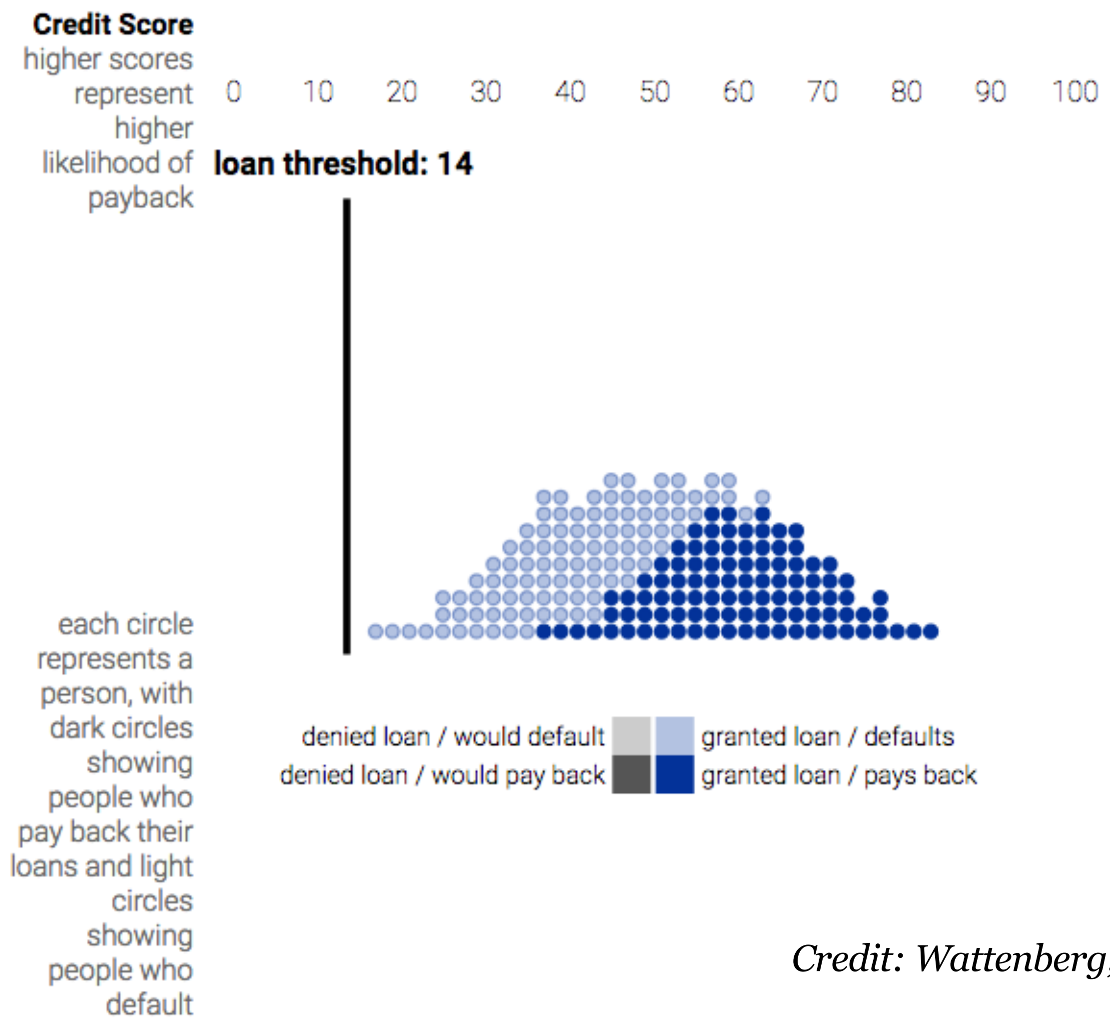
```
>>> # Given: pretrained regression object model
>>> # Given: 2D array of features x_NF

>>> x_NF.shape
(N, F)
```

```
>>> yproba_N2 = model.predict_proba(x_NF)
>>> yproba_N2.shape
(N, 2)
```

Column index 1 gives probability of positive label given input features $p(Y = 1 \mid X)$

