k-Nearest Neighbors and Linear Classifiers

Saurabh Gupta

Slides from Lana Lazebnik, Justin Johnson

Outline

- Examples of classification models: nearest neighbor, linear
- Empirical loss minimization framework
- Linear classification models
  1. Linear regression
  2. Logistic regression
  3. Perceptron training algorithm
  4. Support vector machines
- General recipe: data loss, regularization
- Multi-class classification with a Softmax Function

Recall: The basic *supervised learning* framework

$$y = f(x)$$

output

prediction function

input

- **Training** (or **learning**): given a *training set* of labeled examples $\{(x_1, y_1), \dots, (x_N, y_N)\}$, instantiate a predictor f
- **Testing** (or **inference**): apply f to a new *test example* x and output the predicted value $y = f(x)$

k-Nearest Neighbors

First classifier: **Nearest Neighbor**

```
def train(images, labels):
  # Machine learning!
  return model
```

→ Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

Slide from Justin Johnson

Distance Metric to compare images

L1 distance: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

\-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

$\xrightarrow{\text{add}}$ 456

Slide from Justin Johnson

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Slide from Justin Johnson

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor Classifier

Memorize training data

Slide from Justin Johnson

Nearest Neighbor Classifier

```
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

For each test image:
  Find nearest training image
  Return label of nearest image

Slide from Justin Johnson

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor Classifier

Q: With N examples, how fast is training?

Slide from Justin Johnson

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor Classifier

Q: With N examples, how fast is training?

A: O(1)

Slide from Justin Johnson

Nearest Neighbor Classifier

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Q: With N examples, how fast is training?

A: O(1)

Q: With N examples, how fast is testing?

Slide from Justin Johnson

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor Classifier

Q: With N examples, how fast is training?

A: O(1)

Q: With N examples, how fast is testing?

A: O(N)

Slide from Justin Johnson

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor Classifier

Q: With N examples, how fast is training?

A: O(1)

Q: With N examples, how fast is testing?

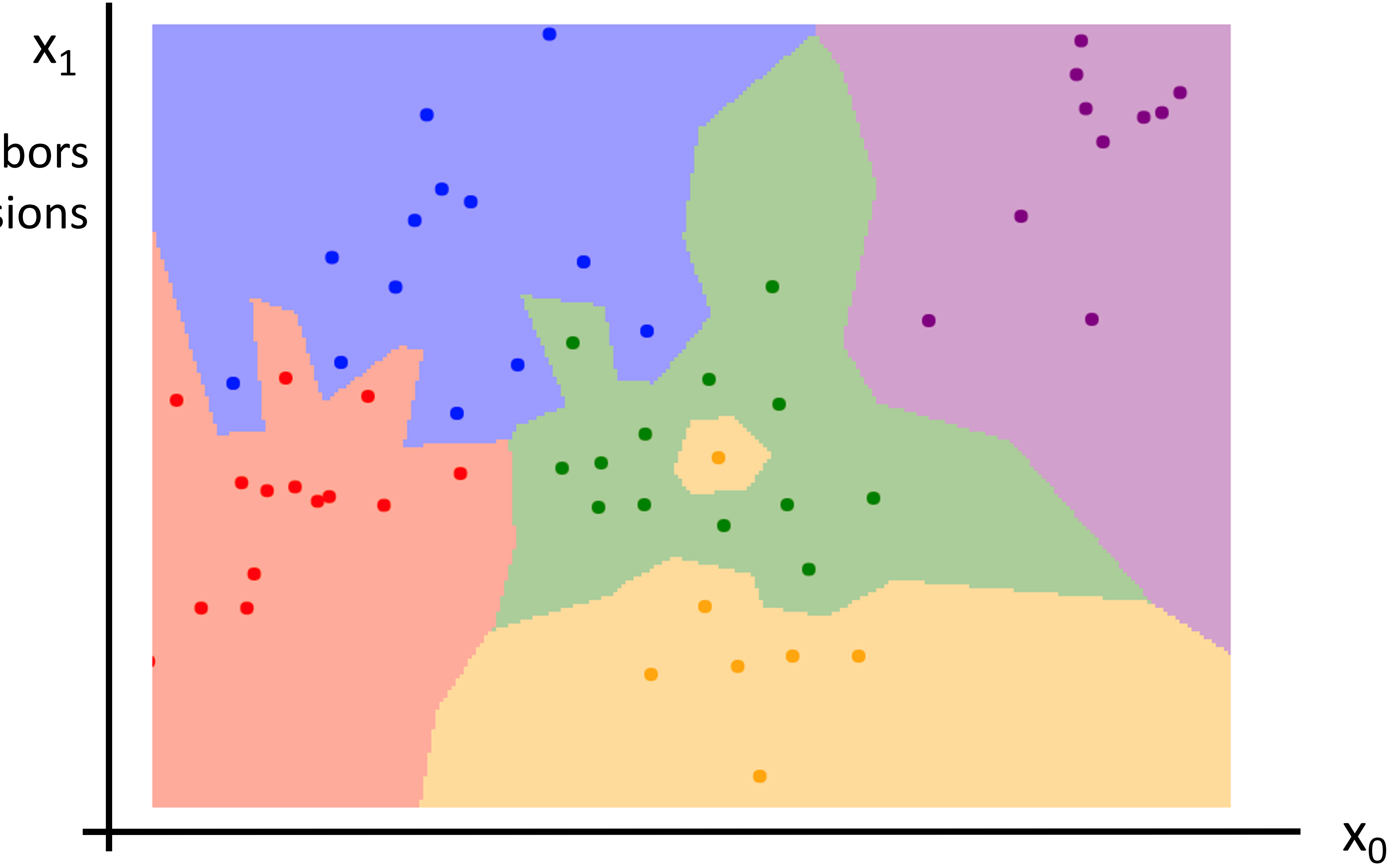
A: O(N)

This is **bad**: We can afford slow training, but we need fast testing!

Slide from Justin Johnson

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

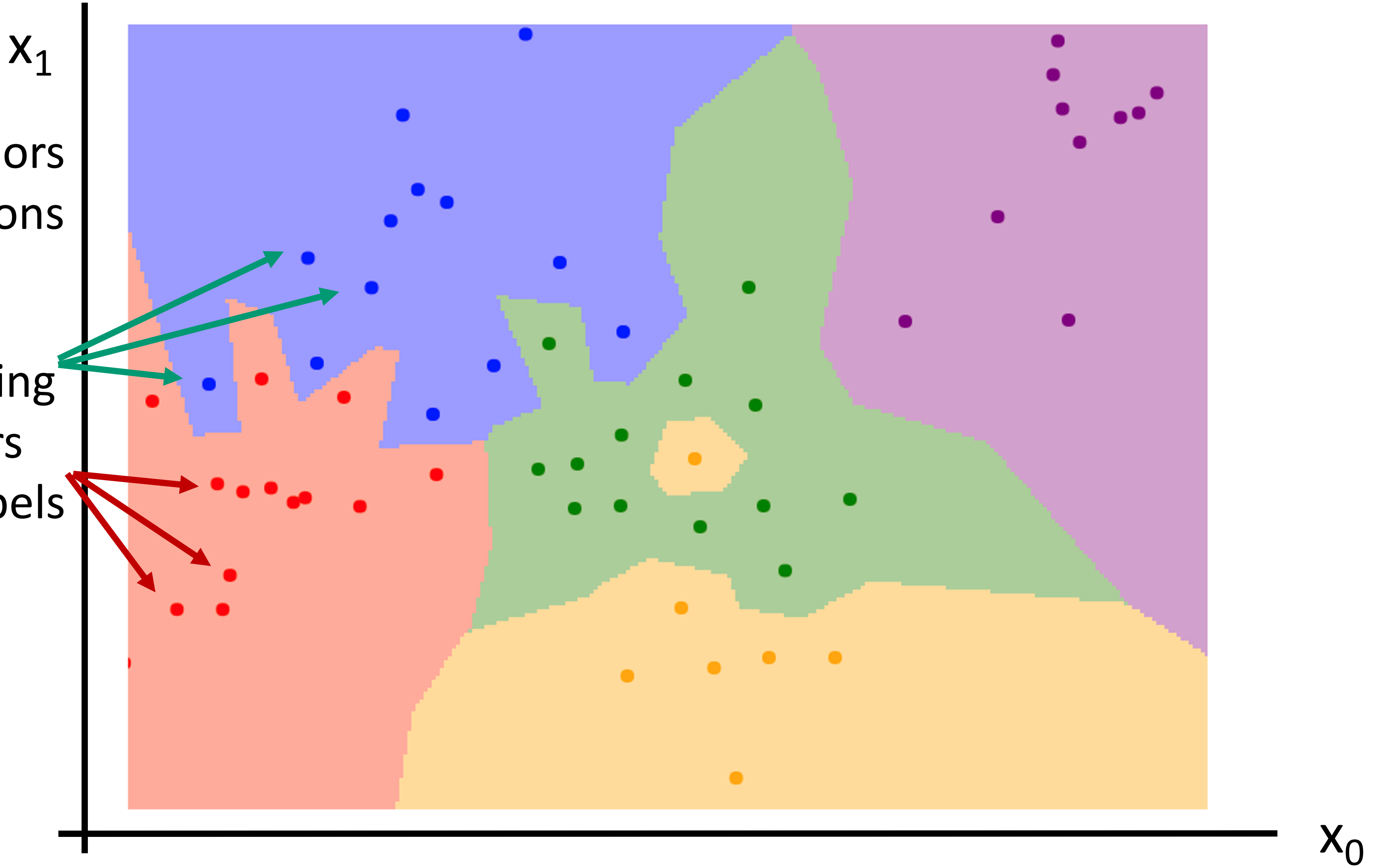


Slide from Justin Johnson

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels



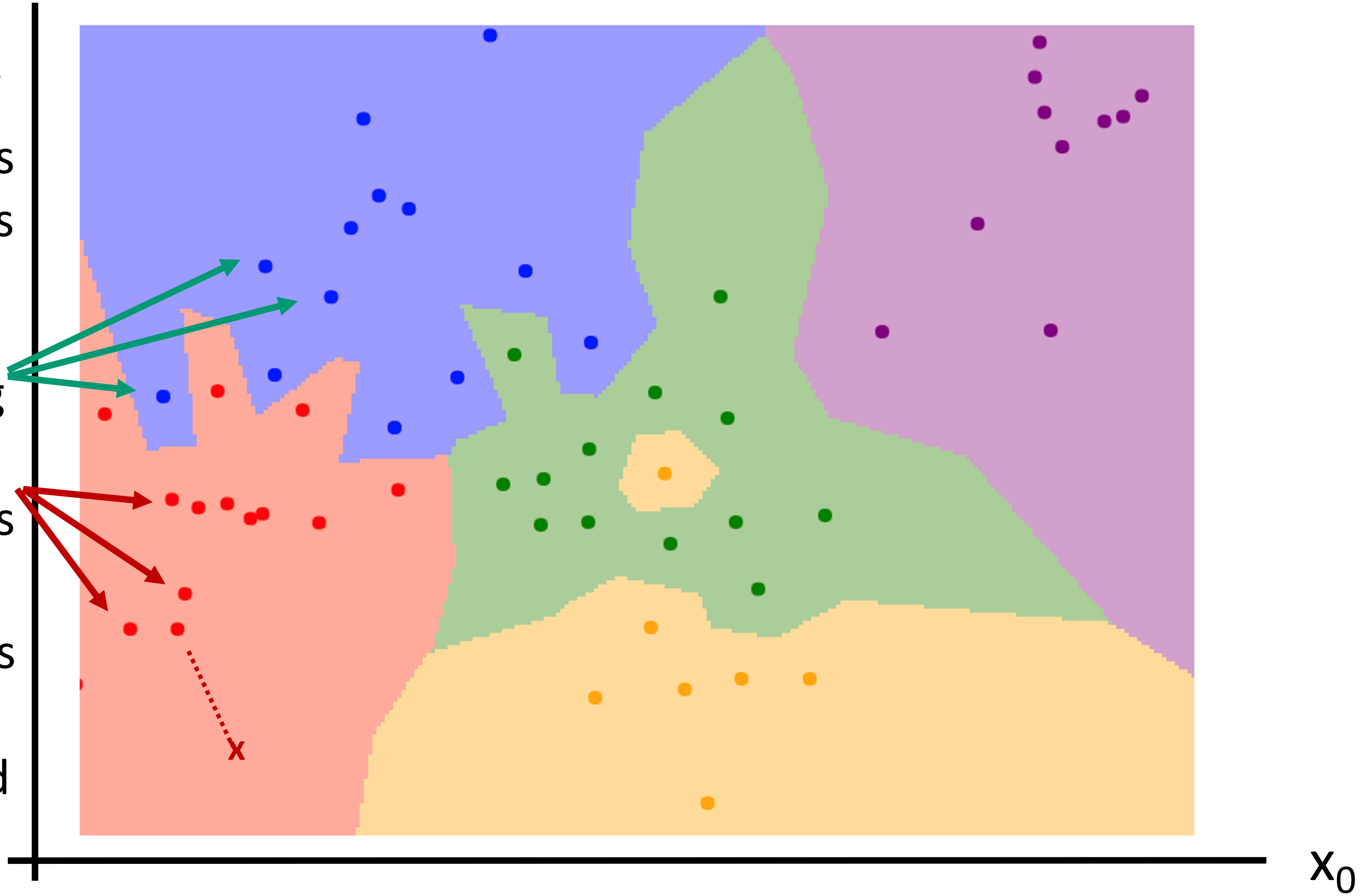
Slide from Justin Johnson

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



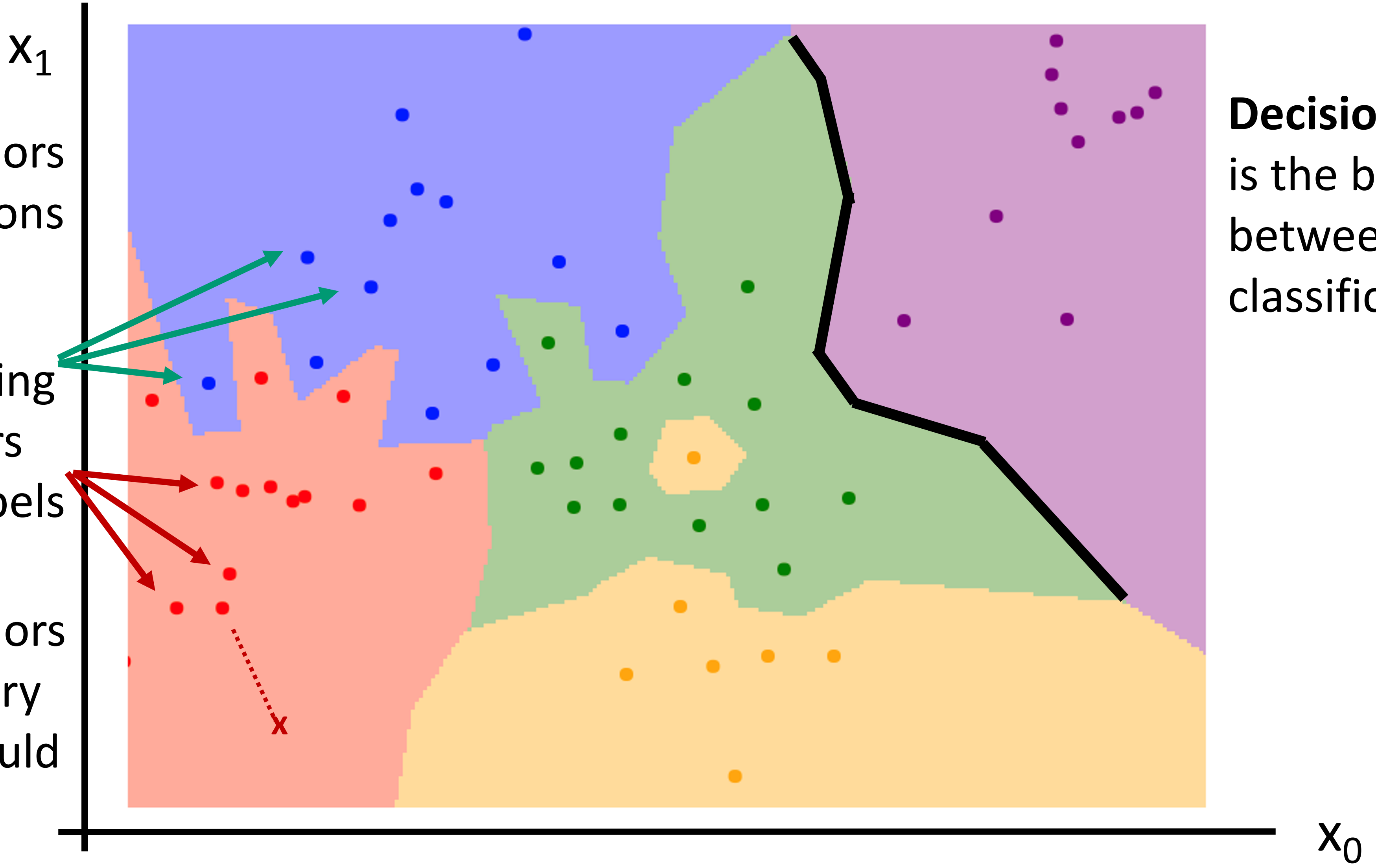
Slide from Justin Johnson

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



Decision boundary is the boundary between two classification regions

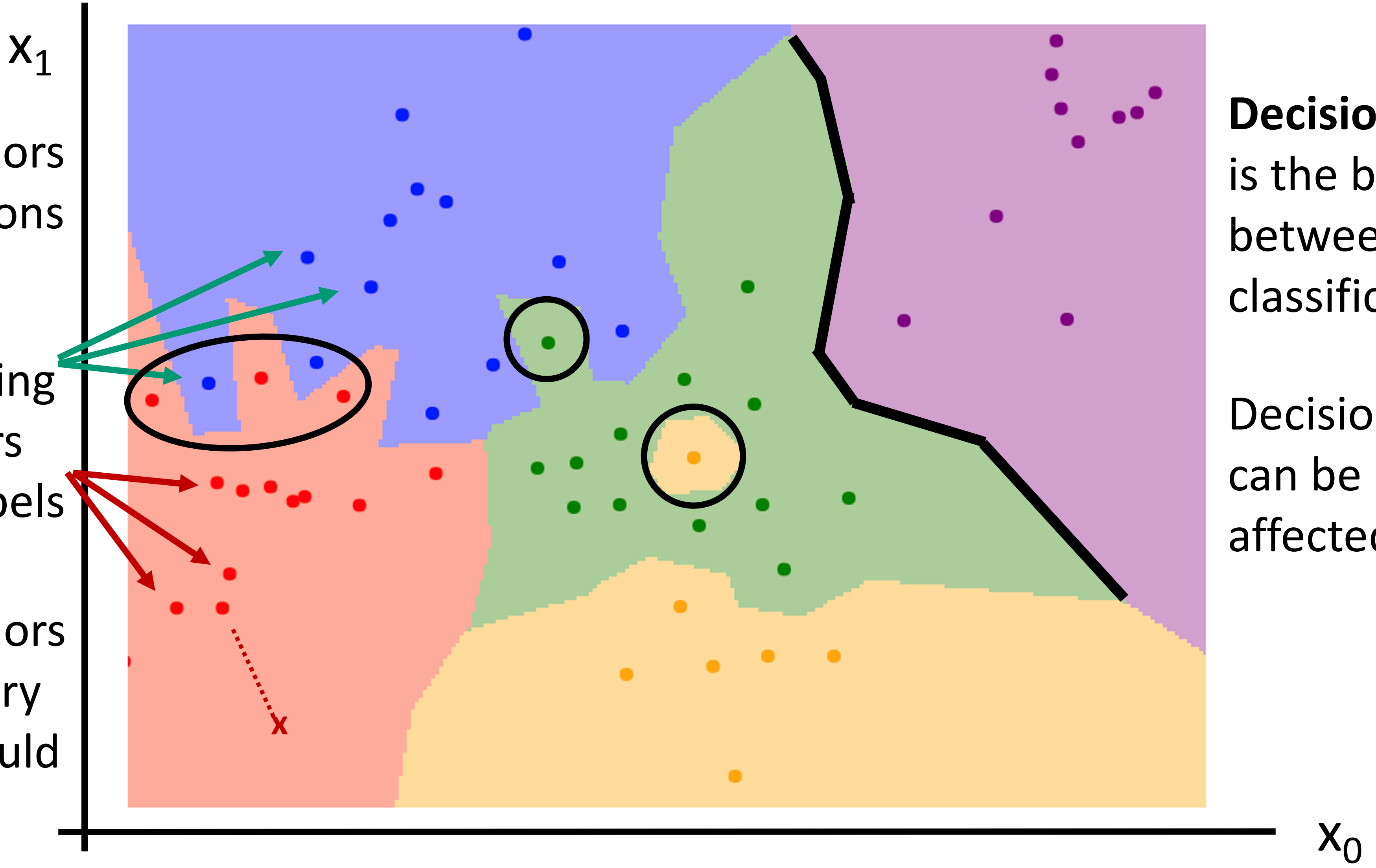
Slide from Justin Johnson

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



Decision boundary is the boundary between two classification regions

Decision boundaries can be noisy; affected by outliers

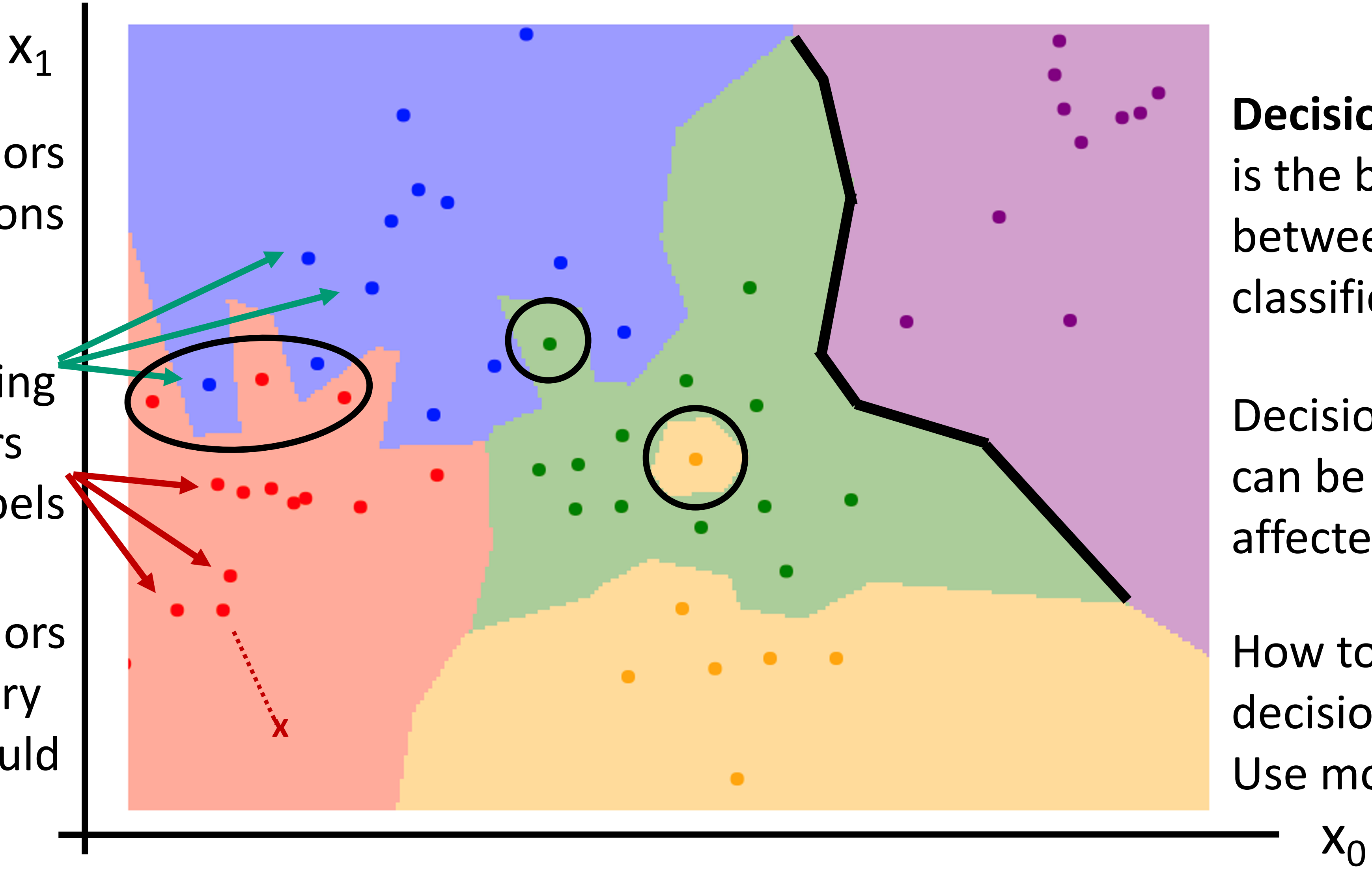
Slide from Justin Johnson

Nearest Neighbor **Decision Boundaries**

Nearest neighbors in two dimensions

Points are training examples; colors give training labels

Background colors give the category a test point would be assigned



Decision boundary is the boundary between two classification regions

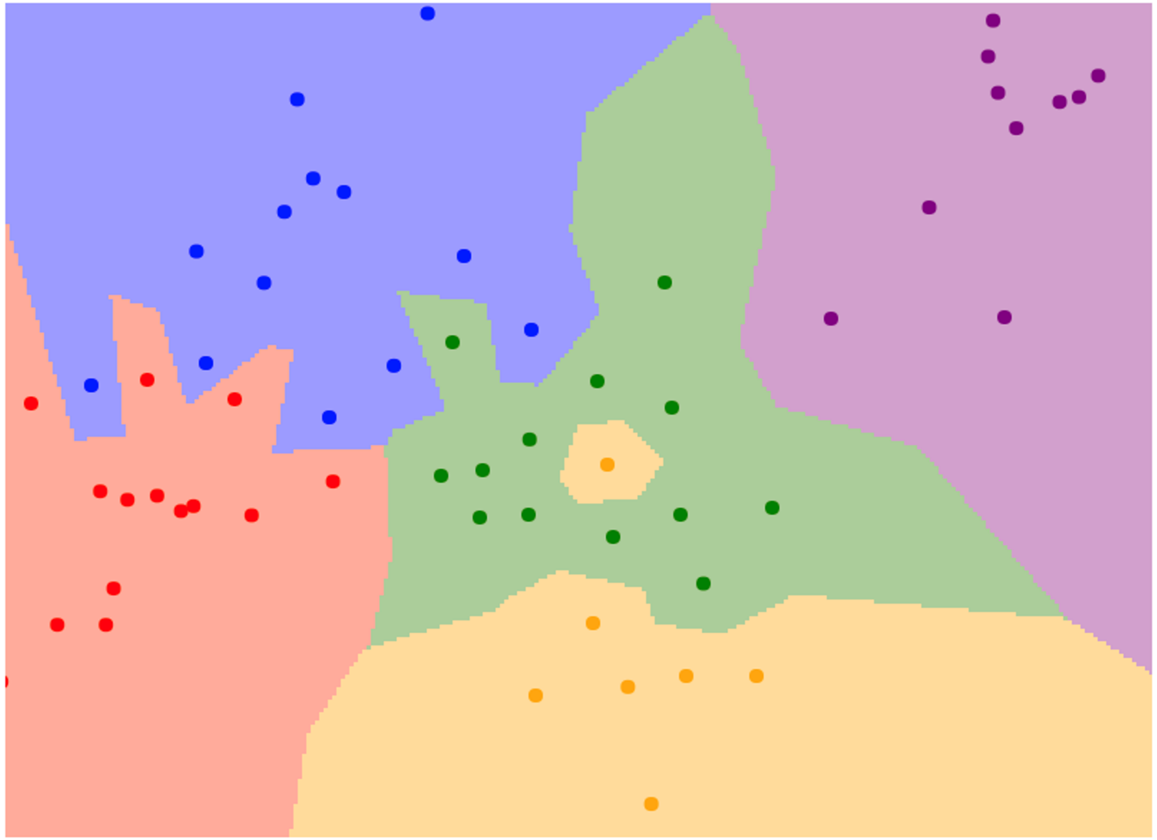
Decision boundaries can be noisy; affected by outliers

How to smooth out decision boundaries? Use more neighbors!

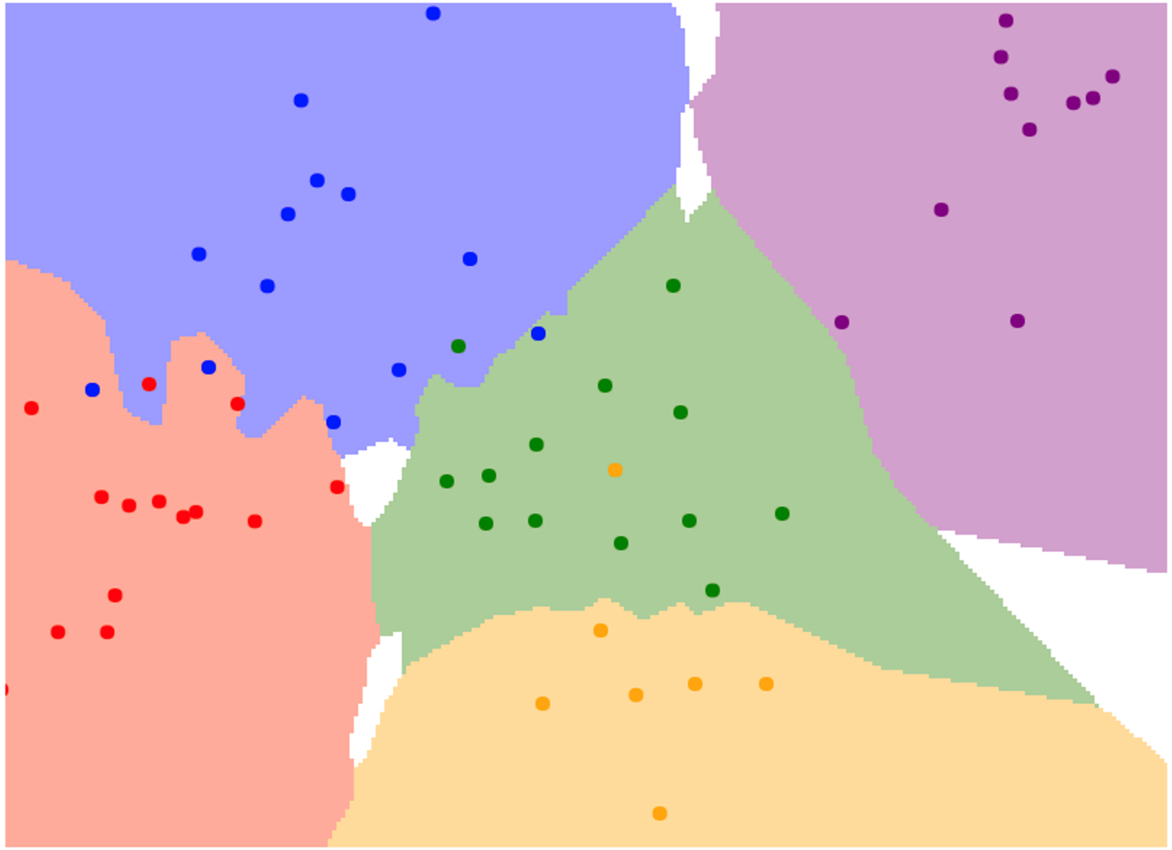
Slide from Justin Johnson

K-Nearest Neighbors

K = 1



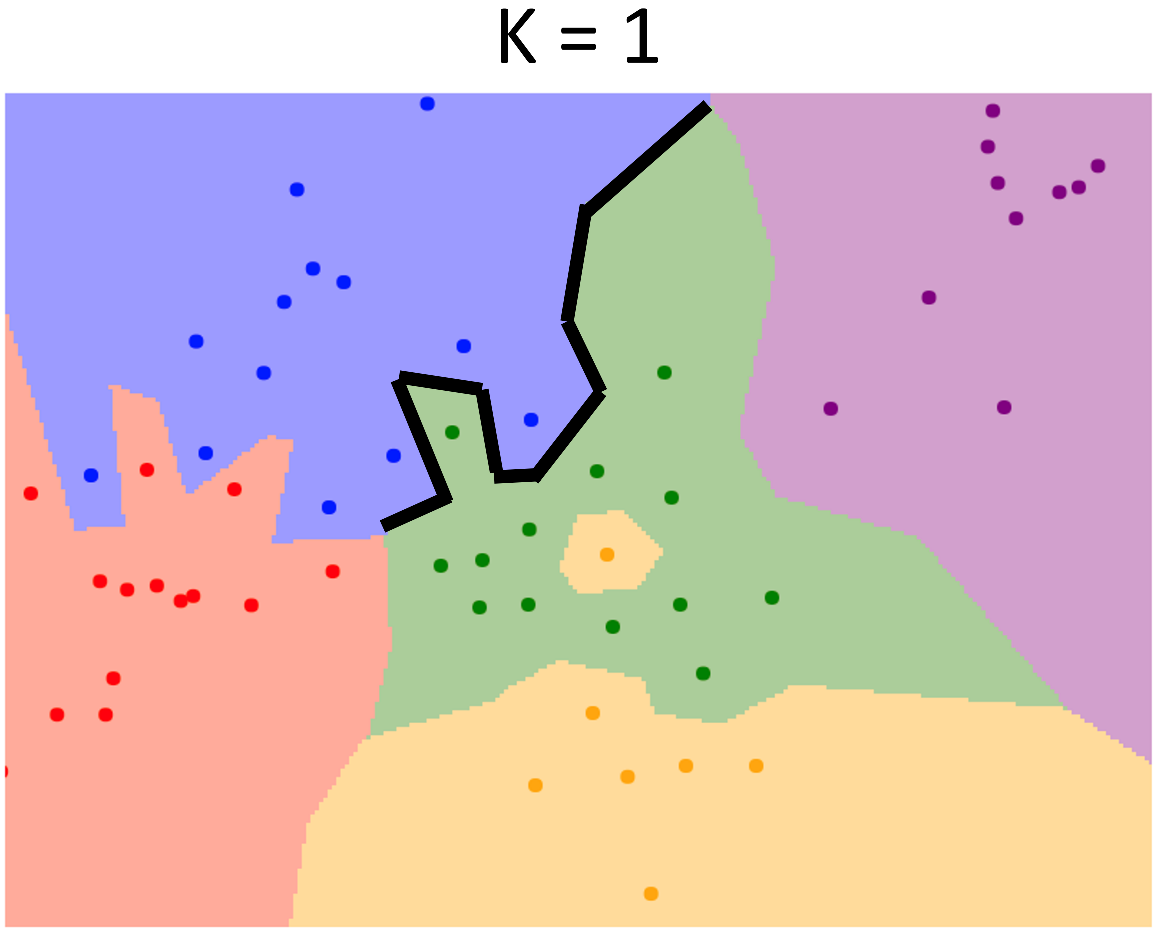
K = 3

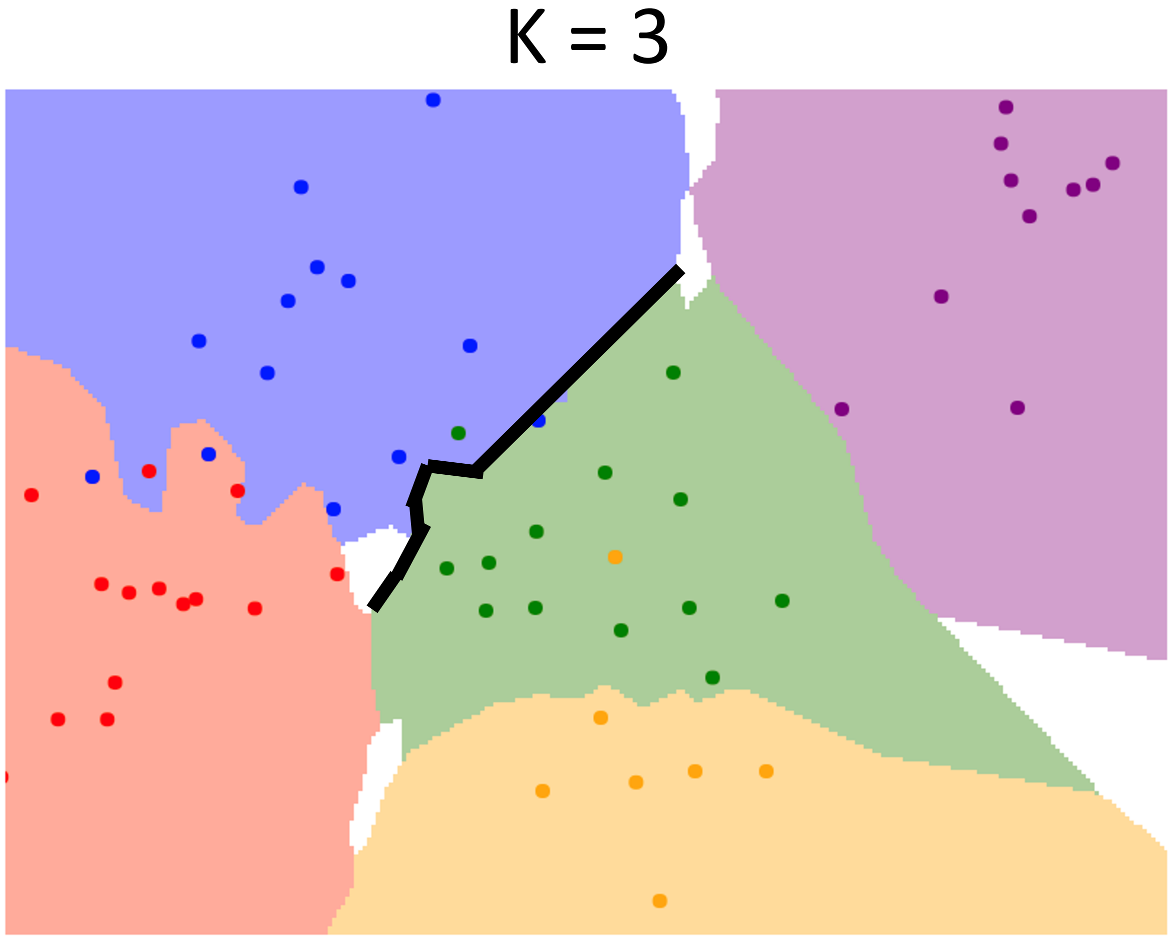


Instead of copying label from nearest neighbor, take **majority vote** from K closest points

Slide from Justin Johnson

K-Nearest Neighbors



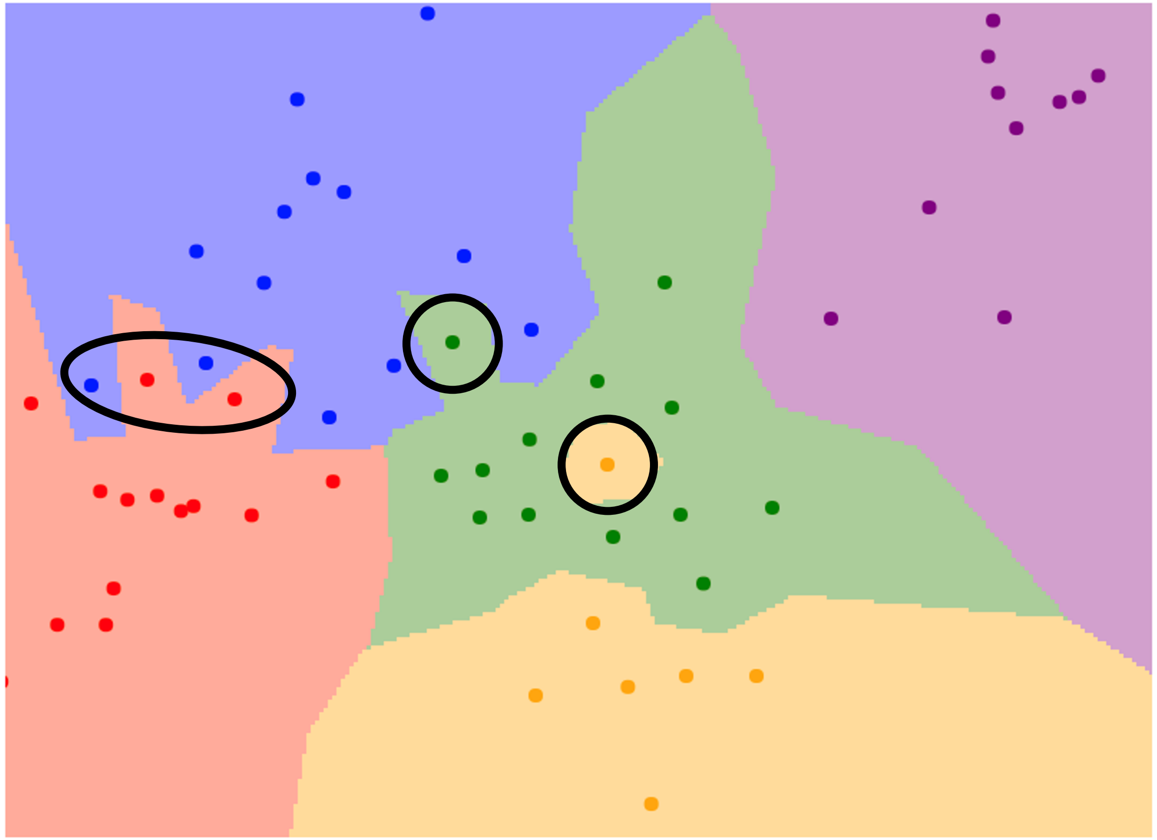


Using more neighbors helps smooth out rough decision boundaries

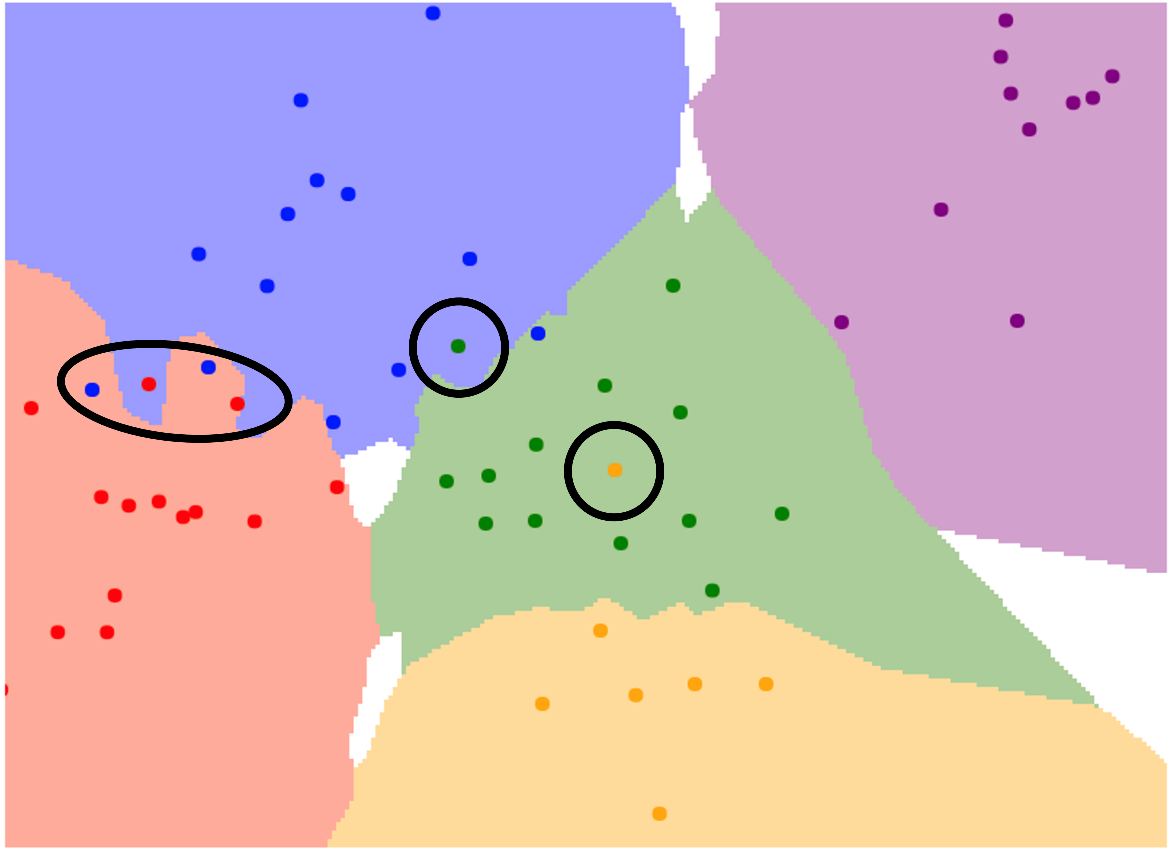
Slide from Justin Johnson

K-Nearest Neighbors

K = 1



K = 3

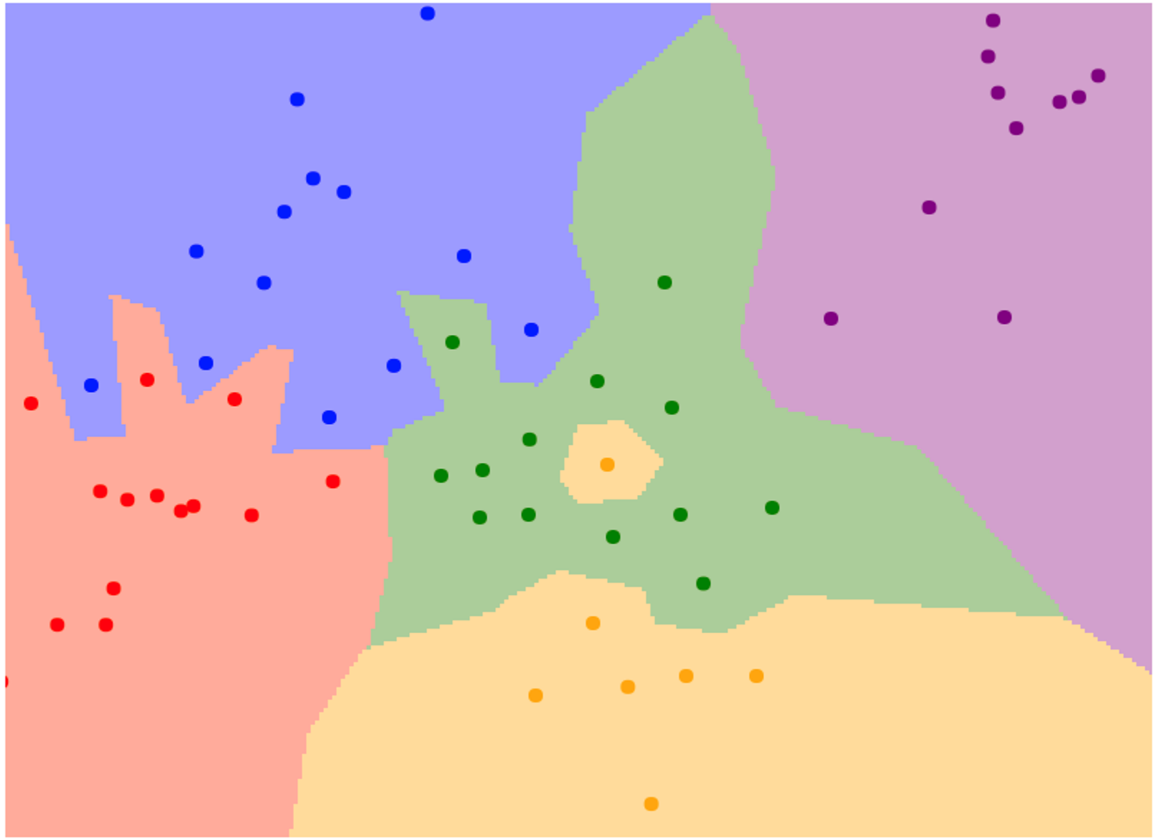


Using more neighbors helps reduce the effect of outliers

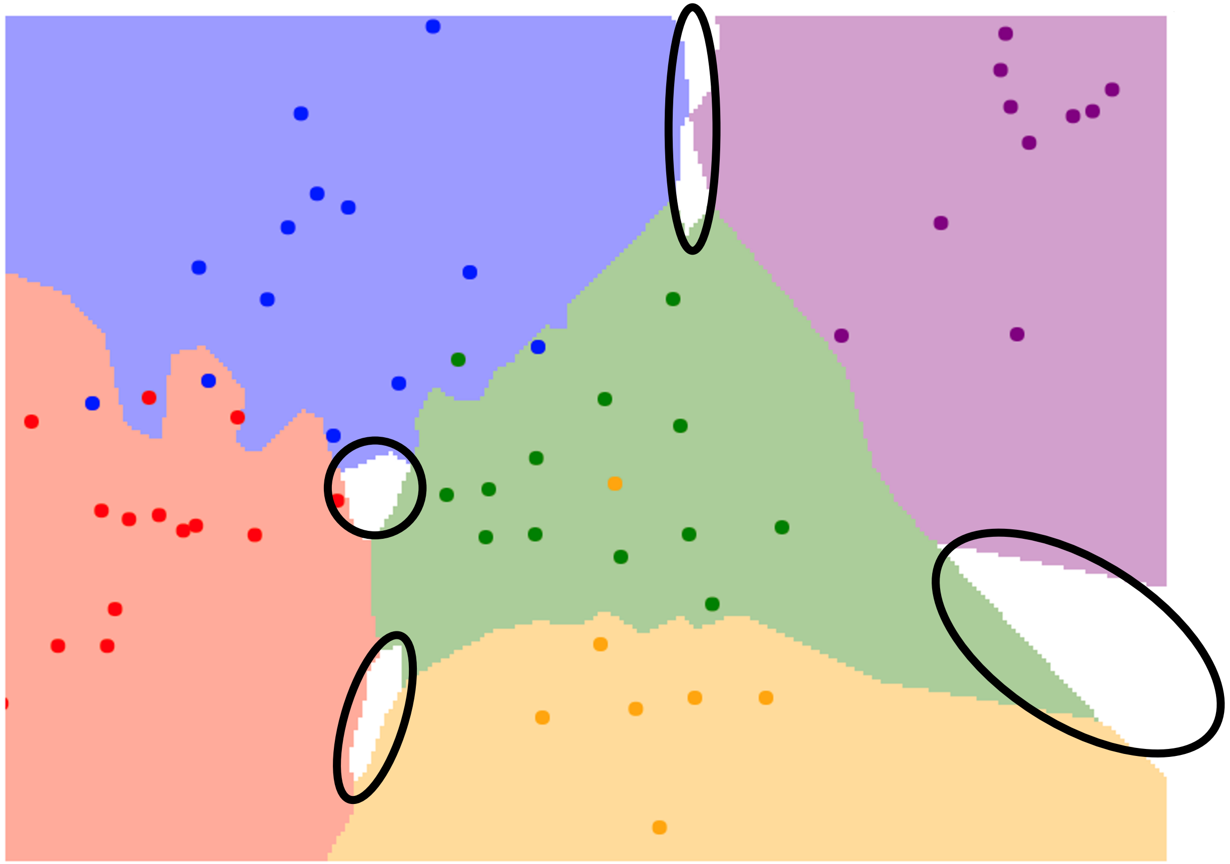
Slide from Justin Johnson

K-Nearest Neighbors

K = 1



K = 3



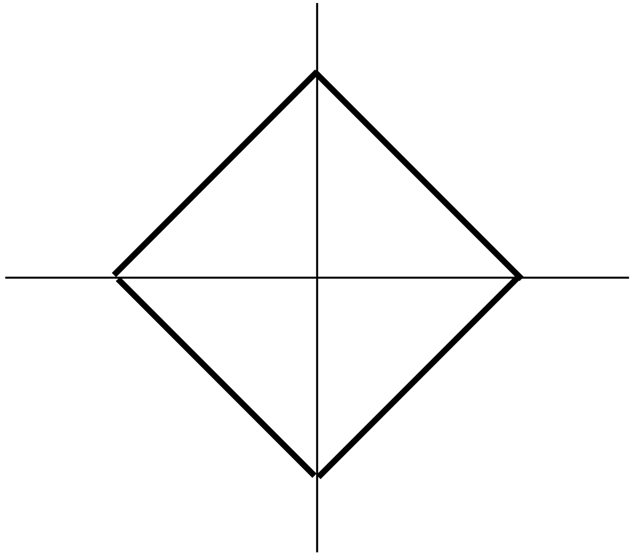
When K > 1 there can be ties between classes. Need to break somehow!

Slide from Justin Johnson

K-Nearest Neighbors: **Distance Metric**

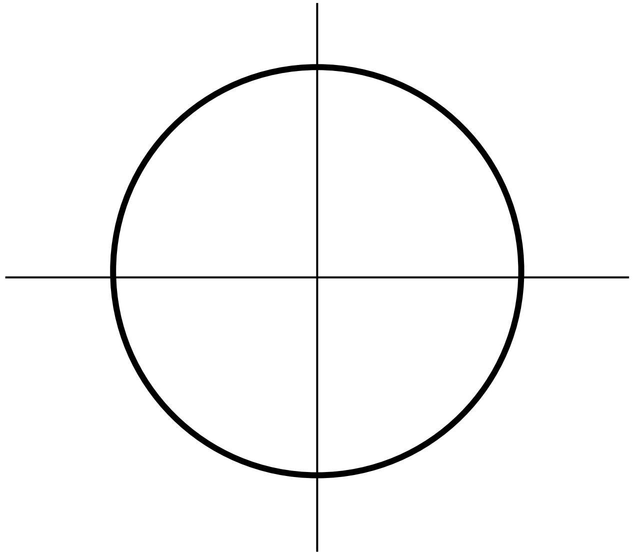
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p \left| I_1^p - I_2^p \right|$$



L2 (Euclidean) distance

$$d_1(I_1, I_2) = \left(\sum_p \left(I_1^p - I_2^p \right)^2 \right)^{\frac{1}{2}}$$

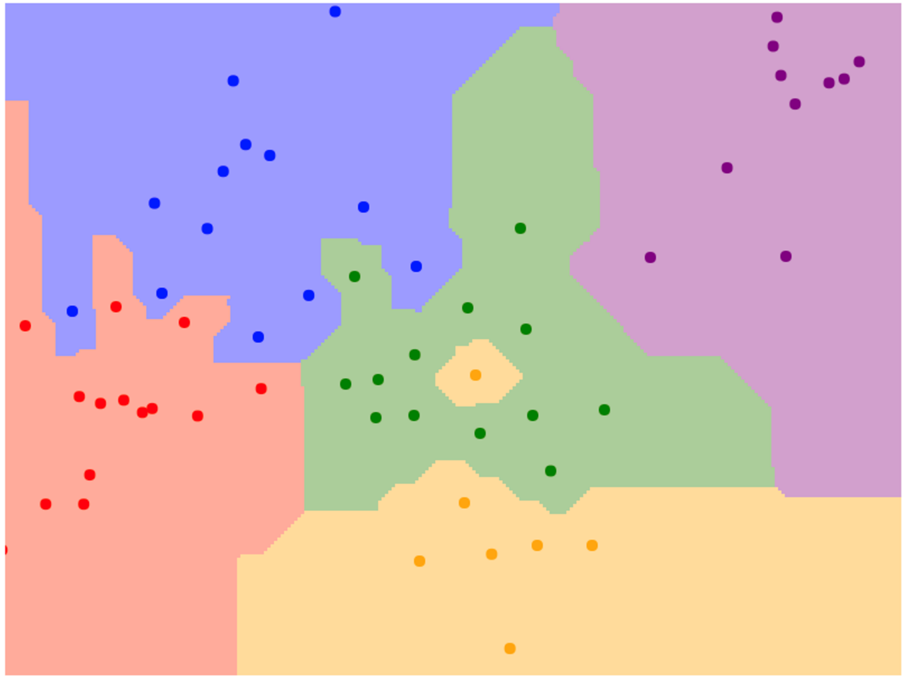


Slide from Justin Johnson

K-Nearest Neighbors: **Distance Metric**

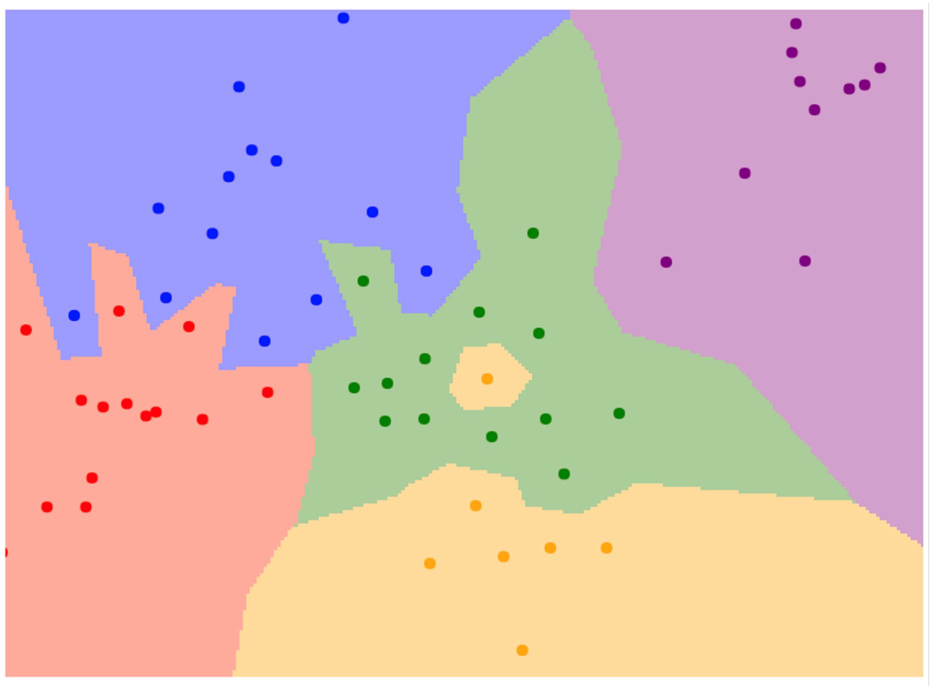
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p \left| I_1^p - I_2^p \right|$$



L2 (Euclidean) distance

$$d_1(I_1, I_2) = \left(\sum_p \left(I_1^p - I_2^p \right)^2 \right)^{\frac{1}{2}}$$



K = 1

Slide from Justin Johnson

Hyperparameters

What is the best value of **K** to use?
What is the best **distance metric** to use?

These are examples of **hyperparameters**: choices about our learning algorithm that we don’t learn from the training data; instead we set them at the start of the learning process

Hyperparameters

What is the best value of **K** to use?
What is the best **distance metric** to use?

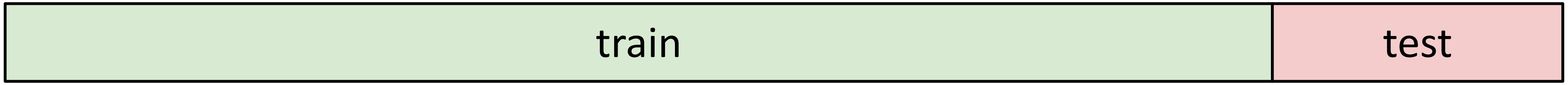
These are examples of **hyperparameters**: choices about our learning algorithm that we don't learn from the training data; instead we set them at the start of the learning process

Very problem-dependent. In general need to try them all and see what works best for our data / task.

Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the training data

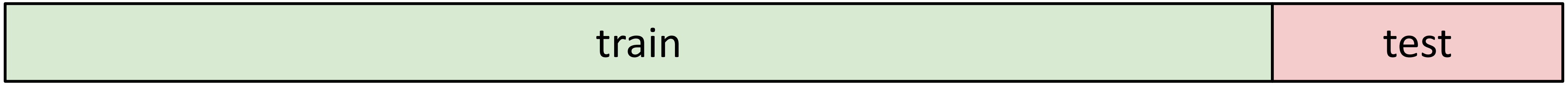
BAD: K = 1 always works perfectly on training data (in general, memorization is sufficient for acing the train set)



Setting Hyperparameters

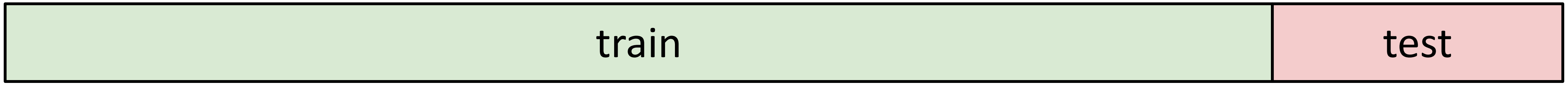
Idea #1: Choose hyperparameters that work best on the training data

BAD: K = 1 always works perfectly on training data (in general, memorization is sufficient for acing the train set)



Idea #2: Choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data.

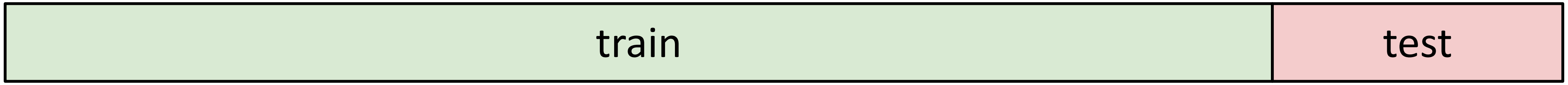


Slide from Justin Johnson

Setting Hyperparameters

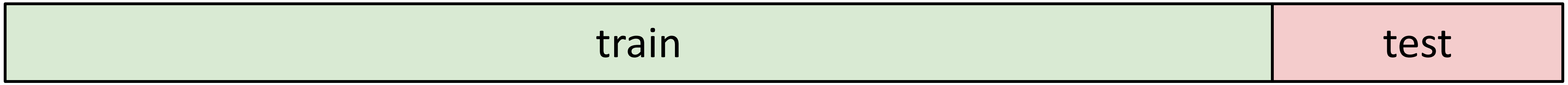
Idea #1: Choose hyperparameters that work best on the training data

BAD: K = 1 always works perfectly on training data (in general, memorization is sufficient for acing the train set)



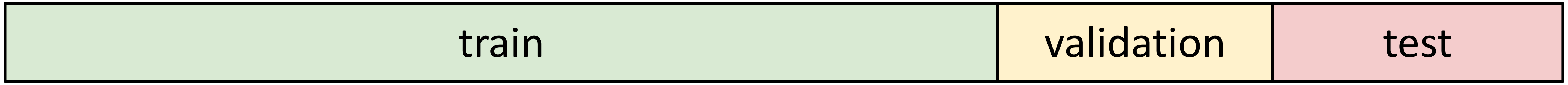
Idea #2: Choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data.



Idea #3: Split dataset into **train** and **val**; choose hyperparameters on val and evaluate on test.