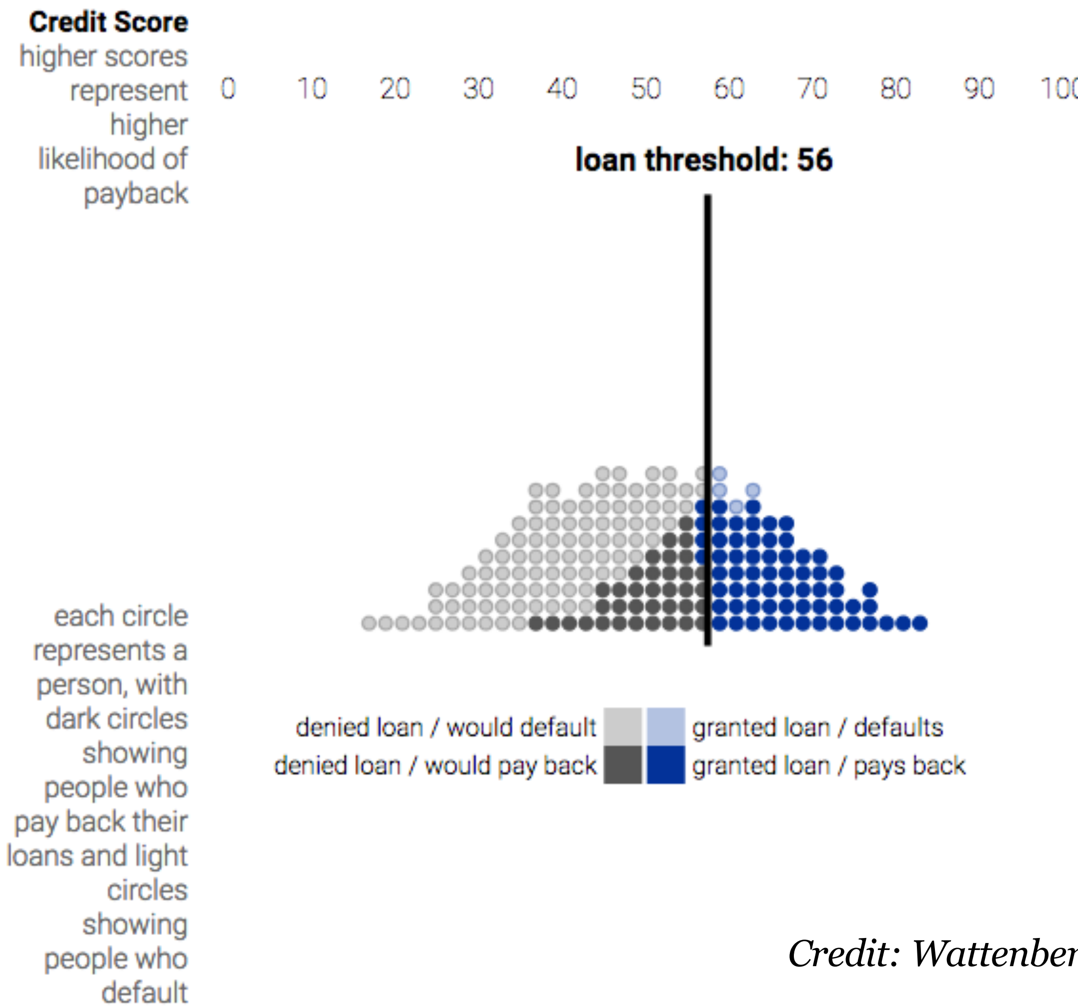
```
>>> yproba_N2[:, 1]
[0.003, 0.358, 0.987, 0.111, 0.656]
```