Better!



Slide from Justin Johnson

Setting Hyperparameters

Your Dataset

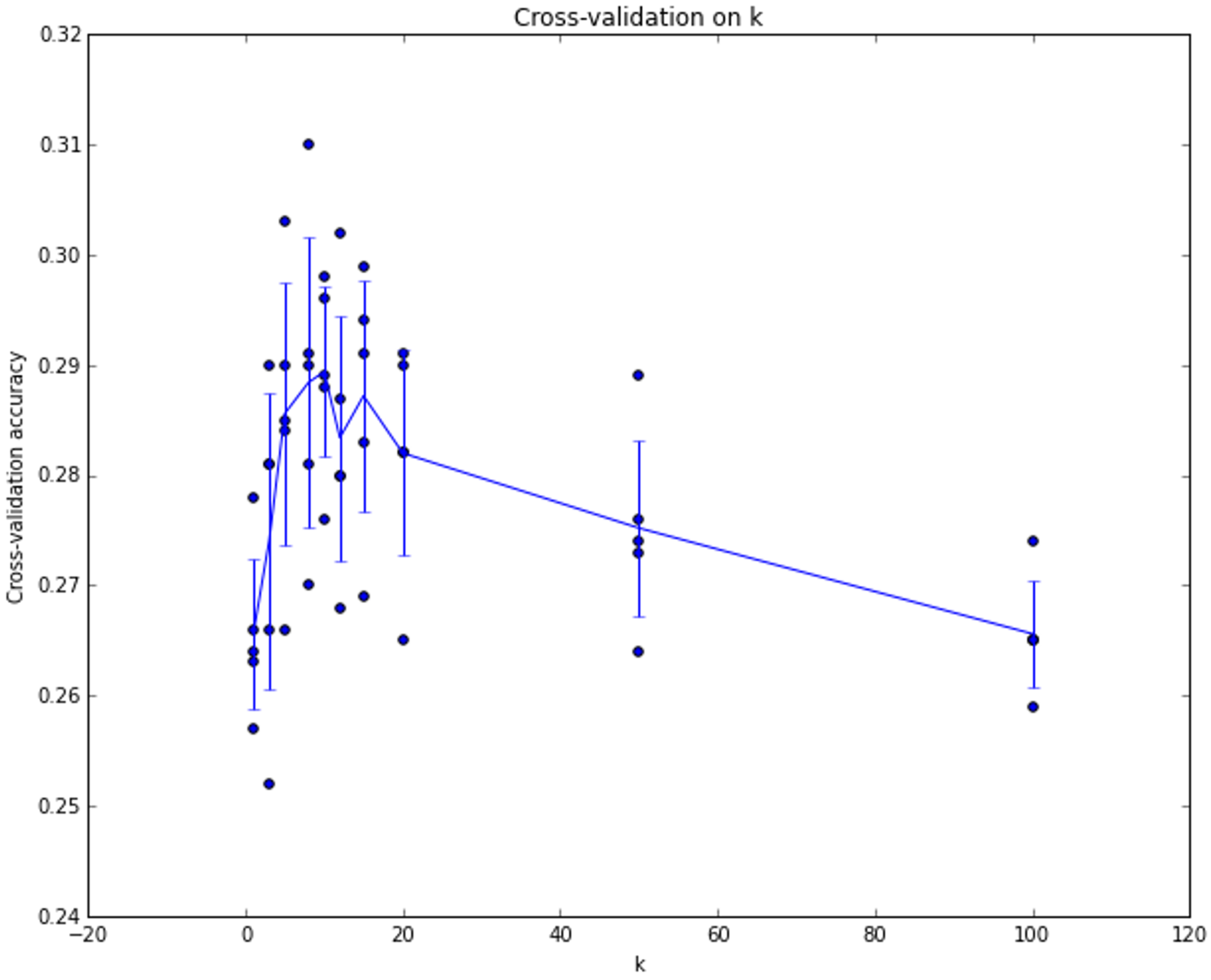
Idea #4: **Cross-Validation**: Split data into **folds**, try each fold as validation and average the results

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but (unfortunately) not used too frequently in deep learning

Slide from Justin Johnson

Setting Hyperparameters



Example of 5-fold cross-validation for the value of **k.**

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

(Seems that k ~ 7 works best for this data)

Slide from Justin Johnson

K-Nearest Neighbor on raw pixels is **seldom used**

- Very slow at test time
- Distance metrics on pixels are not informative

Original

Boxed

Shifted

Tinted

(all 3 images have same L2 distance to the one on the left)

Slide from Justin Johnson

Nearest Neighbor with ConvNet features works well!

Devlin et al, "Exploring Nearest Neighbor Approaches for Image Captioning", 2015

Slide from Justin Johnson

Can transfer more than just label!

Image Captioning with Nearest Neighbor

A bedroom with a bed and a couch.

A cat sitting in a bathroom sink.

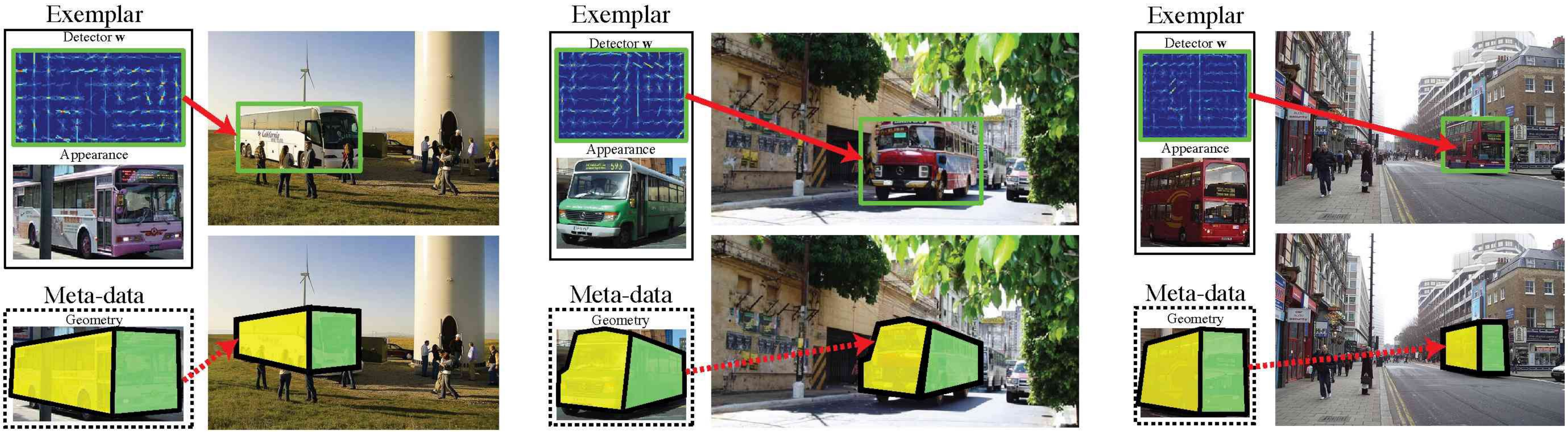
A train is stopped at a train station.

A wooden bench in front of a building.

Devlin et al, "[Exploring Nearest Neighbor Approaches for Image Captioning]", 2015

Slide from Justin Johnson

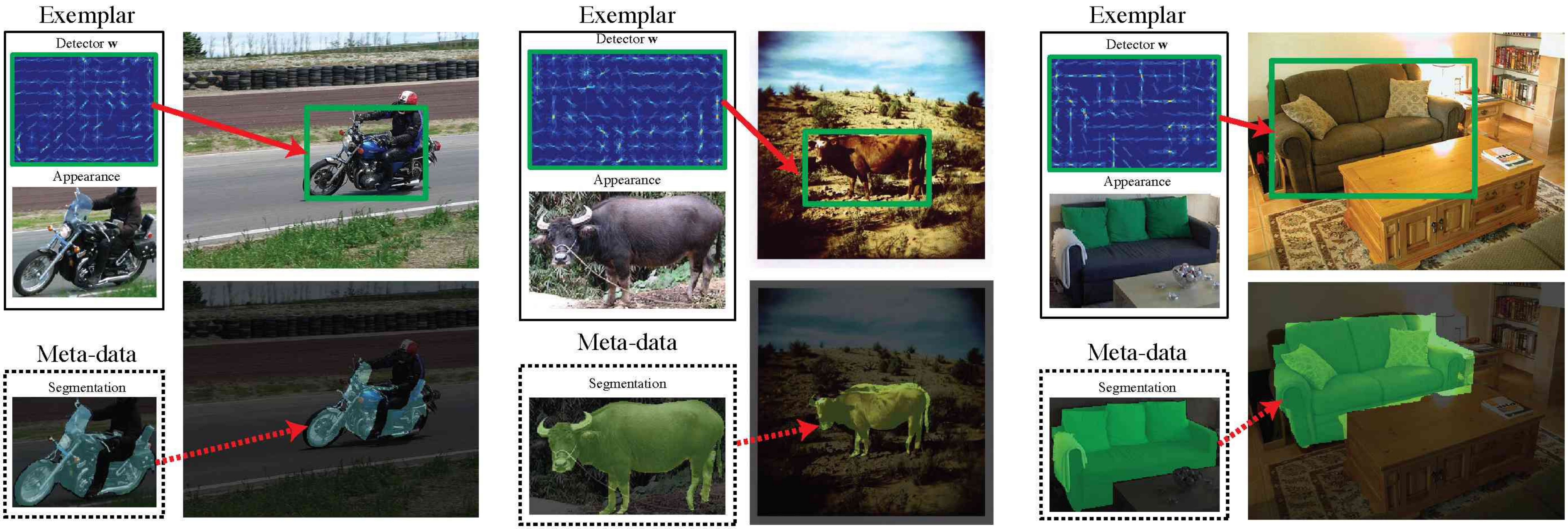
Can transfer more than just label!



Malisiewicz et al, "[Ensemble of Exemplar-SVMs for Object Detection and Beyond]", ICCV 2011

Slide from Justin Johnson

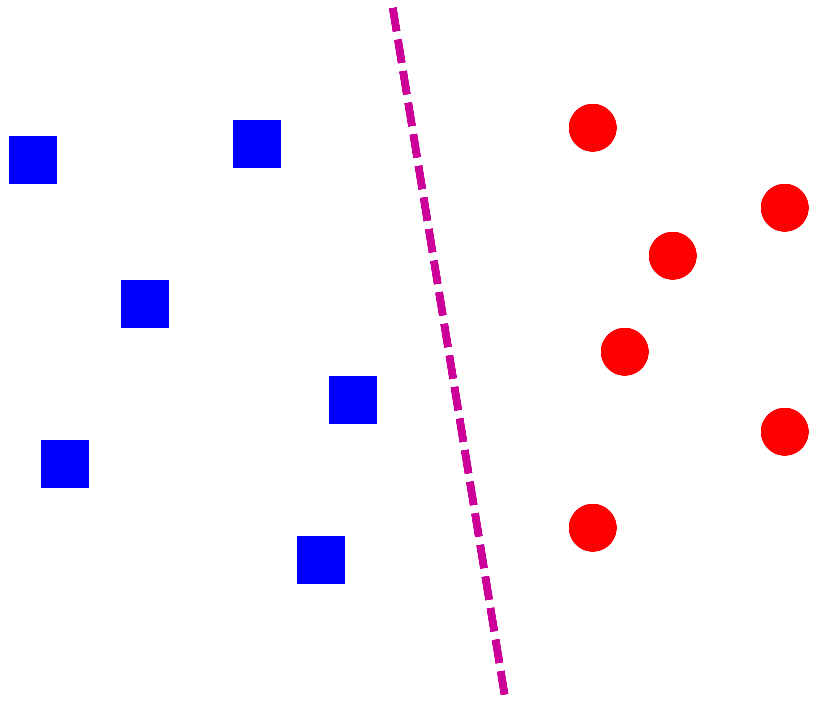
Can transfer more than just label!



Malisiewicz et al, "[Ensemble of Exemplar-SVMs for Object Detection and Beyond]()", ICCV 2011

Slide from Justin Johnson

Linear classifier



- Find a *linear function* to separate the classes:

$$f(x) = \mathrm{sgn}\left(w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \cdots + w^{(D)}x^{(D)} + b\right) = \mathrm{sgn}(w \cdot x + b)$$

Outline

- Examples of classification models: nearest neighbor, linear
- Empirical loss minimization framework

Empirical loss minimization

- Let's formalize the setting for learning of a *parametric model* in a supervised scenario

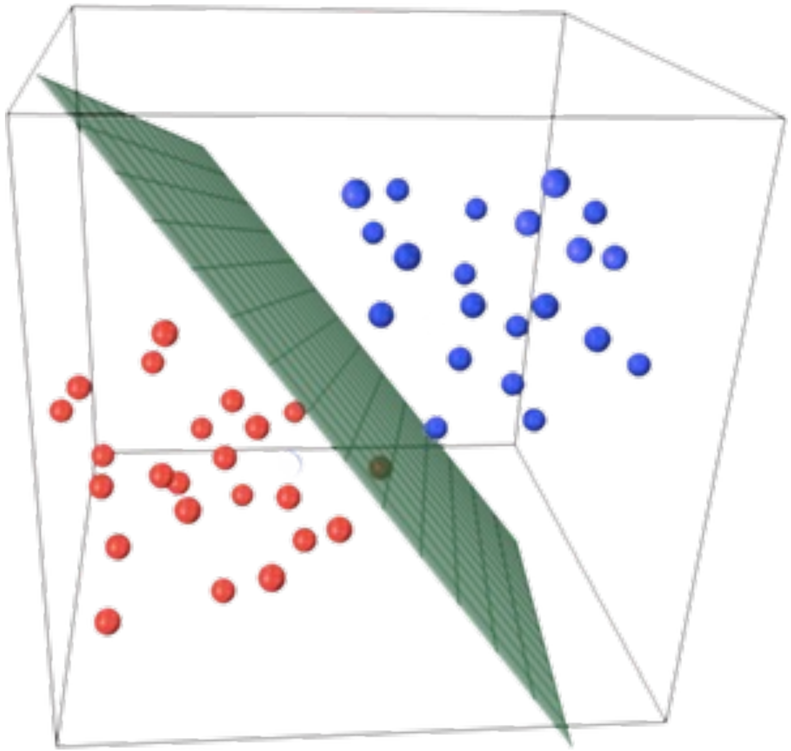


Image source

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$
- Find: predictor f
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Source: Y. Liang

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$
- Find: predictor f
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

What kinds of functions?

Source: [Y. Liang]

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Hypothesis class

Source: [Y. Liang]

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Connection between training and test data?

Source: Y. Liang

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data i.i.d. from distribution D

Source: [Y. Liang]

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data i.i.d. from distribution D

What kind of performance measure?

Source: [Y. Liang]

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

Various loss functions

Source: Y. Liang

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

- Example losses:

$0 - 1$ loss: $l(f, x, y) = \mathbb{I}[f(x) \neq y]$ and $L(f) = \Pr[f(x) \neq y]$

l_2 loss: $l(f, x, y) = [f(x) - y]^2$ and $L(f) = \mathbb{E}[\,[f(x) - y]^2]$

Source: Y. Liang

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

Can't optimize this directly

Source: Y. Liang

Empirical loss minimization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$ that minimizes