Thresholding to get Binary Decisions



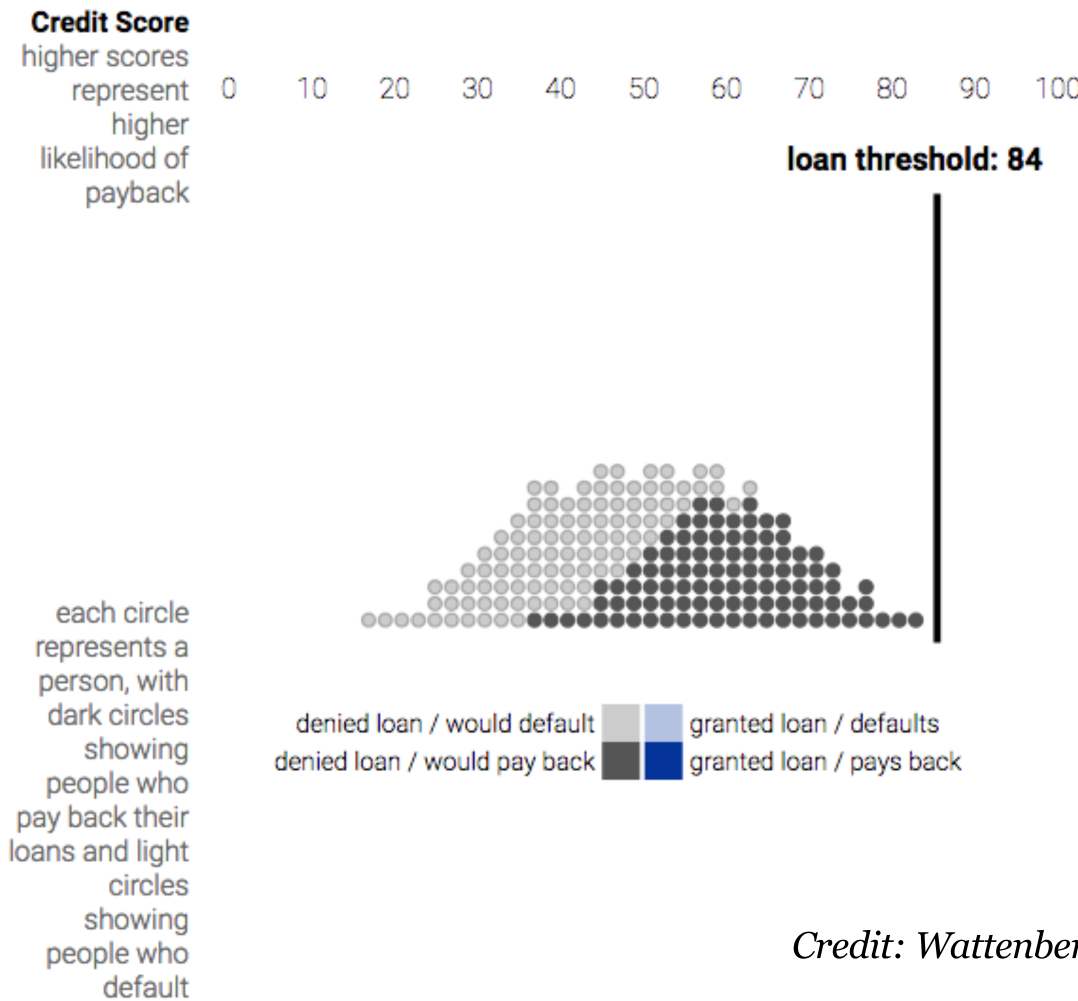
Credit: Wattenberg, Viégas, Hardt

Thresholding to get Binary Decisions



Credit: Wattenberg, Viégas, Hardt

Thresholding to get Binary Decisions



Credit: Wattenberg, Viégas, Hardt

Classifier: Training Step

Goal: Given a labeled dataset, learn a **function** that can perform (probabilistic) prediction well

- Input: Pairs of features and labels/responses

$$\{x_n, y_n\}_{n=1}^{N}$$

- Output: $\hat{y}(\cdot) : \mathbb{R}^F \rightarrow \{0, 1\}$

Useful to break into two steps:

1) *Produce real-valued scores OR probabilities in [0, 1]*
2) *Threshold to make binary decisions*

Classifier: Training Step

```
>>> # Given: 2D array of features x_NF
>>> # Given: 1D array of binary labels y_N

>>> y_N.shape
(N,)
>>> x_NF.shape
(N, F)

>>> model = BinaryClassifier()
>>> model.fit(x_NF, y_N)
>>> # Now can call predict or predict_proba
```

Classifier: **Evaluation Step**

Goal: Assess quality of predictions

Many ways in practice:

1) Evaluate probabilities / scores directly

    cross entropy loss (aka log loss), hinge loss, ...

2) Evaluate binary decisions at specific threshold

    accuracy, TPR, TNR, PPV, NPV, etc.

3) Evaluate across range of thresholds

    ROC curve, Precision-Recall curve

Metric: Confusion Matrix

Counting mistakes in binary predictions

#TN : num. true negative
#FN : num. false negative

#TP : num. true positive
#FP : num. false positive

		classifier calls	
		"negative" C=0	"positive" C=1
true outcome	Y=0	#TN	#FP
	Y=1	#FN	#TP

Metric: Accuracy

accuracy = fraction of correct predictions

$$= \frac{TP + TN}{TP + TN + FN + FP}$$

Potential problem:

Suppose your dataset has 1 positive example and 99 negative examples

What is the accuracy of the classifier that always predicts "negative"?