$$\hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} l(f, x_i, y_i)$$

Empirical loss

Source: [Y. Liang]()

Supervised learning in a nutshell

1. **Collect *training data* and labels**
2. **Specify model:** select *hypothesis class* and *loss function*
3. **Train model:** find the function in the hypothesis class that minimizes the *empirical loss* on the training data

Outline

- Example classification models: nearest neighbor, linear
- Empirical loss minimization
- Linear classification models
  1. Linear regression
  2. Logistic regression
  3. Perceptron training algorithm
  4. Support vector machines

Training linear classifiers

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \dots, n\},$
  $y_i \in \{-1,1\}$
- Hypothesis class: $f_w(x) = \text{sgn}(w^T x)$
- Classification with *bias*, i.e. $f_w(x) = \text{sgn}(w^T x + b)$, can be reduced to the case w/o bias by letting $\widetilde{w} = [w; b]$ and $\tilde{x} = [x; 1]$

Source: [Y. Liang]()

Training linear classifiers

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \ldots, n\}$,
  $y_i \in \{-1,1\}$
- Hypothesis class: $f_w(x) = \text{sgn}(w^T x)$
- Loss: how about minimizing the number of mistakes on the training data?

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\text{sgn}(w^T x_i) \neq y_i]$$

- Difficult to optimize directly (NP-hard), so people resort to *surrogate loss functions*

Source: Y. Liang

Linear regression ("straw man" model)

- Find $f_w(x) = w^T x$ that minimizes l_2 *loss* or *mean squared error*

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2$$

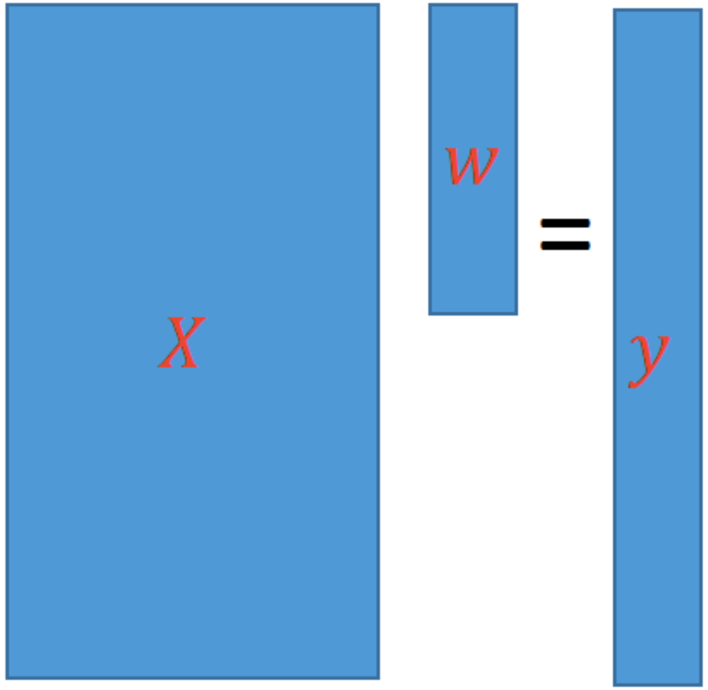
- Ignores the fact that $y \in \{-1,1\}$ but is easy to optimize

Source: Y. Liang

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \ldots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{n}\|Xw - Y\|_2^2$$



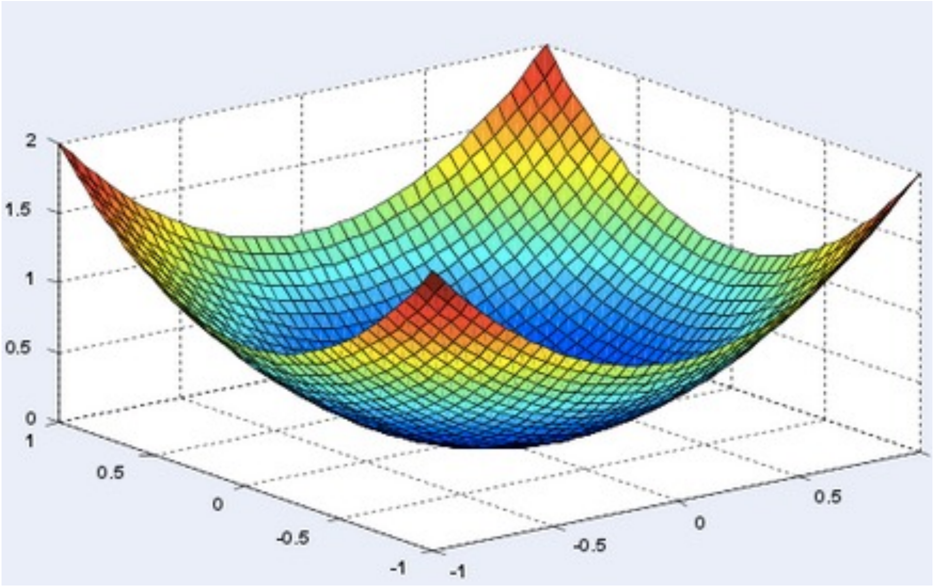
Source: [Y. Liang]

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \dots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{n}\|Xw - Y\|_2^2$$

- This is a *convex* function of the weights



Source: Y. Liang

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \ldots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{n}\|Xw - Y\|_2^2$$

- Find the *gradient* w.r.t. w:
  $\nabla_w \|Xw - Y\|_2^2$

Source: Y. Liang

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \dots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{n}\|Xw - Y\|_2^2$$

- Find the *gradient* w.r.t. w:

$$\begin{aligned} \nabla_w \|Xw - Y\|_2^2 &= \nabla_w[(Xw - Y)^T(Xw - Y)] \\ &= \nabla_w[w^T X^T X w - 2w^T X^T Y + Y^T Y] \\ &= 2X^T X w - 2X^T Y \end{aligned}$$

- Set gradient to zero to get the minimizer:

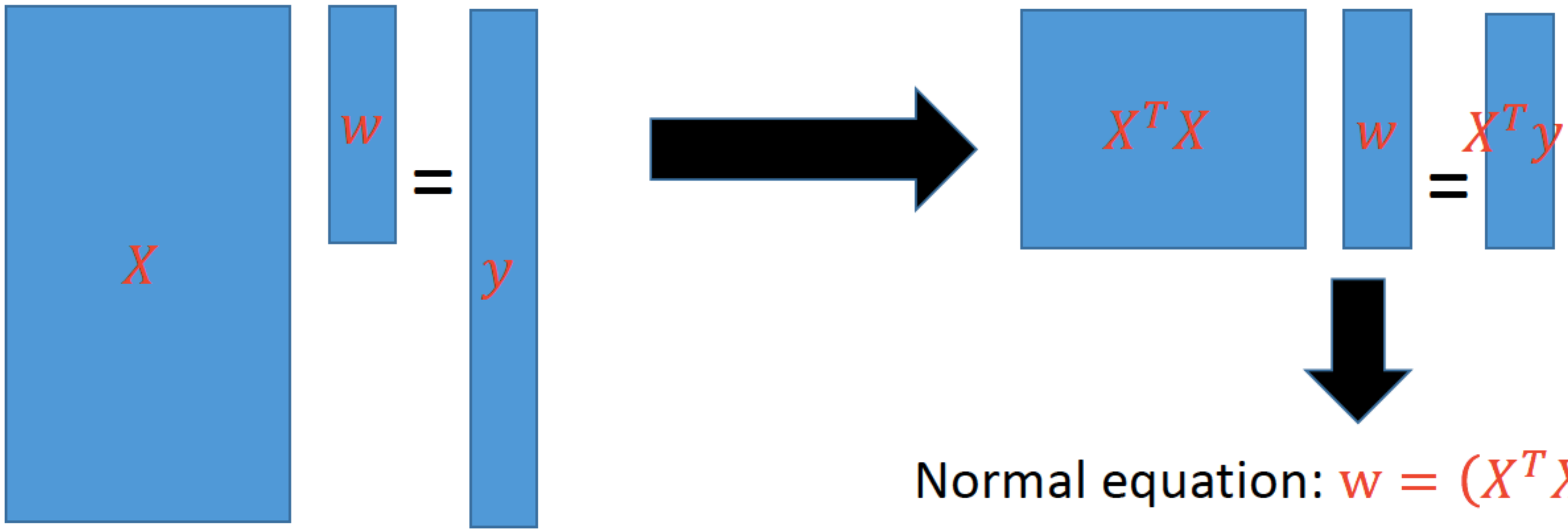
$$X^T X w = X^T Y$$

$$w = (X^T X)^{-1} X^T Y$$

Source: Y. Liang

Linear regression: Optimization

- Linear algebra view
  - If X is invertible, simply solve $Xw = Y$ and get $w = X^{-1}Y$
  - But typically X is a "tall" matrix so you need to find the *least squares solution* to an over-constrained system



Normal equation: $w = (X^T X)^{-1} X^T y$

Source: [Y. Liang]

Problem with linear regression

- In practice, very sensitive to outliers

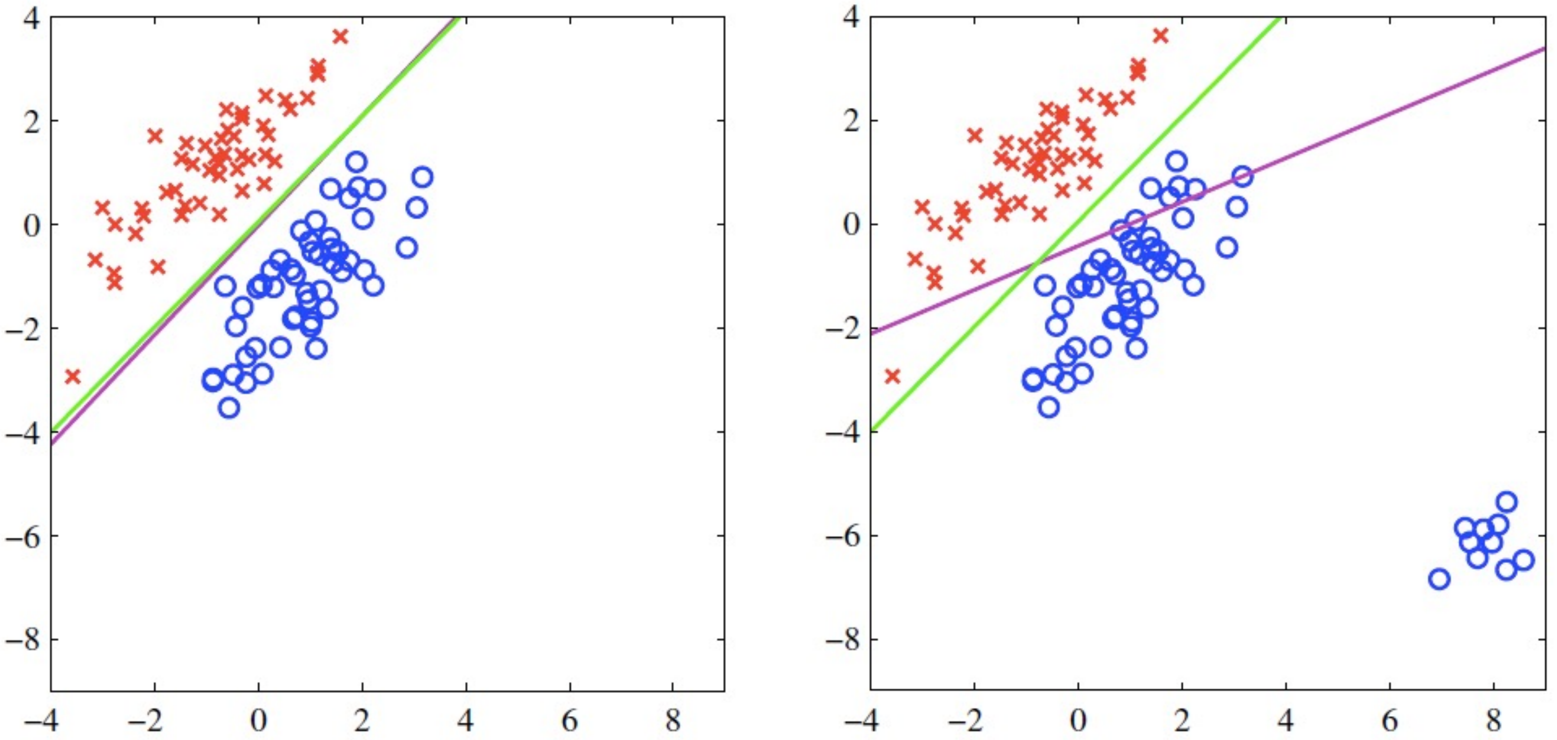
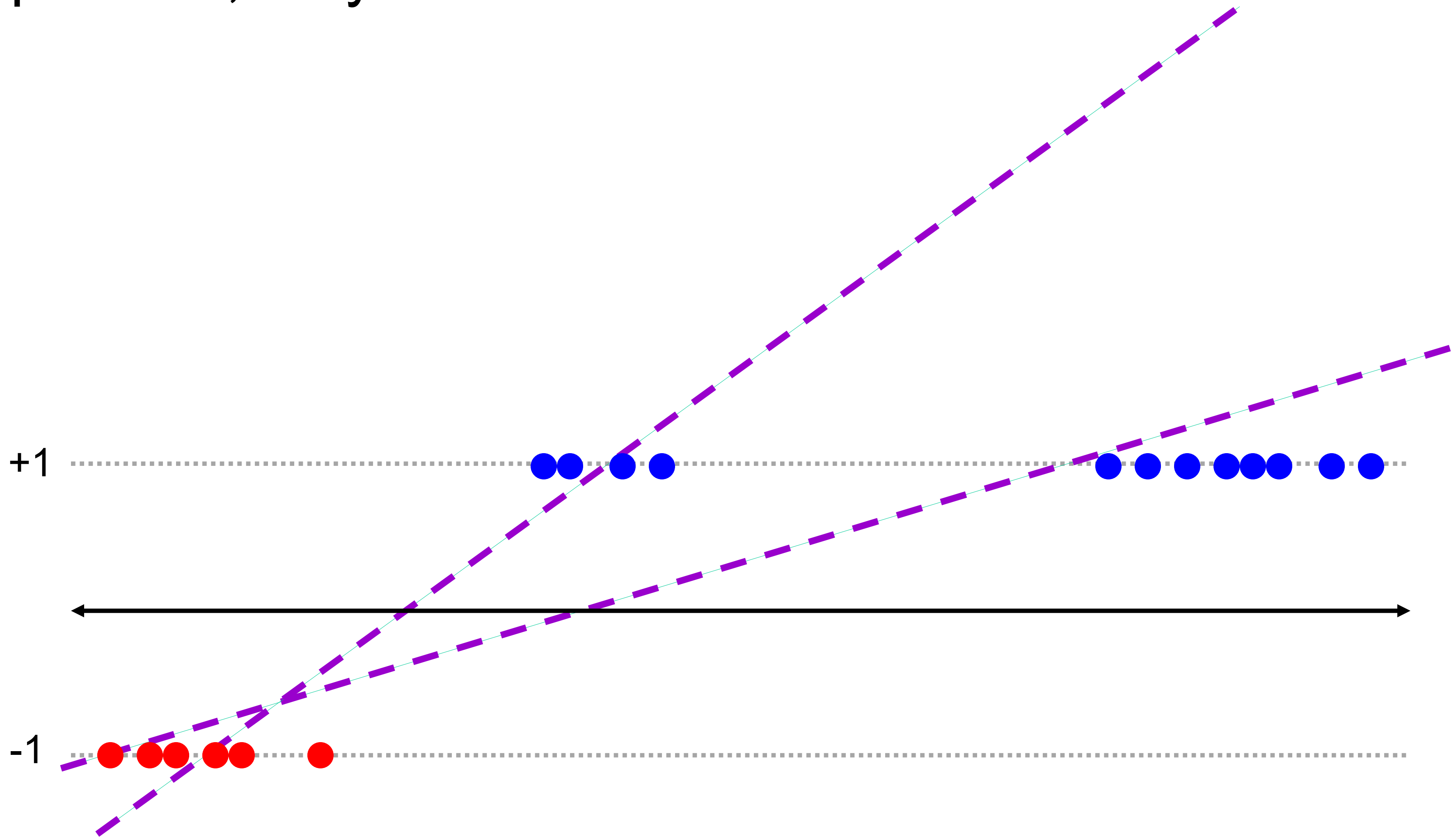


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

Figure from *Pattern Recognition and Machine Learning*, Bishop

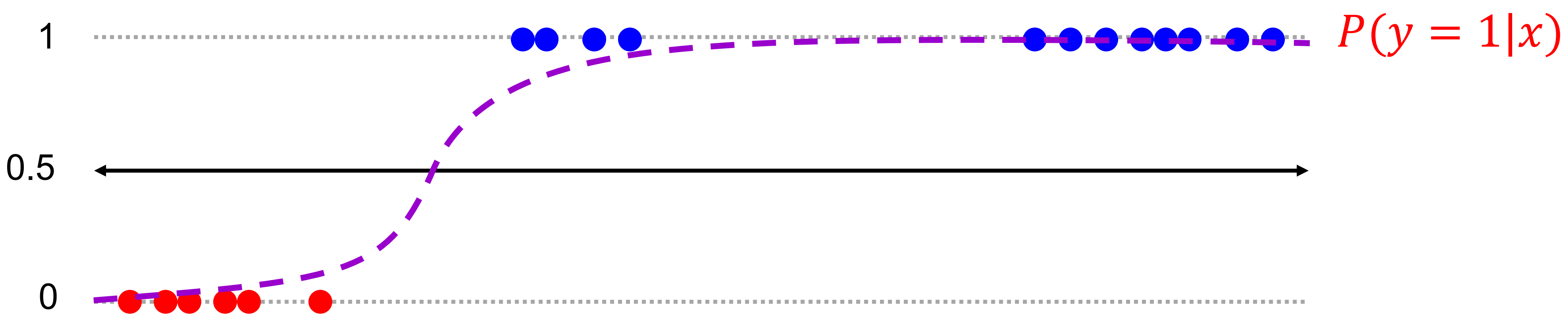
Problem with linear regression

- In practice, very sensitive to outliers



Next idea

- Instead of a linear function, how about we fit a function representing the *confidence* of the classifier?