Metric: Accuracy

accuracy = fraction of correct predictions

$$= \frac{TP + TN}{TP + TN + FN + FP}$$

Potential problem:

Suppose your dataset has 1 positive example and 99 negative examples

What is the accuracy of the classifier that always predicts "negative"?

99%!

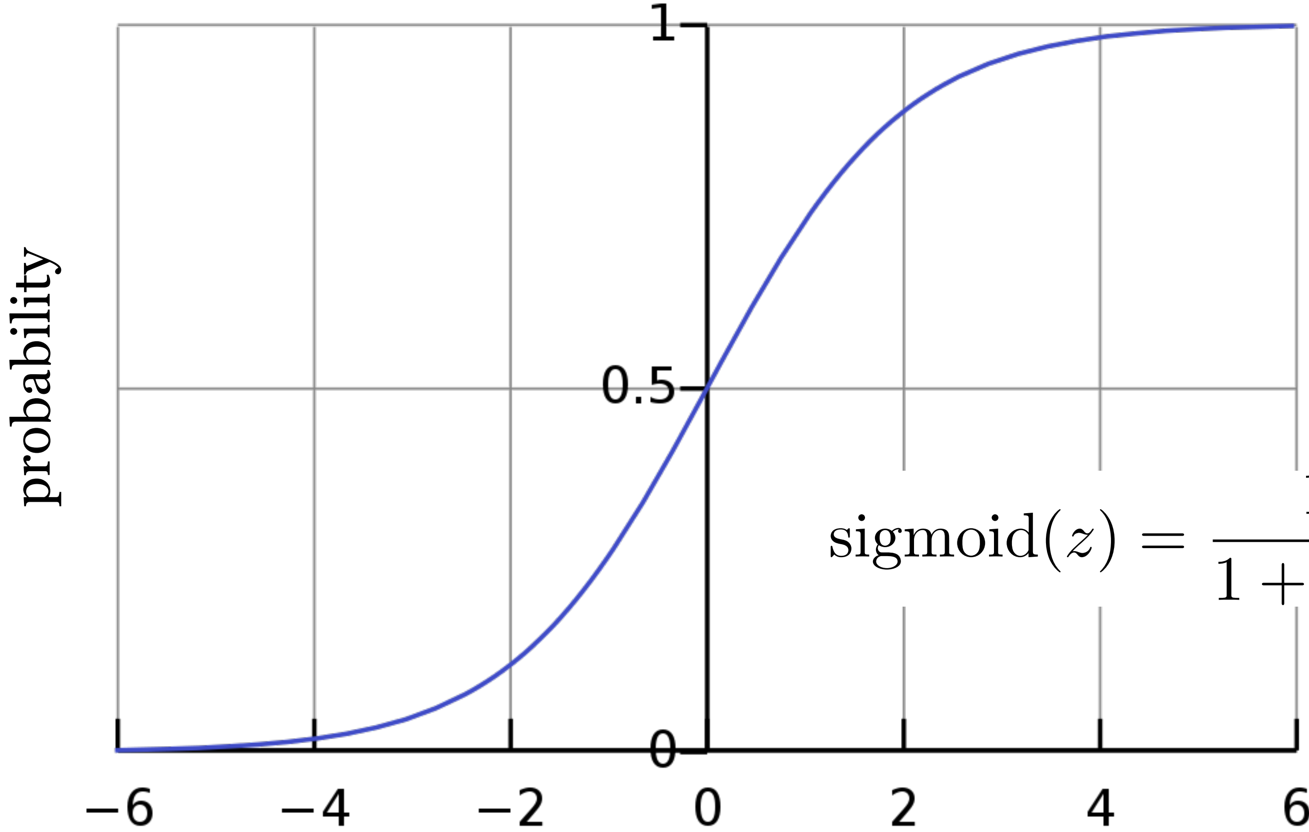
Objectives: Classifier Overview

- 3 steps of a classification task
  - Prediction
    - Making hard binary decisions
    - Predicting class probabilities
  - Training
  - Evaluation

- A "taste" of 2 Methods
  - Logistic Regression
  - K-Nearest Neighbors

Logistic Sigmoid Function

Goal: Transform real values into probabilities



$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

Logistic Regression

Parameters:

weight vector $w = [w_1, w_2, \dots w_f \dots w_F]$

bias scalar b

Prediction:

$$\hat{p}(x_i, w, b) = p(y_i = 1 | x_i) \triangleq \mathrm{sigmoid}\left(\sum_{f=1}^{F} w_f x_{if} + b \right)$$