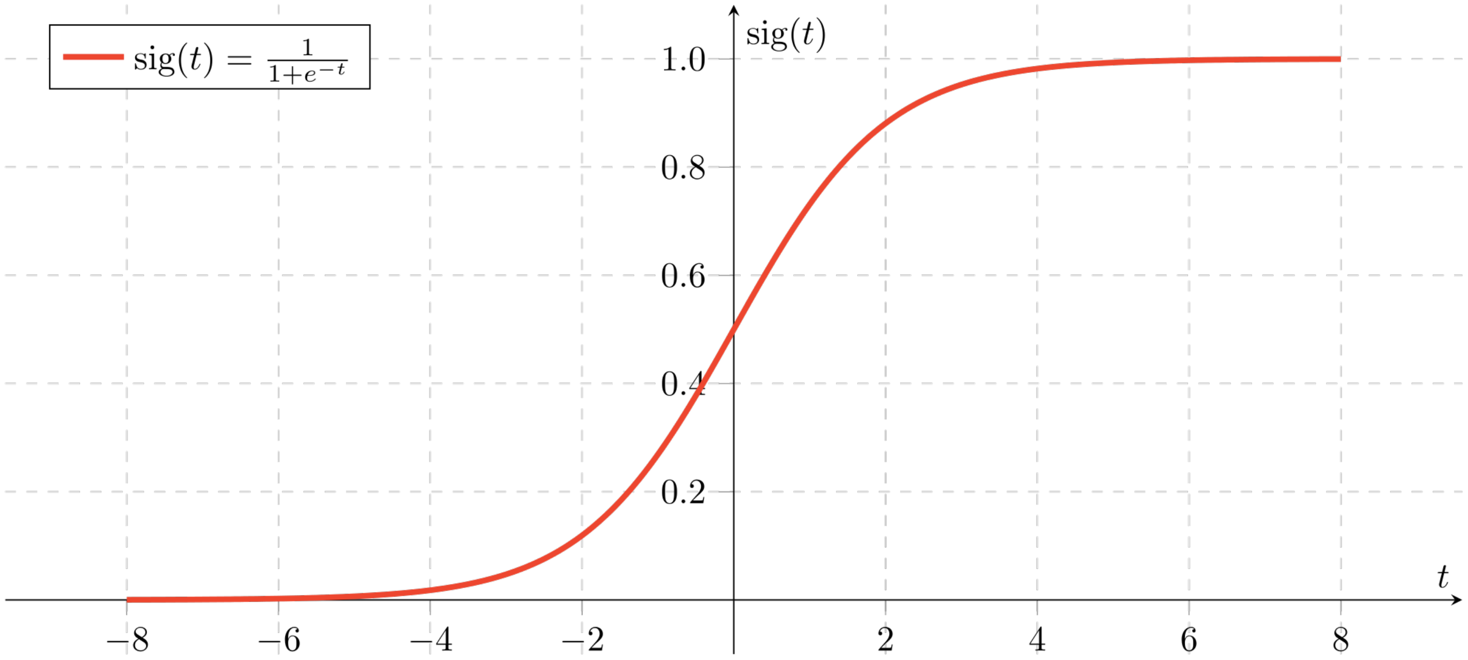
Linear classifiers: Outline

- Example classification models: nearest neighbor, linear
- Empirical loss minimization
- Linear classification models
  1. Linear regression (least squares)
  2. Logistic regression

Logistic regression

- Let's learn a probabilistic classifier estimating the probability of the input x having a positive label, given by putting a *sigmoid function* around the linear response $w^T x$:

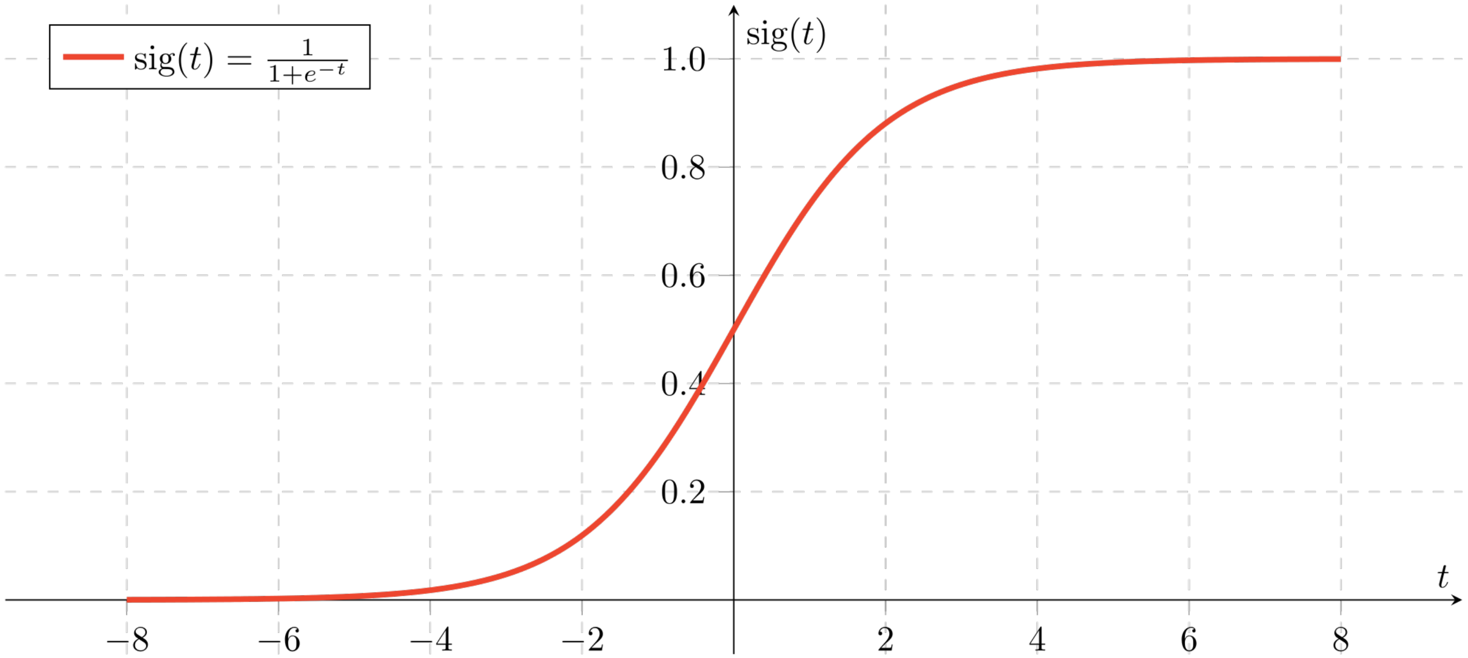
$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$



Sigmoid: Properties

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

- What is the range?
- What is $\sigma(0)$?
- What is $P_w(y = -1|x)$?



Sigmoid: Properties

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

- What is the range?
- What is $\sigma(0)$?
- What is $P_w(y = -1|x)$?

$$P_w(y = -1|x) = 1 - P_w(y = 1|x) = 1 - \sigma(w^T x)$$

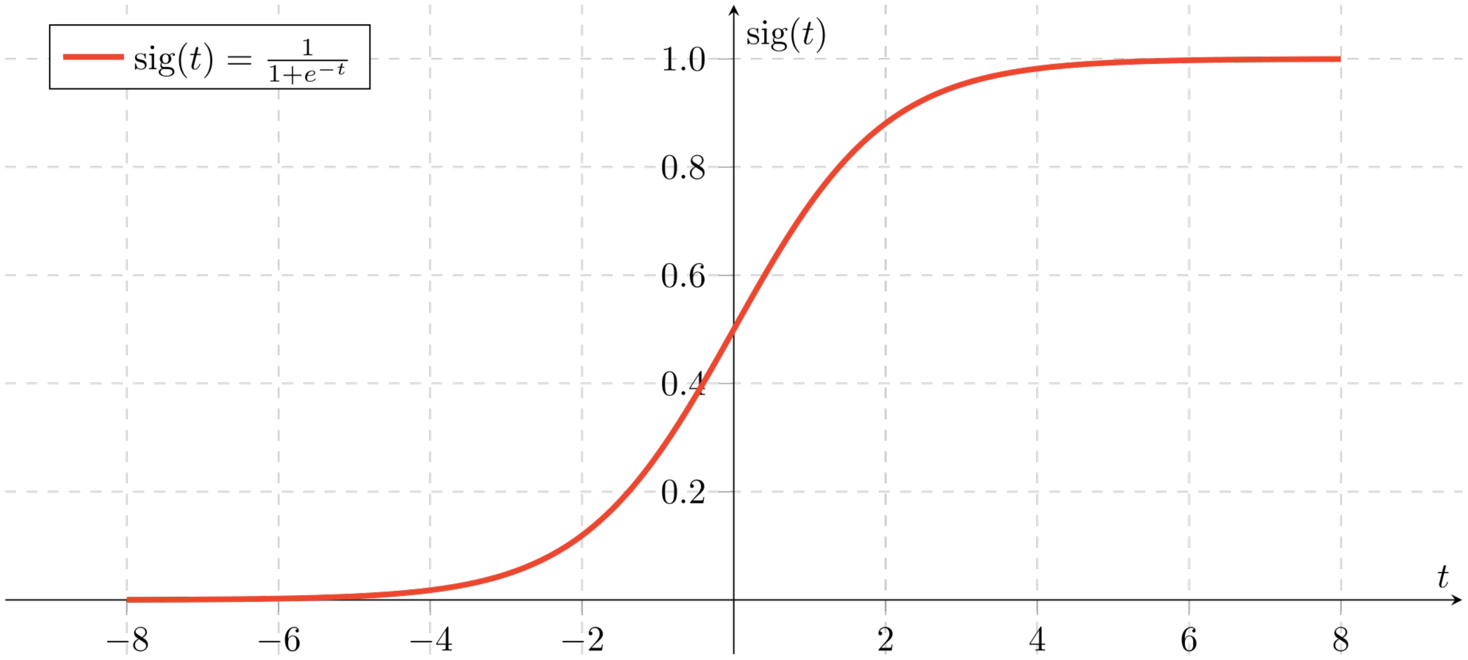
$$= \frac{1+\exp(-w^T x)-1}{1+\exp(-w^T x)} = \frac{\exp(-w^T x)}{1+\exp(-w^T x)} = \frac{1}{\exp(w^T x)+1}$$

$$= \sigma(-w^T x)$$

Sigmoid: Properties

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

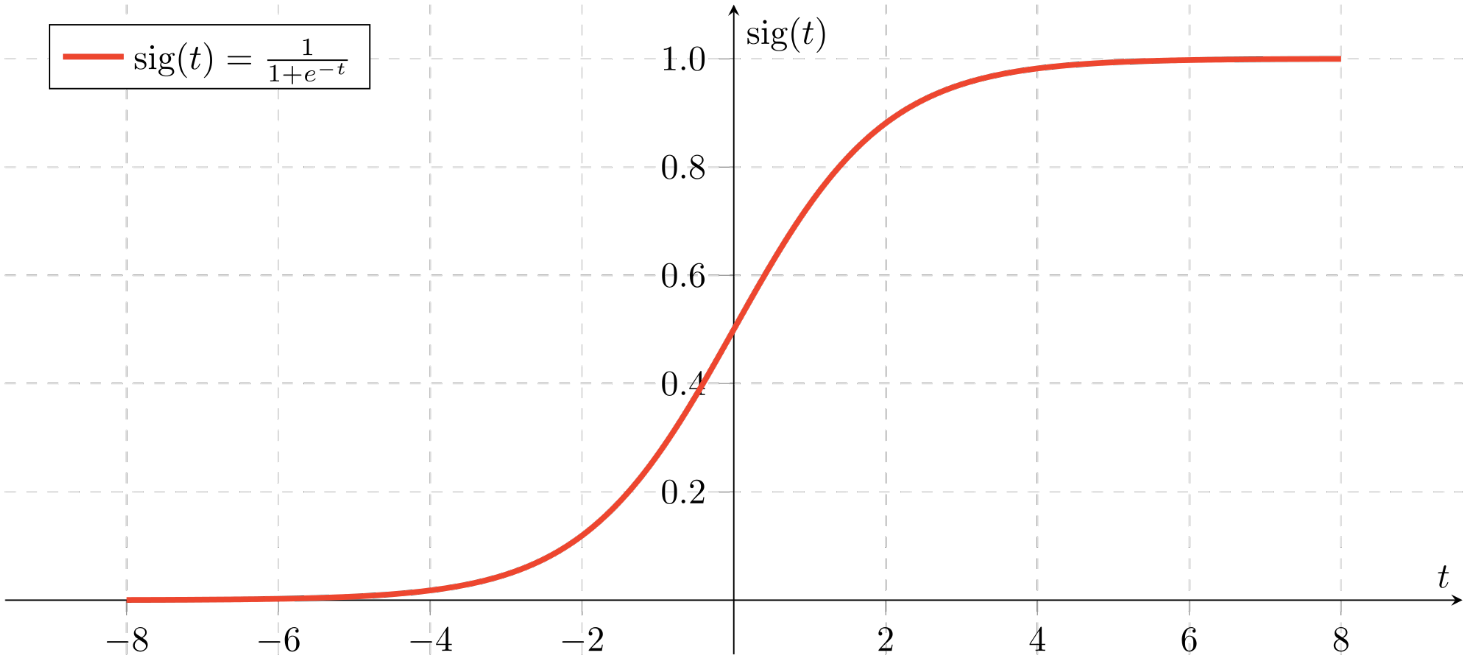
- Sigmoid is *symmetric* in the following sense: $1 - \sigma(t) = \sigma(-t)$



Sigmoid: Properties

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

- What happens if we scale w by a constant?



Sigmoid: Properties

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

- What happens if we scale w by a constant?

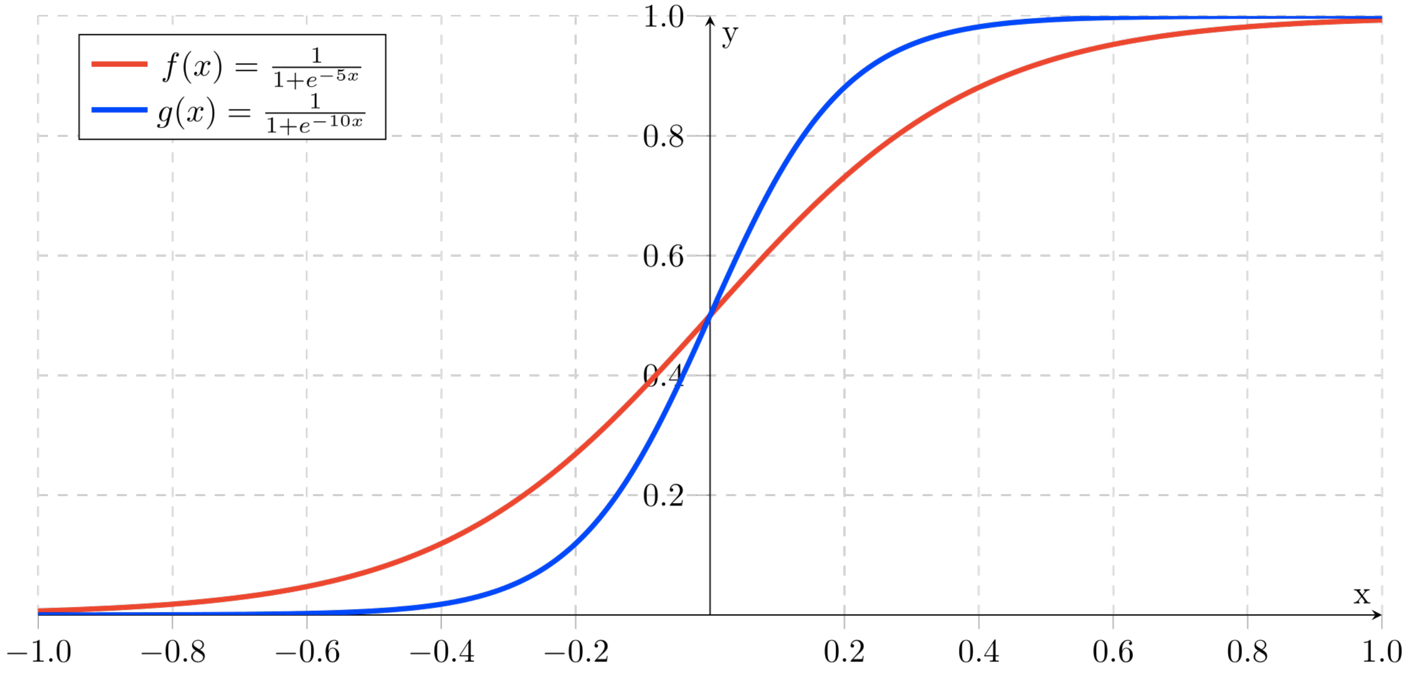


Image source

Logistic loss

- Given: $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1,1\}$
- Maximum (conditional) likelihood estimate: find w that minimizes

$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log P_w(y_i|x_i)$$

$$l(w, x_i, y_i) = -\log P_w(y_i|x_i)$$

- If $y_i = 1$:

$$P_w(y_i|x_i) = \sigma(w^T x_i)$$

- If $y_i = -1$:

$$P_w(y_i|x_i) = 1 - \sigma(w^T x_i) = \sigma(-w^T x_i)$$

- Thus,

$$l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)$$

Logistic loss

$$l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)$$

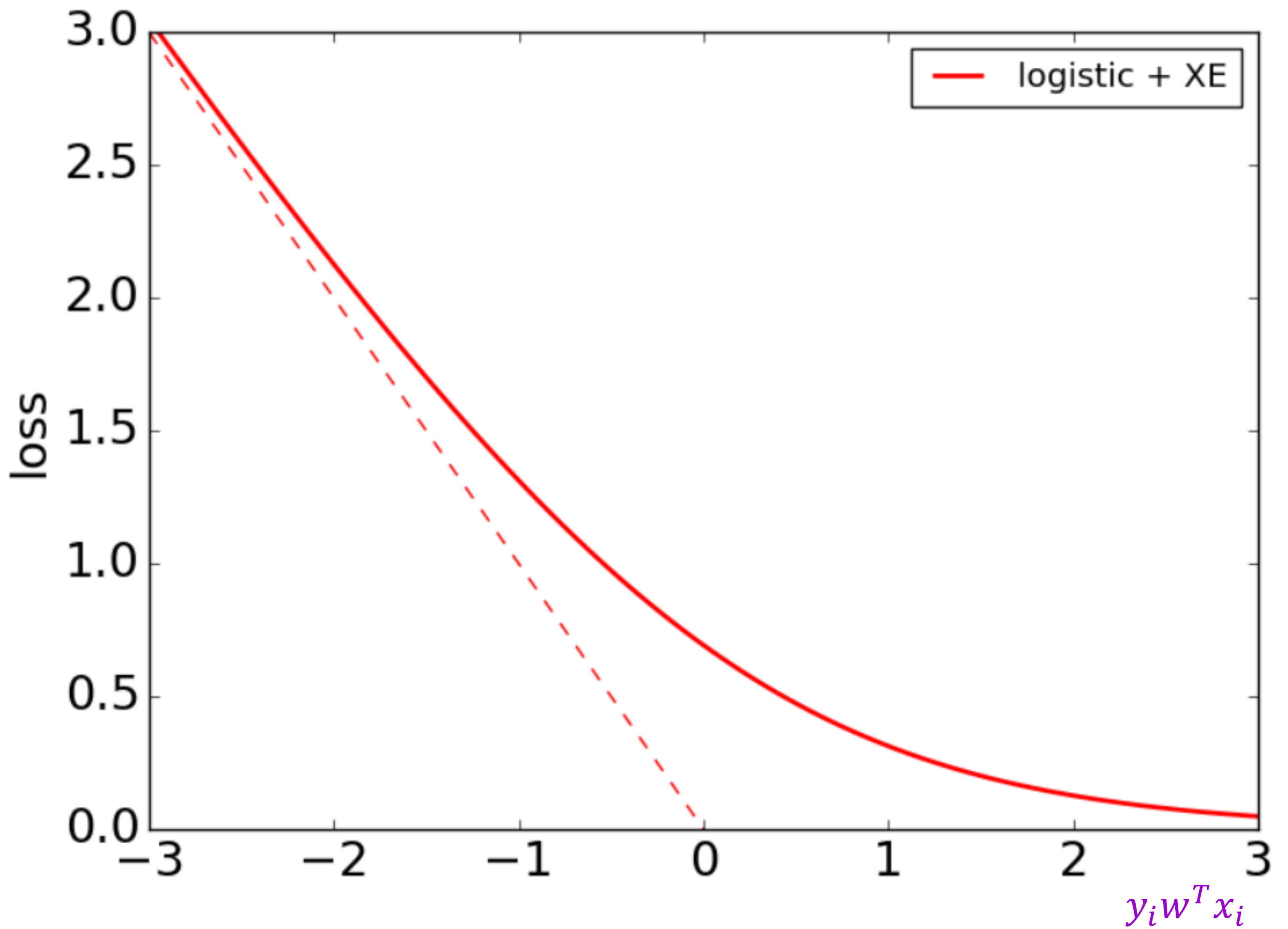


Figure source

Logistic loss: Optimization

- Given: $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1,1\}$
- Find w that minimizes

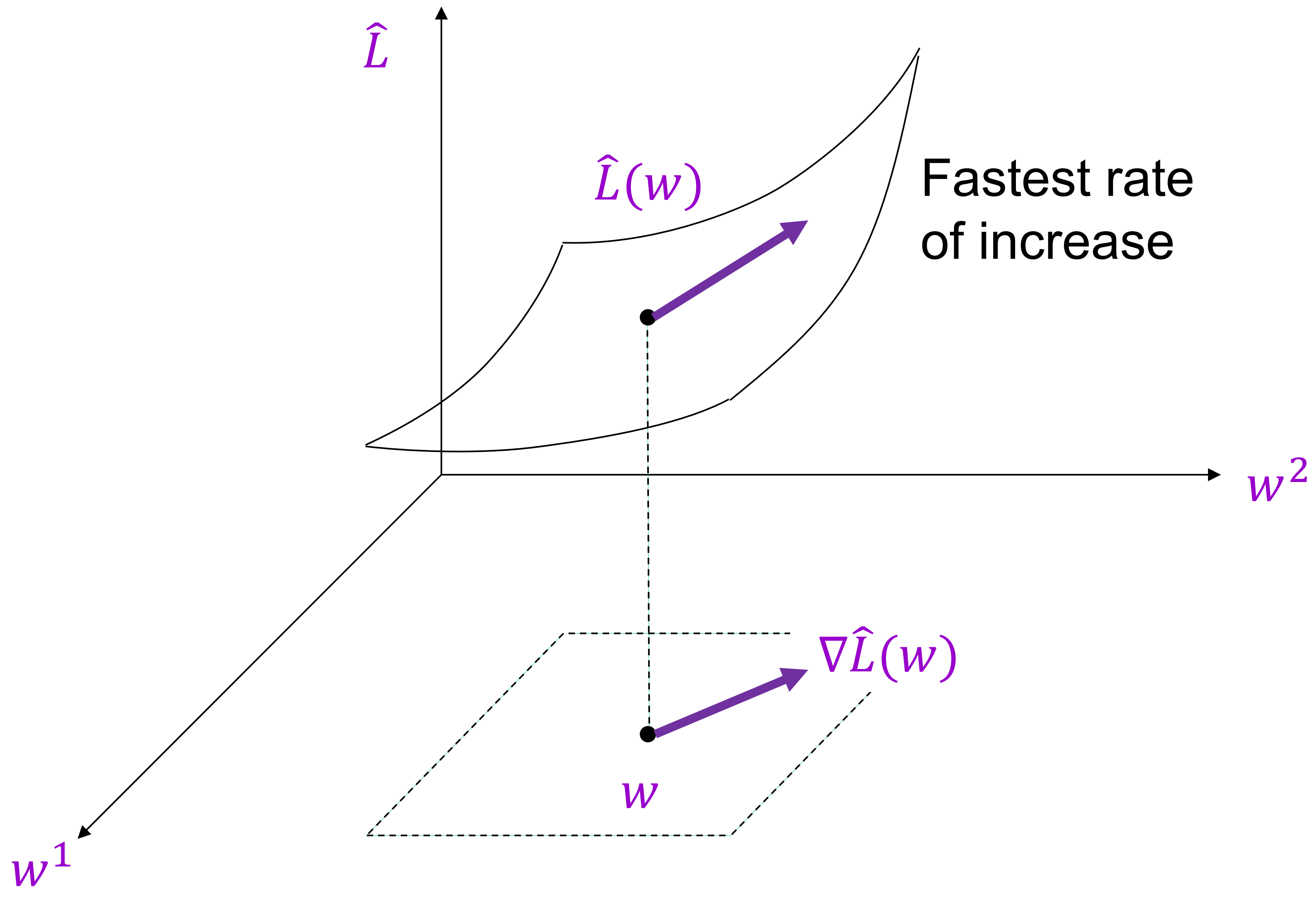
$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log P_w(y_i|x_i)$$

- There is no closed-form expression for the minimum and we need to use *gradient descent* to find it

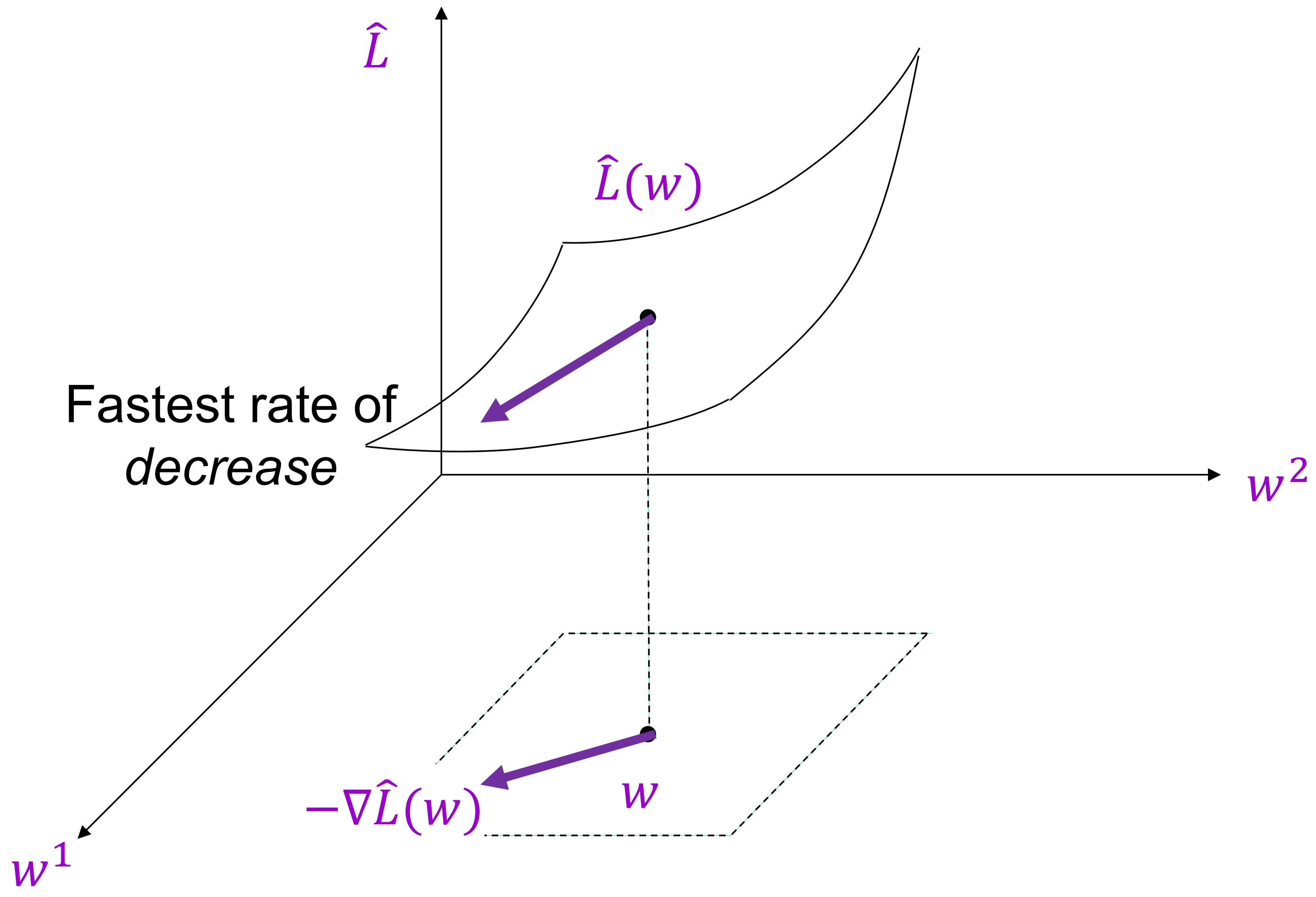
Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- Repeat until convergence:
  - Find $\nabla\hat{L}(w)$, the *gradient* of the loss w.r.t. w
  - Take a small step in the *opposite* direction: $w \leftarrow w - \eta\, \nabla\hat{L}(w)$

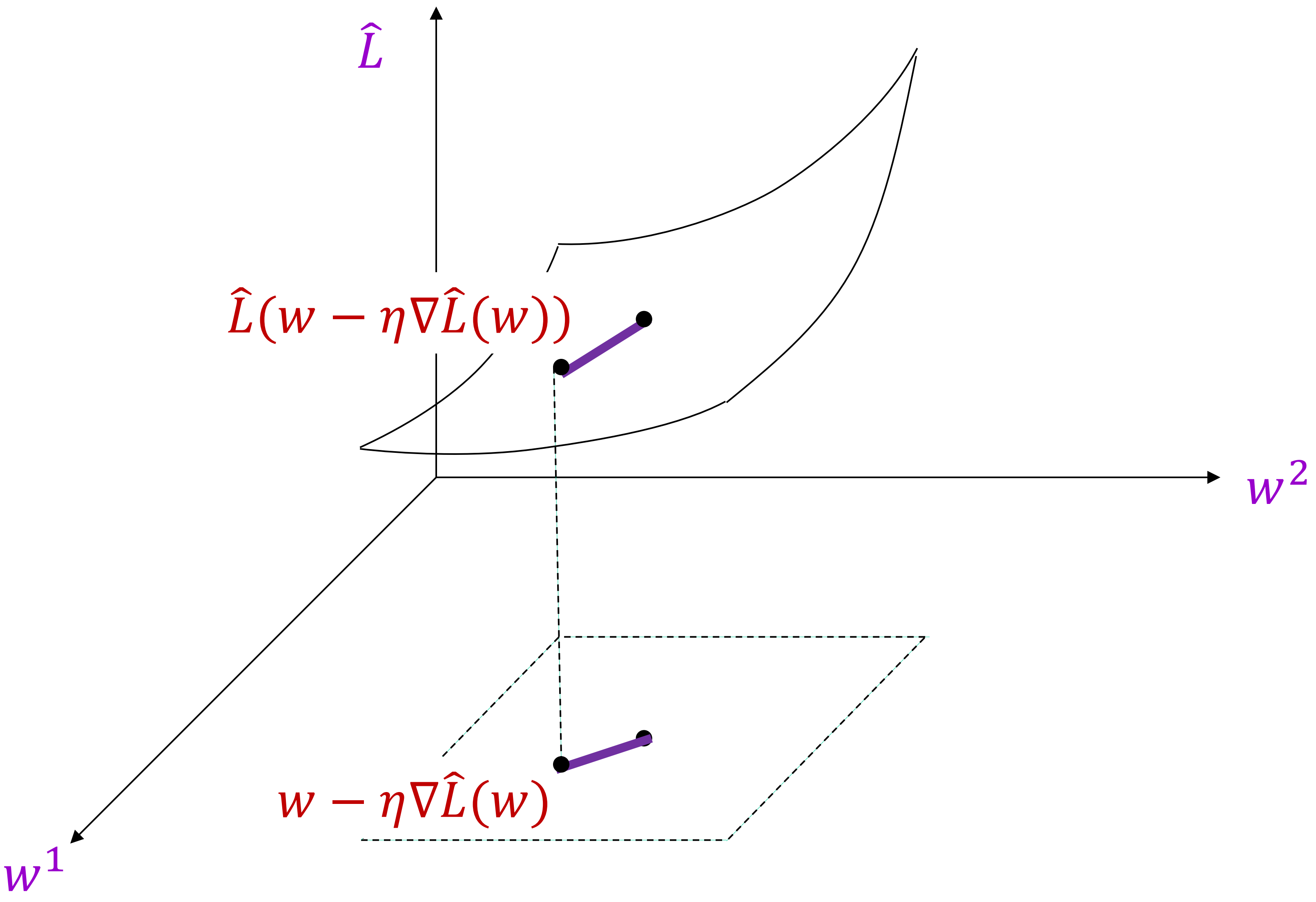
The gradient vector



The gradient vector

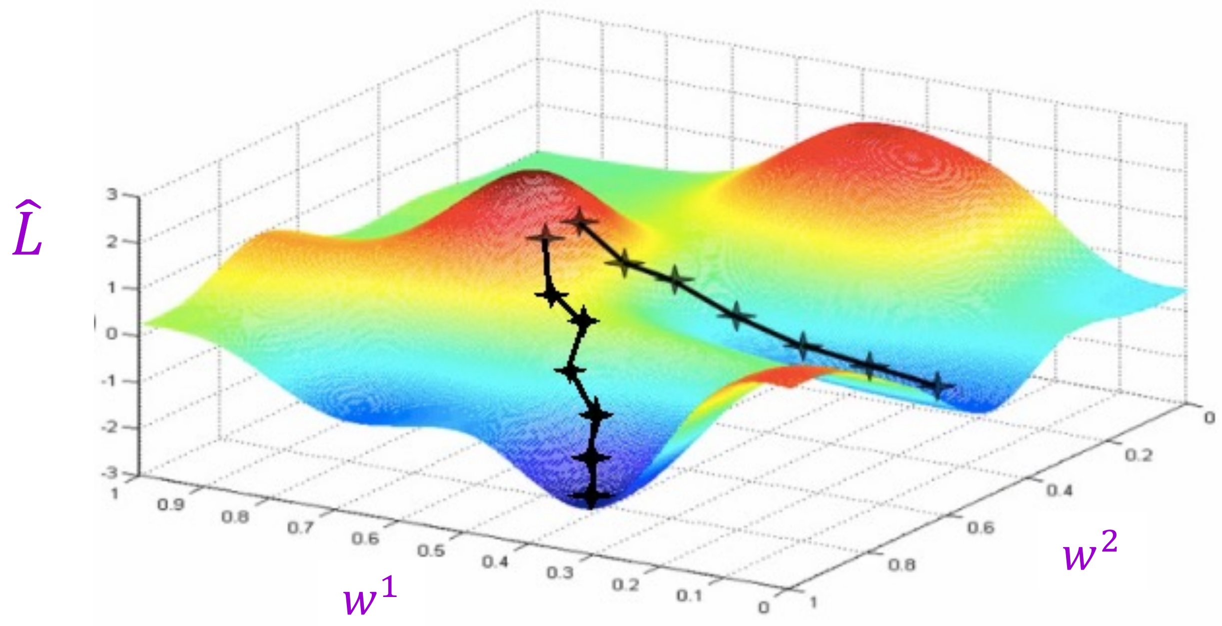


Gradient descent



Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- Repeat until convergence:
  - Find $\nabla\hat{L}(w)$, the *gradient* of the loss w.r.t. w
  - Take a small step in the *opposite* direction: $w \leftarrow w - \eta\, \nabla\hat{L}(w)$



Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- Repeat until convergence:
  - Find $\nabla\hat{L}(w)$, the *gradient* of the loss w.r.t. w
  - Take a small step in the *opposite* direction: $w \leftarrow w - \eta \nabla\hat{L}(w)$
  - η is the step size or *learning rate*

Full batch gradient descent

- Since $\hat{L}(w) = \frac{1}{n}\sum_{i=1}^{n} l(w, x_i, y_i)$, we have

$$\nabla\hat{L}(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla l(w, x_i, y_i)$$

- For a single parameter update, need to cycle through the entire training set!

Stochastic gradient descent (SGD)

- At each iteration, take a *single data point* (x_i, y_i) and perform a parameter update using $\nabla l(w, x_i, y_i)$, the gradient of the loss for that point:

$$w \leftarrow w - \eta\, \nabla l(w, x_i, y_i)$$

- This is called an *online* or *stochastic* update
- In practice, *mini-batch SGD* is typically used:
  - Group data into mini-batches of size b
    - Compute gradient of the loss for the mini-batch $(x_1, y_1), \ldots, (x_b, y_b)$:

$$\nabla \hat{L} = \frac{1}{b} \sum_{i=1}^{b} \nabla l(w, x_i, y_i)$$

    - Update parameters: $w \leftarrow w - \eta \nabla \hat{L}$

SGD for logistic regression

$$l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)$$

- Let's find the gradient:

$$\nabla l(w, x_i, y_i) = -\nabla_w \log \sigma(y_i w^T x_i)$$
$$= -\frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)}$$

- Derivative of log:

$$\left[\log\left(g(a)\right)\right]' = \frac{g'(a)}{g(a)}$$

SGD for logistic regression

$$l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)$$

- Let's find the gradient:

$$\nabla l(w, x_i, y_i) = -\nabla_w \log \sigma(y_i w^T x_i)$$

$$= -\frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)}$$

$$= -\frac{\sigma(y_i w^T x_i)\sigma(-y_i w^T x_i) y_i x_i}{\sigma(y_i w^T x_i)}$$

Derivative of sigmoid:

$$\sigma'(a) = \sigma(a)(1 - \sigma(a)) = \sigma(a)\sigma(-a)$$

SGD for logistic regression

$$l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)$$

- Let's find the gradient:

$$\nabla l(w, x_i, y_i) = -\nabla_w \log \sigma(y_i w^T x_i)$$

$$= -\frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)}$$

$$= -\frac{\sigma(y_i w^T x_i)\sigma(-y_i w^T x_i) y_i x_i}{\sigma(y_i w^T x_i)}$$

- We also used the *chain rule*: $[g_2(g_1(a))]' = g_2'(g_1(a))g_1'(a)$

SGD for logistic regression

$$l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)$$

- Let's find the gradient:

$$\nabla l(w, x_i, y_i) = -\nabla_w \log \sigma(y_i w^T x_i)$$
$$= -\frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)}$$
$$= -\frac{\sigma(y_i w^T x_i)\sigma(-y_i w^T x_i) y_i x_i}{\sigma(y_i w^T x_i)}$$
$$= -\sigma(-y_i w^T x_i) y_i x_i$$

- SGD update:

$$w \leftarrow w + \eta\, \sigma(-y_i w^T x_i) y_i x_i$$

SGD for logistic regression

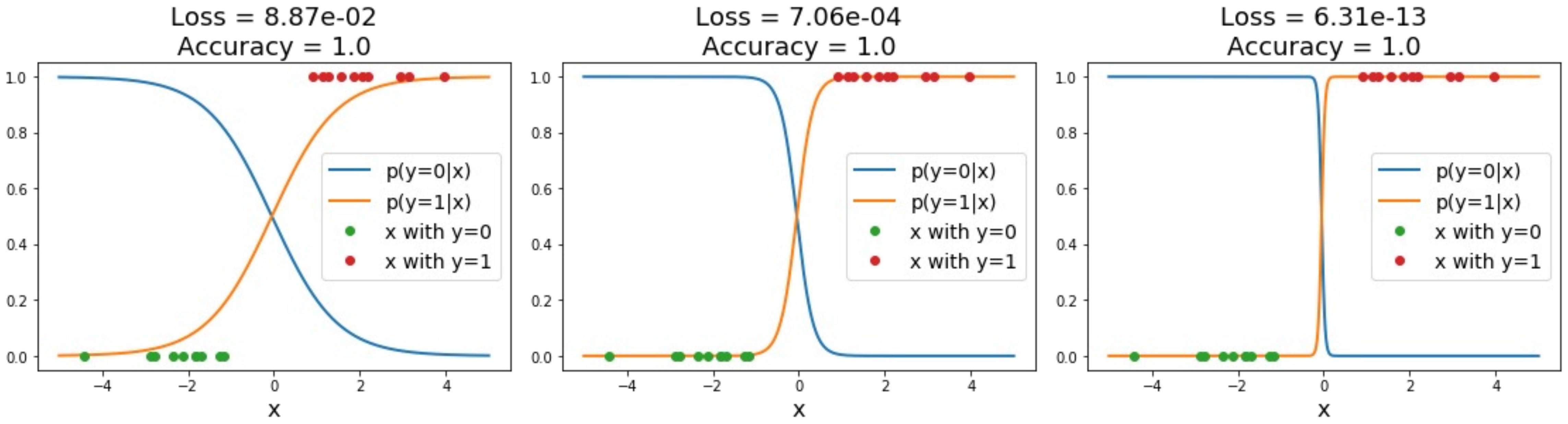
- Let's take a closer look at the SGD update:

$$w \leftarrow w + \eta\, \sigma(-y_i w^T x_i) y_i x_i$$

- What happens if x_i is *incorrectly,* but confidently, classified?
  - The update rule approaches $w \leftarrow w + \eta\, y_i x_i$
- What happens if x_i is *correctly,* and confidently, classified?
  - The update approaches zero (but never actually reaches zero)

SGD for logistic regression

- Logistic regression *does not converge* for linearly separable data!
  - Scaling w by ever larger constants makes the classifier more confident and keeps increasing the likelihood of the data



Source: [J. Johnson]

Linear classifiers: Outline

- Example classification models: nearest neighbor, linear
- Empirical loss minimization
- Linear classification models
  1. Linear regression (least squares)
  2. Logistic regression
  3. Perceptron loss

Recall: The shape of logistic loss

$$l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)$$

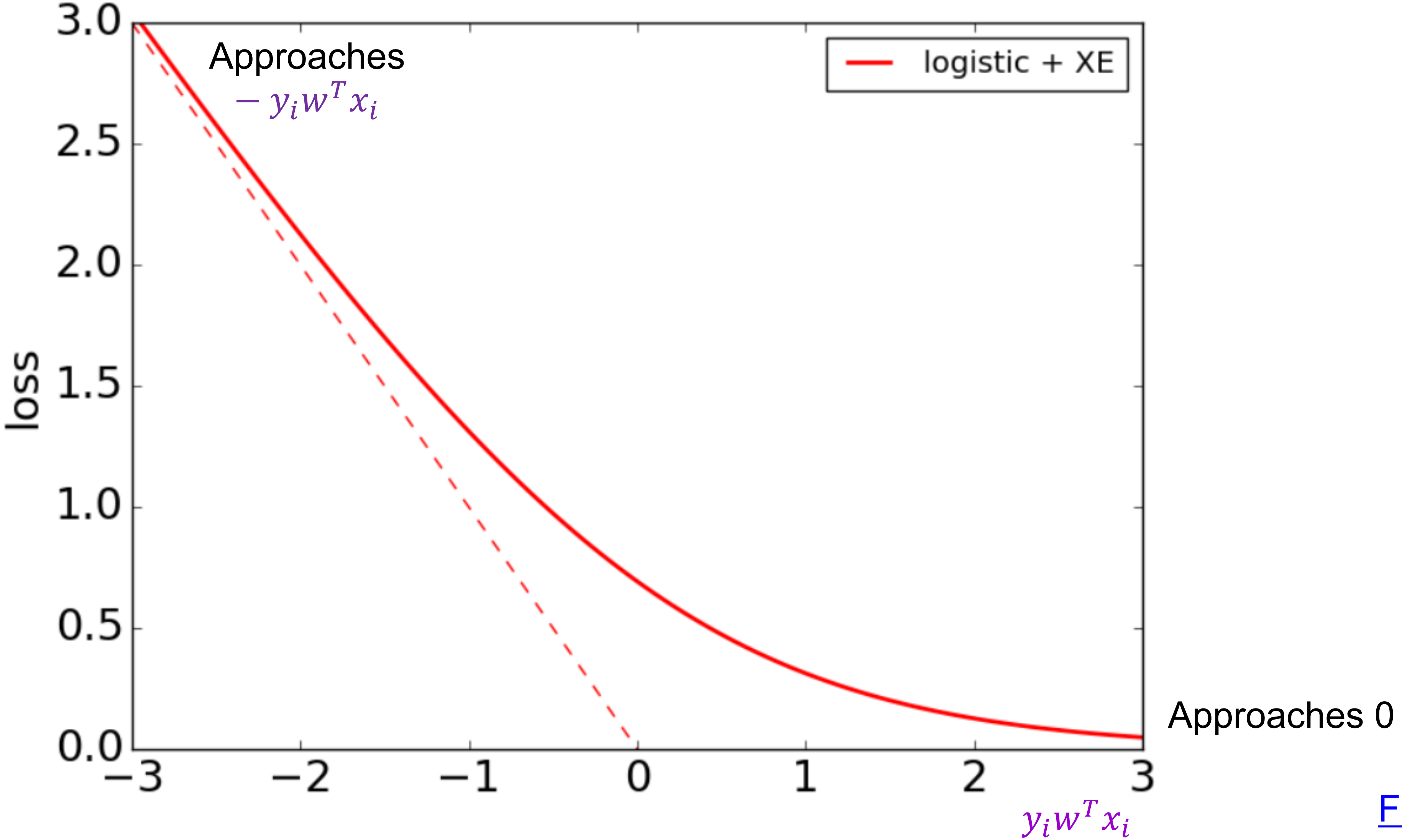
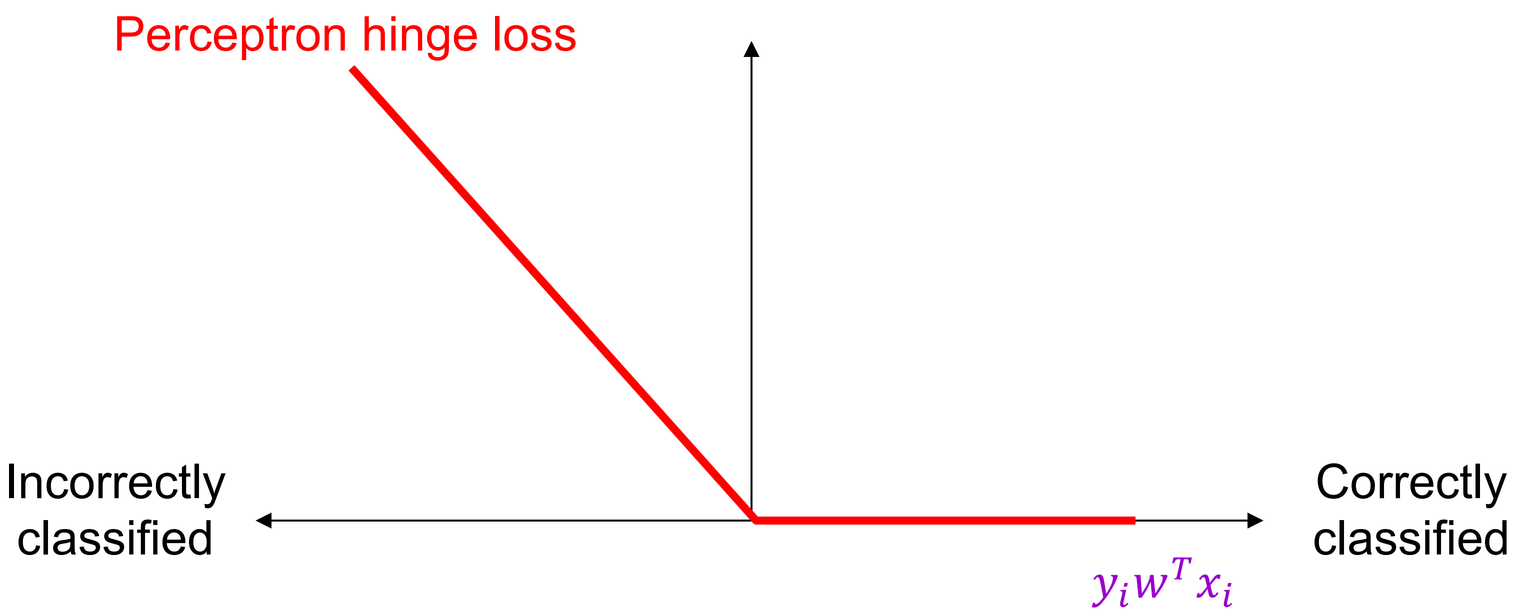


Figure source

Perceptron

- Let's define the *perceptron hinge loss*:

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$



Perceptron