Training:

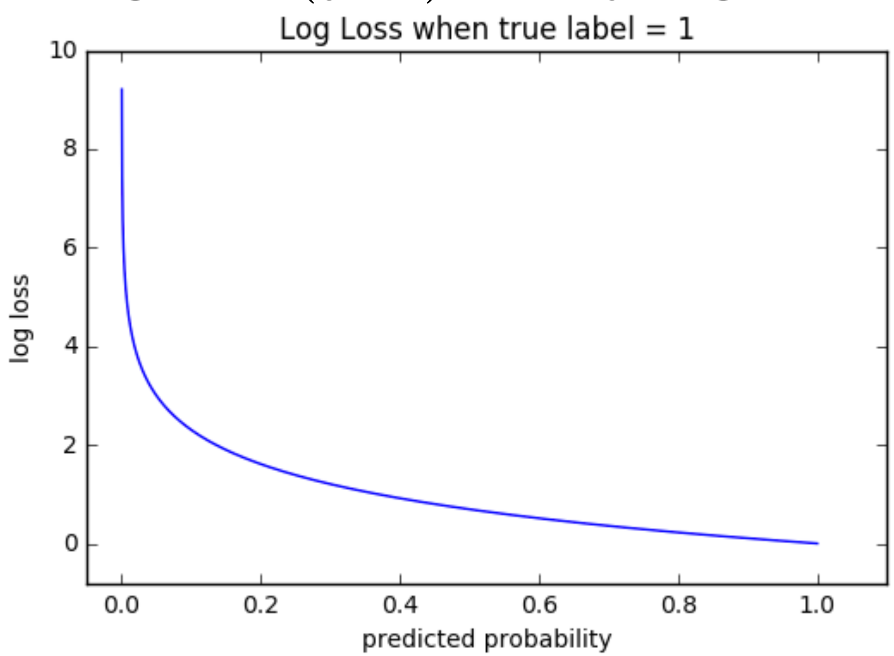
find weights and bias that minimize "loss"

Measuring prediction quality for a probabilistic classifier

Use the log loss (aka "binary cross entropy")

from sklearn.metrics import log_loss

$$\text{log\_loss}(y, \hat{p}) = -y \log \hat{p} - (1 - y) \log(1 - \hat{p})$$



Advantages:
- smooth
- easy to take derivatives!

Logistic Regression: Training

Optimization: Minimize total log loss on train set

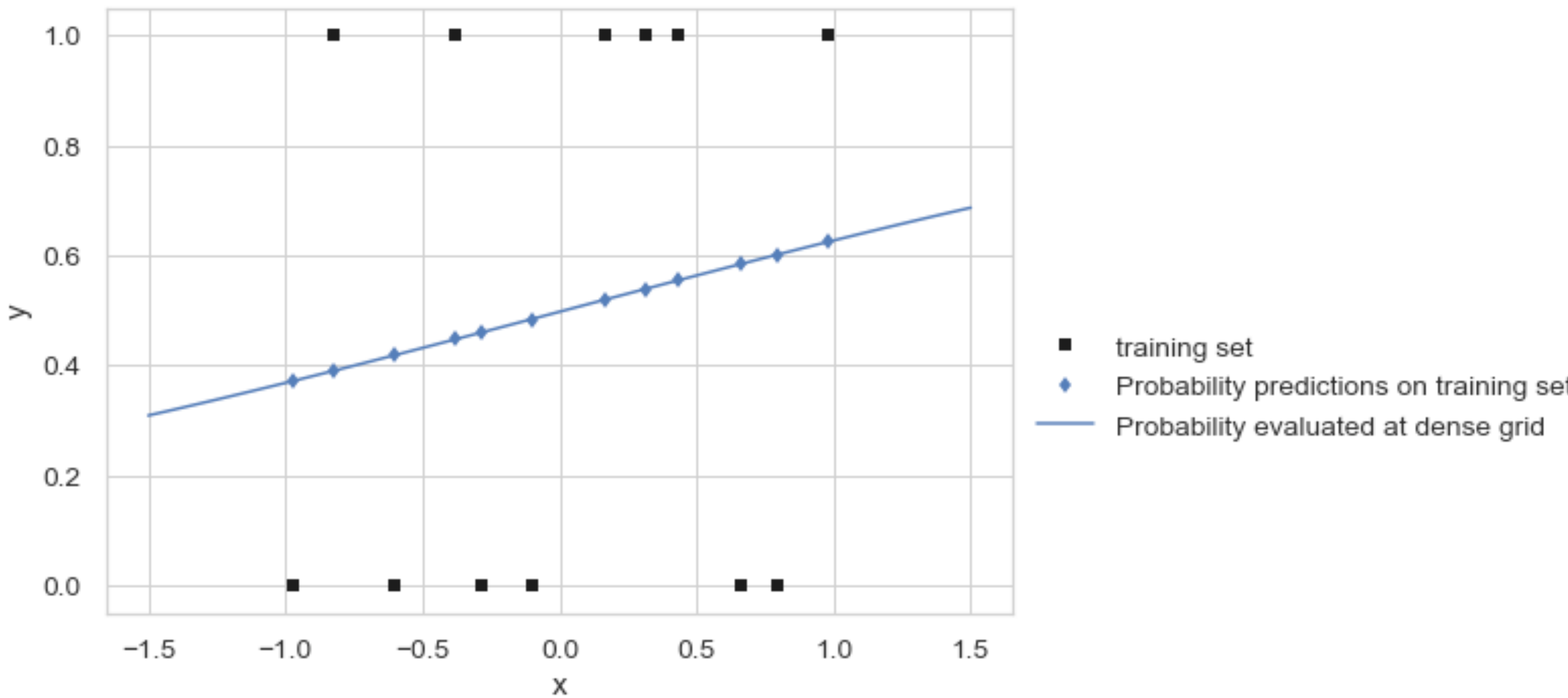
$$\min_{w,b} \sum_{n=1}^{N} \text{log\_loss}(y_n, \hat{p}(x_n, w, b))$$