- Let's define the *perceptron hinge loss*:

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

- Training: find w that minimizes

$$\hat{L}(w) = \frac{1}{n}\sum_{i=1}^{n} l(w, x_i, y_i) = \frac{1}{n}\sum_{i=1}^{n} \max(0, -y_i w^T x_i)$$

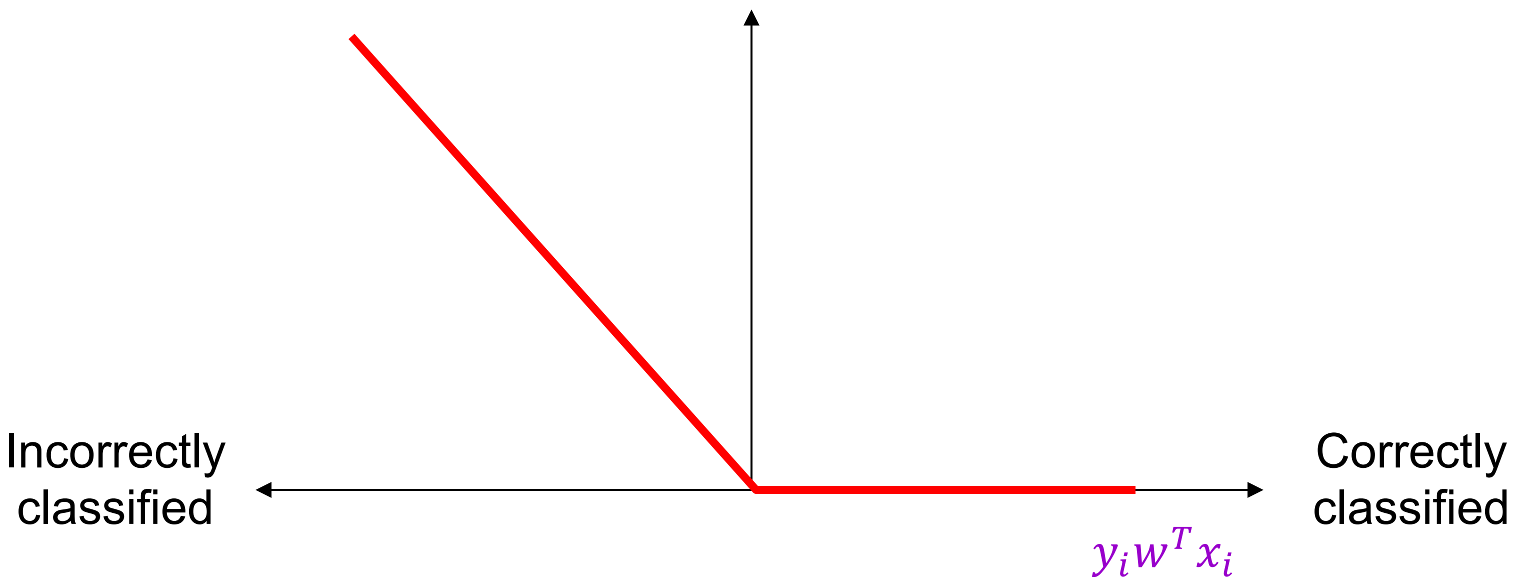
- Once again, there is no closed-form solution, so let's go straight to SGD

Deriving the perceptron update

- Let's differentiate the perceptron hinge loss:

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

(Strictly speaking, this loss is not differentiable, so we need to find a *sub-gradient*: A vector $g \in R^n$ is a sub-gradient of $f: R^n \to R$ at x if for all z, $f(z) \geq f(x) + g^T(z - x)$.)



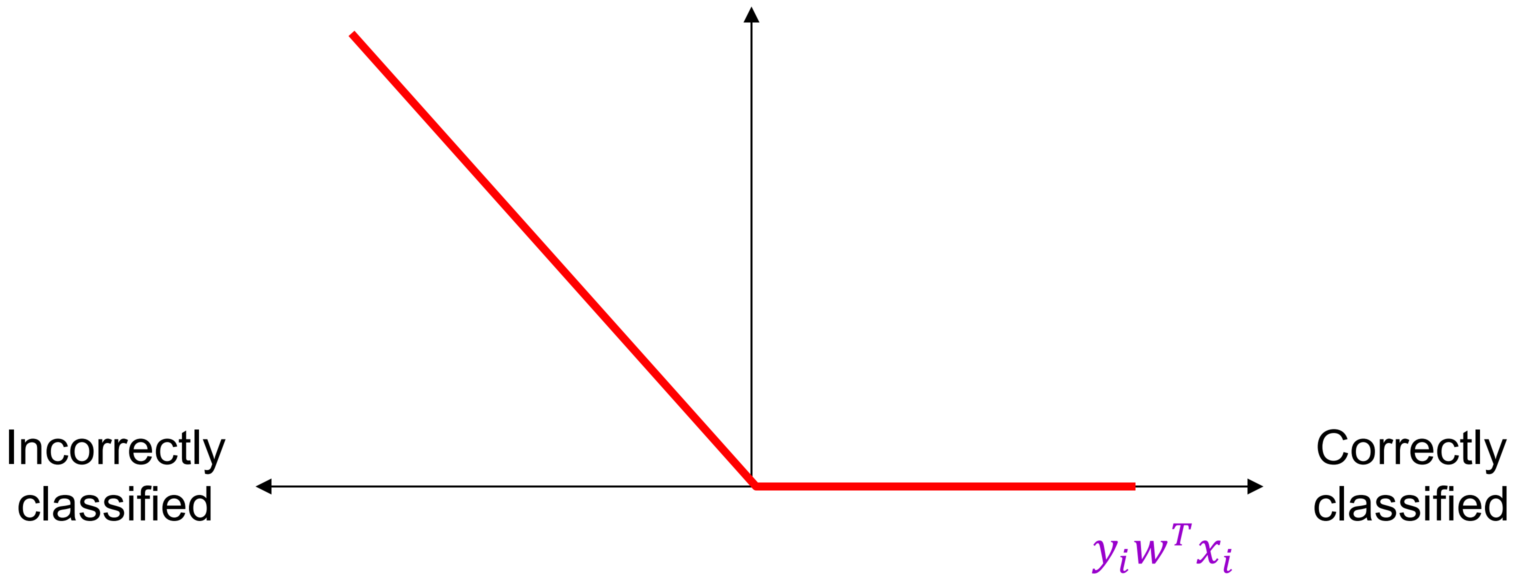
Deriving the perceptron update

- Let's differentiate the perceptron hinge loss:

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

$$\nabla l(w, x_i, y_i) = -\mathbb{I}[y_i w^T x_i < 0] y_i x_i$$

$$\frac{d}{da}\max(0, a) =$$



Deriving the perceptron update

- Let's differentiate the perceptron hinge loss:

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

$$\nabla l(w, x_i, y_i) = -\mathbb{I}[y_i w^T x_i < 0] y_i x_i$$

- We also used the chain rule: $\left[g_2\left(g_1(a)\right)\right]' = g_2'\left(g_1(a)\right) g_1'(a)$

Deriving the perceptron update

- Let's differentiate the perceptron hinge loss:

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

$$\nabla l(w, x_i, y_i) = -\mathbb{I}[y_i w^T x_i < 0] y_i x_i$$

- Corresponding SGD update ($w \leftarrow w - \eta \nabla l(w, x_i, y_i)$):

$$w \leftarrow w + \eta \, \mathbb{I}[y_i w^T x_i < 0] y_i x_i$$

  - If x_i is correctly classified: do nothing
  - If x_i is incorrectly classified: $w \leftarrow w + \eta \, y_i x_i$

Perceptron training algorithm

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training example (x_i, y_i):
- If current prediction $\text{sgn}(w^T x_i)$ does not match y_i then update weights:

$$w \leftarrow w + \eta\, y_i x_i$$

where η is a *learning rate* that should decay slowly* over time

Understanding the perceptron update rule

- **Perceptron update rule:** If $y_i \neq \mathrm{sgn}(w^T x_i)$ then update weights:

$$w \leftarrow w + \eta\, y_i x_i$$

- The raw response of the classifier changes to

$$w^T x_i + \eta\, y_i \|x_i\|^2$$

- How does the response change if $y_i = 1$?
  - The response $w^T x_i$ is initially *negative* and will be *increased*

- How does the response change if $y_i = -1$?
  - The response $w^T x_i$ is initially *positive* and will be *decreased*

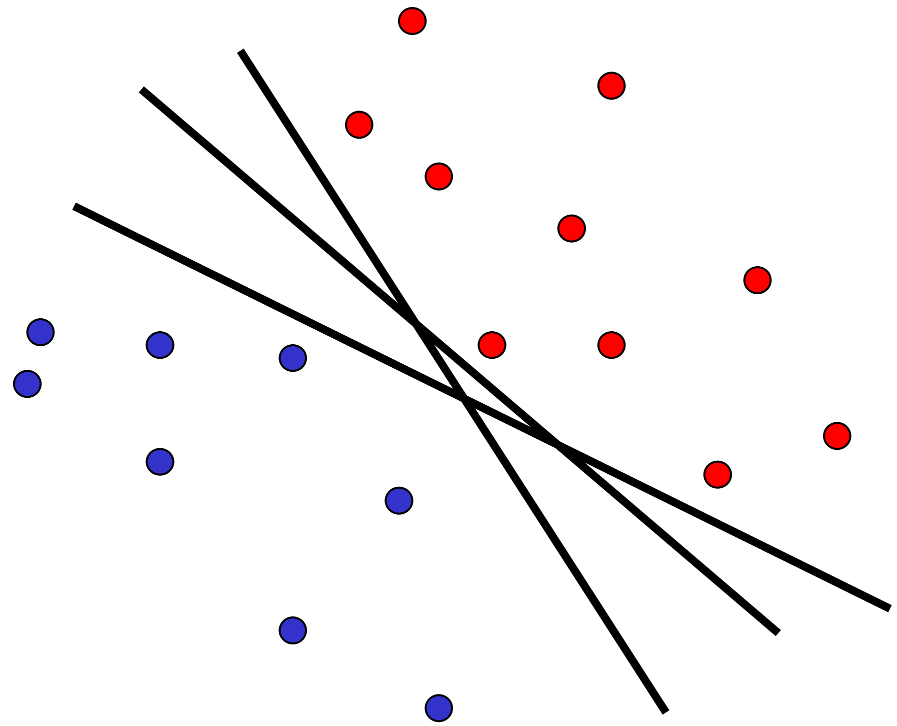
Linear classifiers: Outline

- Example classification models: nearest neighbor, linear
- Empirical loss minimization
- Linear classification models
  1. Linear regression (least squares)
  2. Logistic regression
  3. Perceptron loss
  4. Support vector machine (SVM) loss

Support vector machines

- When the data is linearly separable, which of the many possible solutions should we prefer?

  - **Perceptron training algorithm:**
    no special criterion, solution depends on initialization

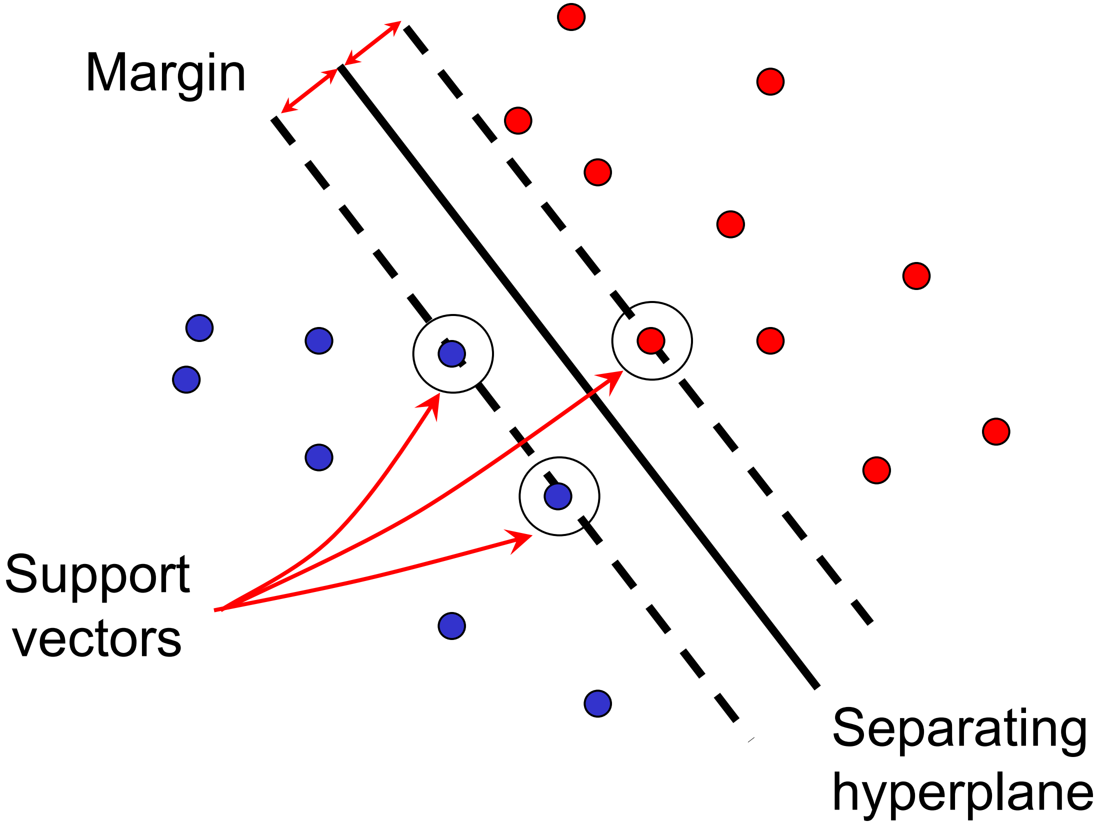


Support vector machines

- When the data is linearly separable, which of the many possible solutions should we prefer?

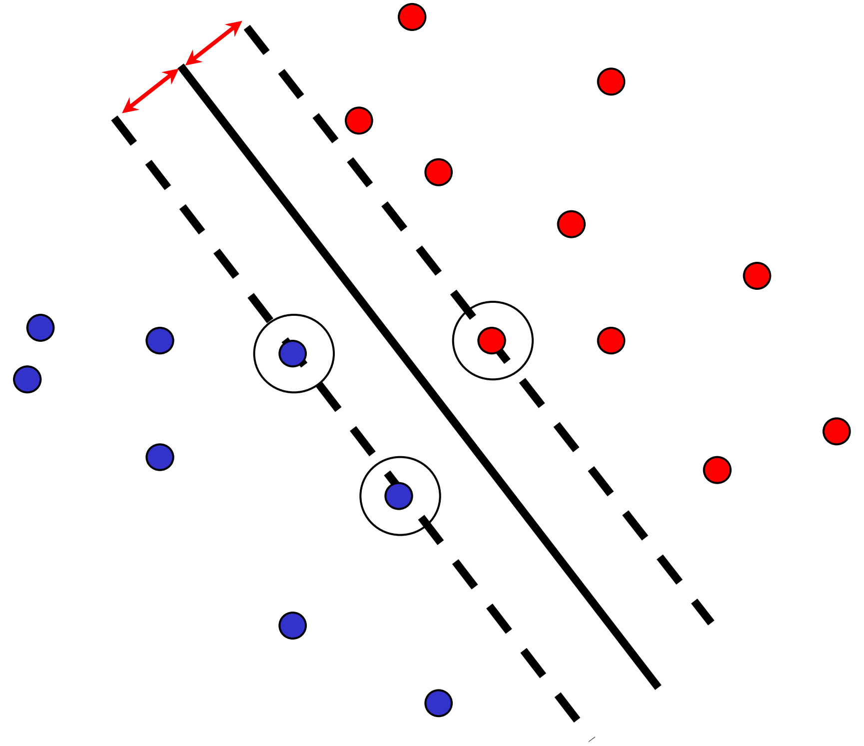
  - **Perceptron training algorithm:** no special criterion, solution depends on initialization

  - **SVM criterion:** maximize the *margin*, or distance between the hyperplane and the closest training example



Finding the maximum margin hyperplane

- We want to maximize the margin, or distance between the hyperplane $w^T x = 0$ and the closest training example x_0