Algorithm: Gradient descent

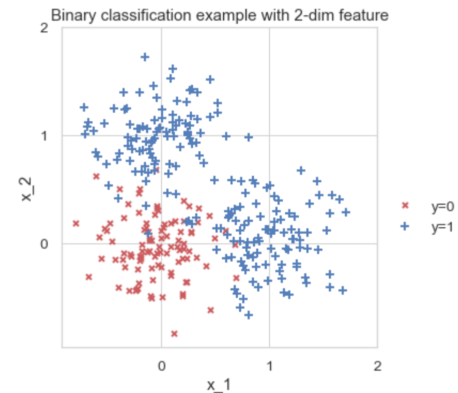
Avoid overfitting: Use L2 or L1 penalty on weights

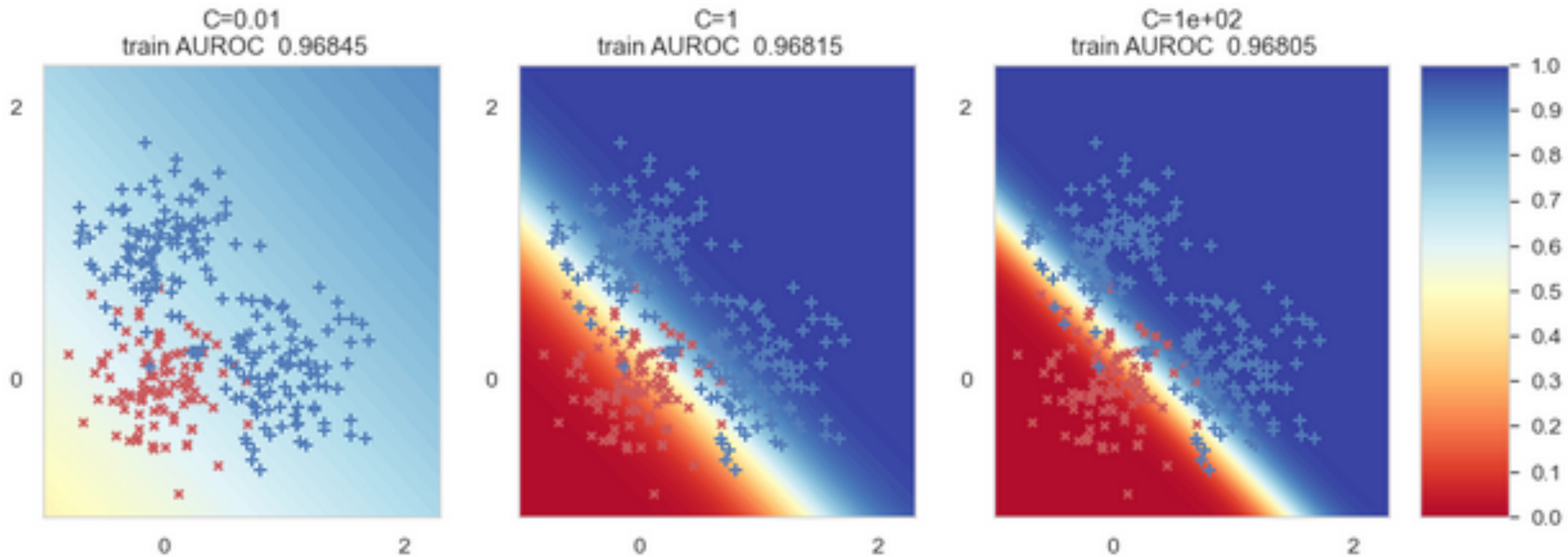
Much more in depth in next class

Visualizing predicted probas for Logistic Regression



Visualizing predicted probas for Logistic Regression





Nearest Neighbor Classifier

Parameters:

none

Prediction:

- find "nearest" training vector to given input x
- predict y value of this neighbor

Training:

none needed (use training data as lookup table)

K nearest neighbor classifier

Parameters:

K : number of neighbors

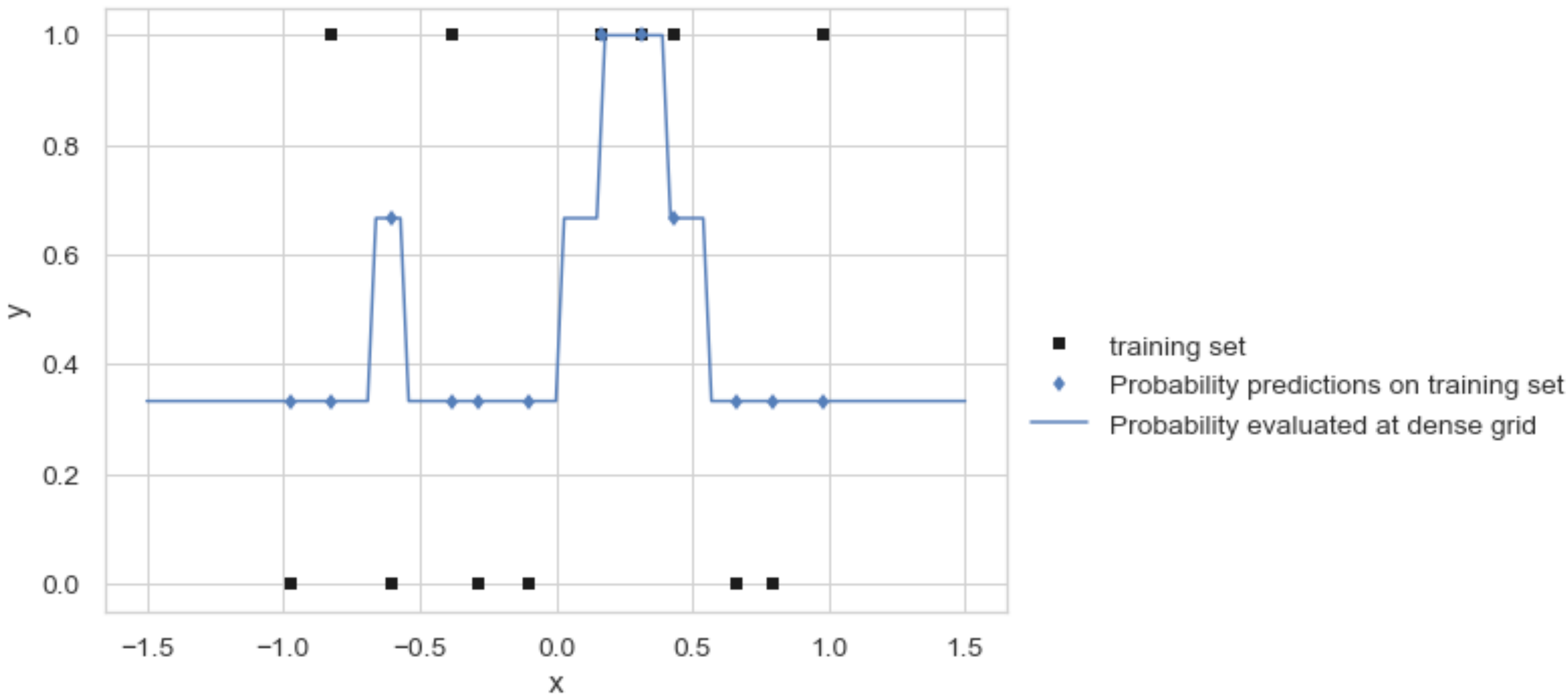
Prediction:

- find K "nearest" training vectors to input *x*
- predict: vote most common *y* in neighborhood
- predict_proba: report fraction of labels

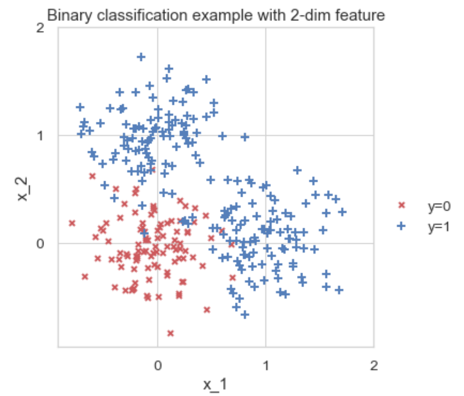
Training:

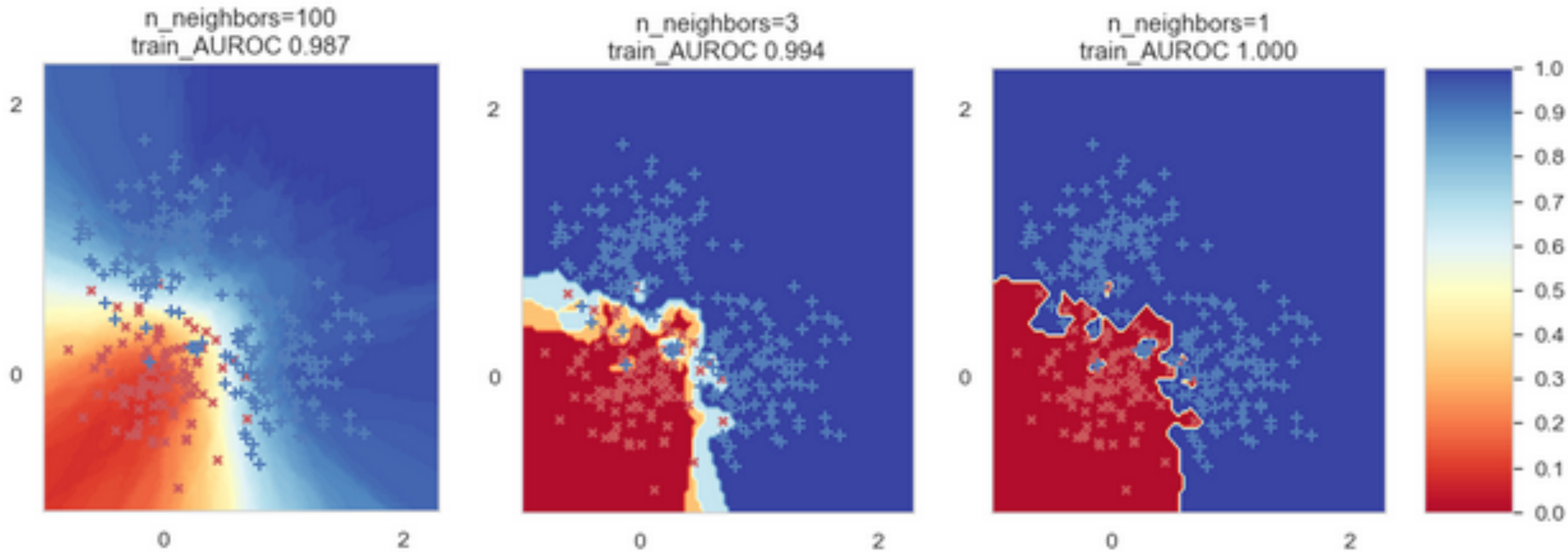
none needed (use training data as lookup table)

Visualizing predicted probas for K-Nearest Neighbors



Visualizing predicted probas for K-Nearest Neighbors





Summary of Methods

	Function class flexibility	Hyperparameters to select (control complexity)	Interpret?
Logistic Regression	Linear	L2/L1 penalty on weights	Inspect weights
K Nearest Neighbors Classifier	Piecewise constant	Number of Neighbors Distance metric	Inspect neighbors