- This distance is given by $\frac{|w^T x_0|}{\|w\|}$ (for derivation see, e.g., here)
- Assuming the data is linearly separable, we can fix the scale of w so that $y_i w^T x_i = 1$ for support vectors and $y_i w^T x_i \geq 1$ for all other points
- Then the margin is given by $\frac{1}{\|w\|}$



Finding the maximum margin hyperplane

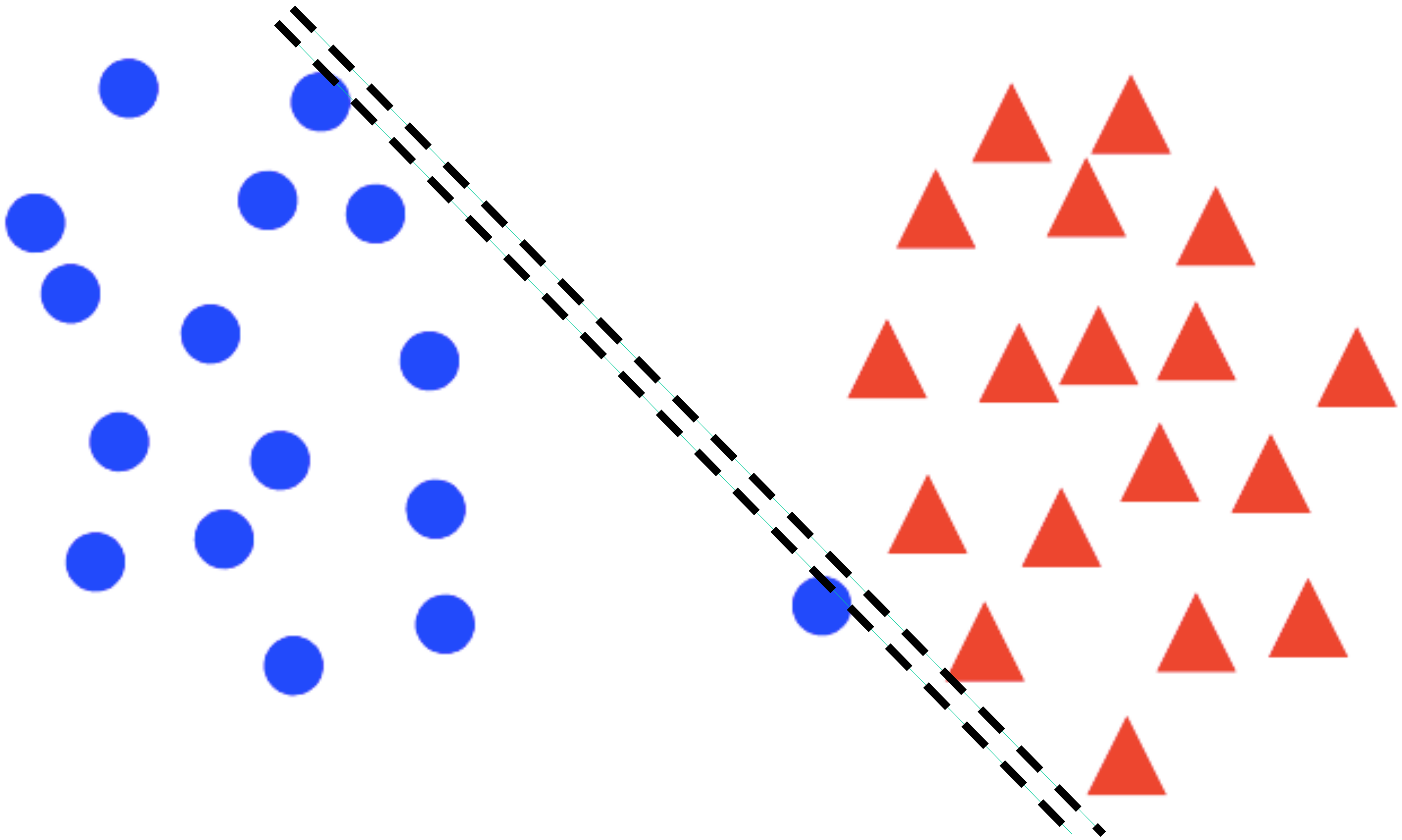
- We want to maximize margin $\frac{1}{\|w\|}$ while correctly classifying all training data: $y_i w^T x_i \geq 1$
- Equivalent problem:

$$\min_w \frac{1}{2}\|w\|^2 \qquad \text{s.t.} \qquad y_i w^T x_i \geq 1 \ \ \forall i$$

- This is a quadratic objective with linear constraints: convex optimization problem, global optimum can be found using well-studied methods

“Soft margin” formulation

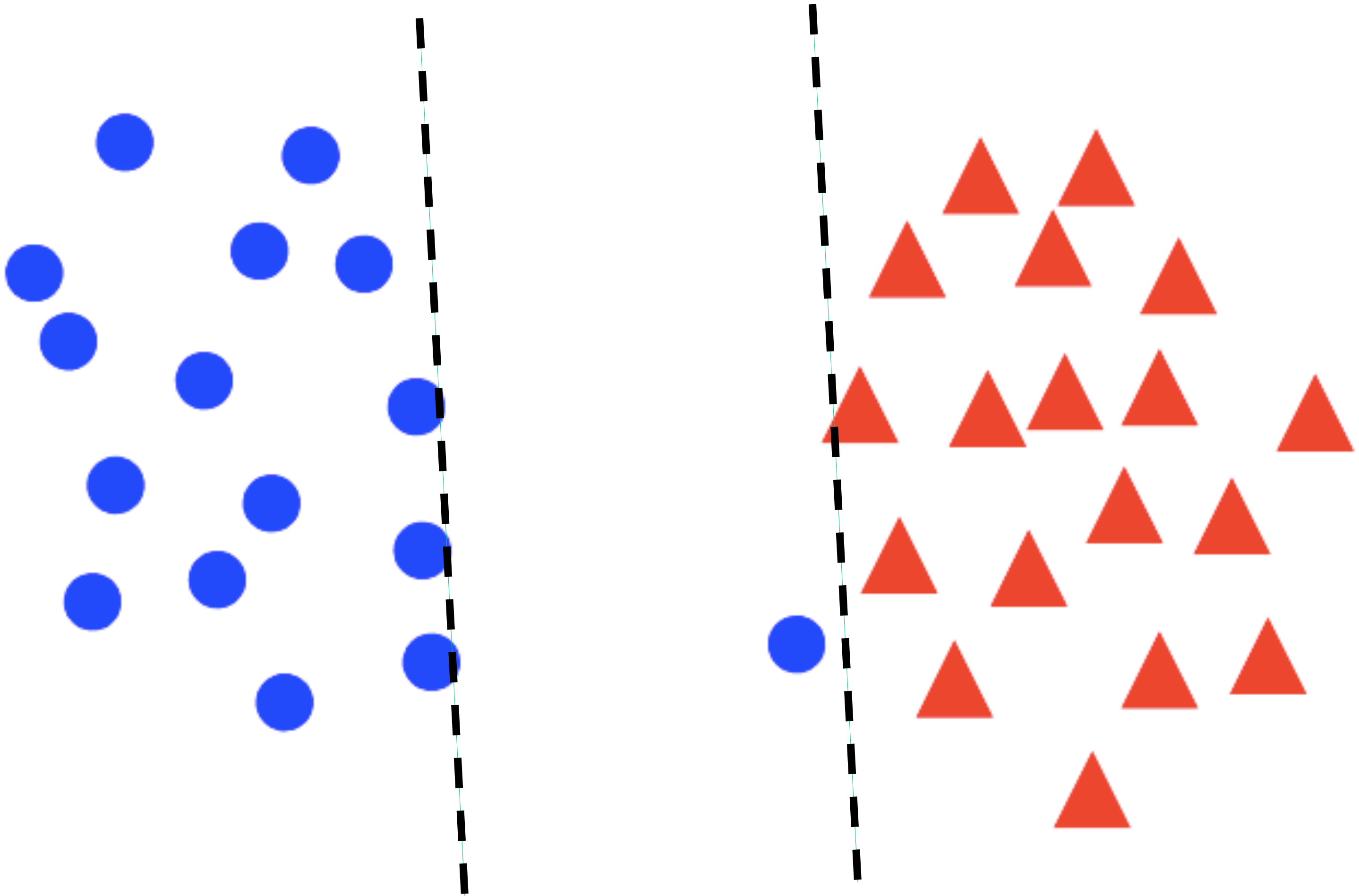
- What about non-separable data?
- And even for separable data, we may prefer a larger margin with a few constraints violated



Source

“Soft margin” formulation

- What about non-separable data?
- And even for separable data, we may prefer a larger margin with a few constraints violated

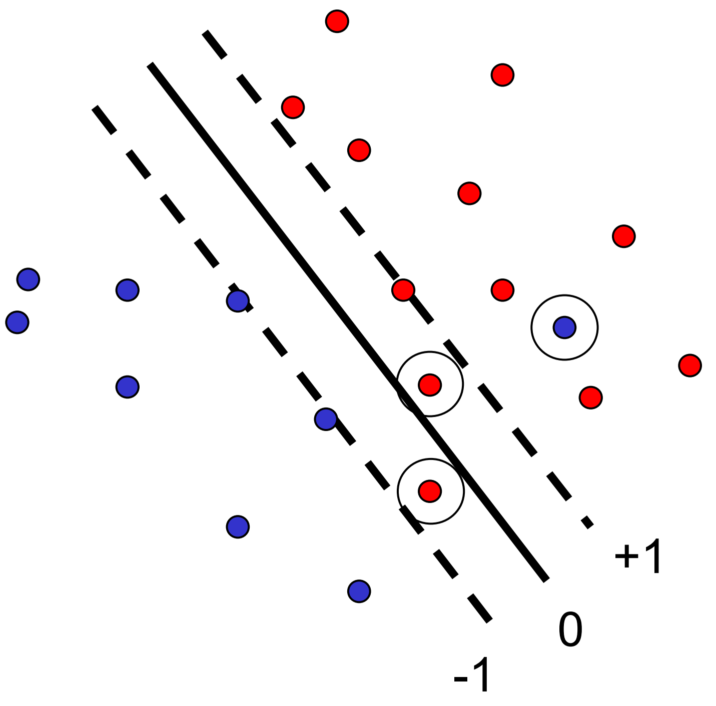


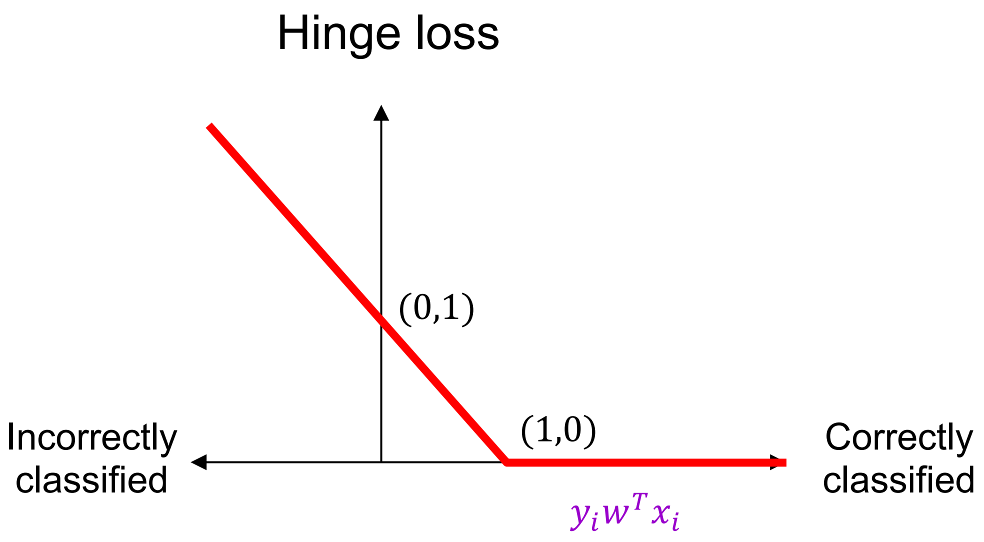
Source

“Soft margin” formulation

- Penalize margin violations using SVM hinge loss:

$$\min_w \frac{\lambda}{2}\|w\|^2 \quad + \quad \sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]$$

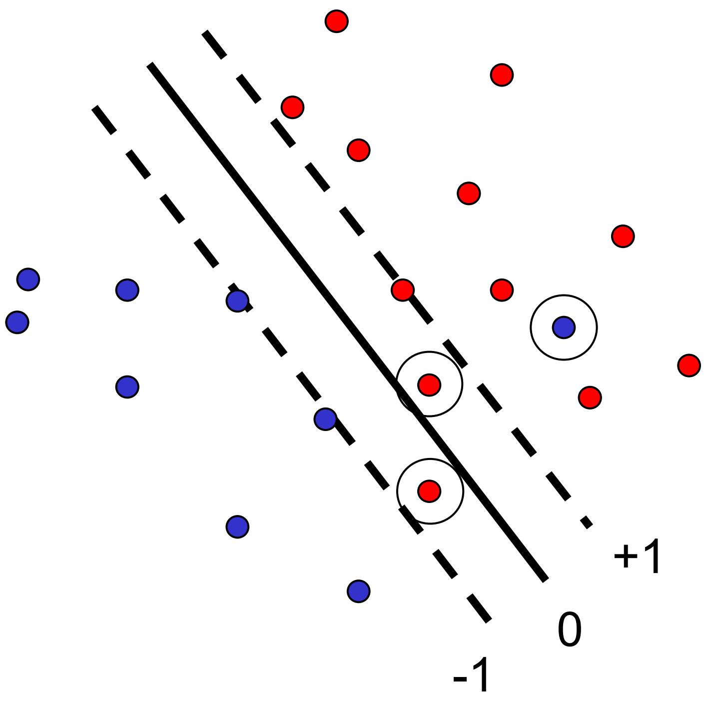


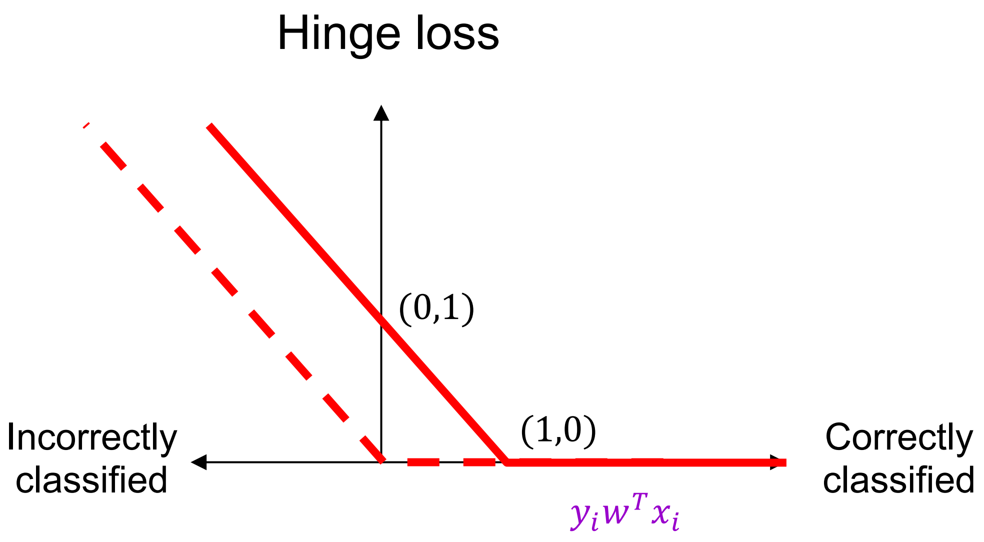


"Soft margin" formulation

- Penalize margin violations using SVM hinge loss:

$$\min_w \frac{\lambda}{2}\|w\|^2 \quad + \quad \sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]$$





Recall hinge loss used by the perceptron update algorithm!

"Soft margin" formulation

- Penalize margin violations using SVM hinge loss:

$$\underbrace{\min_w \frac{\lambda}{2}\|w\|^2}_{\text{Maximize margin – a.k.a. } \textit{regularization}} + \underbrace{\sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]}_{\text{Minimize misclassification loss}}$$

SGD update for SVM

$$l(w, x_i, y_i) = \frac{\lambda}{2n}\|w\|^2 + \max[0, 1 - y_i w^T x_i]$$

$$\nabla l(w, x_i, y_i) = \frac{\lambda}{n} w - \mathbb{I}[y_i w^T x_i < 1] y_i x_i$$

Recall: $\frac{d}{da}\max(0, a) = \mathbb{I}[a > 0]$

SGD update for SVM

$$l(w, x_i, y_i) = \frac{\lambda}{2n}\|w\|^2 + \max[0, 1 - y_i w^T x_i]$$

$$\nabla l(w, x_i, y_i) = \frac{\lambda}{n} w - \mathbb{I}[y_i w^T x_i < 1] y_i x_i$$

- SGD update:
  - If $y_i w^T x_i \geq 1$: $w \leftarrow w - \eta \frac{\lambda}{n} w$
  - If $y_i w^T x_i < 1$: $w \leftarrow w + \eta \left(y_i x_i - \frac{\lambda}{n} w \right)$

S. Shalev-Schwartz et al., [Pegasos: Primal Estimated sub-GrAdient SOlver for SVM](), *Mathematical Programming*, 2011

SVM vs. perceptron

- SVM loss: $l(w, x_i, y_i) = \frac{\lambda}{2n} \|w\|^2 + \max[0, 1 - y_i w^T x_i]$
- SVM update:
  - If $y_i w^T x_i \geq 1$: $w \leftarrow \left(1 - \eta \frac{\lambda}{n}\right) w$
  - If $y_i w^T x_i < 1$: $w \leftarrow \left(1 - \eta \frac{\lambda}{n}\right) w + \eta\, y_i x_i$

- Perceptron loss: $l(w, x_i, y_i) = \max[0, -y_i w^T x_i]$
- Perceptron update:
  - If $y_i w^T x_i < 0$: $w \leftarrow w + \eta\, y_i x_i$
  - Otherwise: do nothing

- What are the differences?

Linear classifiers: Outline

- Examples of classification models: nearest neighbor, linear
- Empirical loss minimization framework
- Linear classification models
  1. Linear regression
  2. Logistic regression
  3. Perceptron training algorithm
  4. Support vector machines
- General recipe: data loss, regularization

General recipe

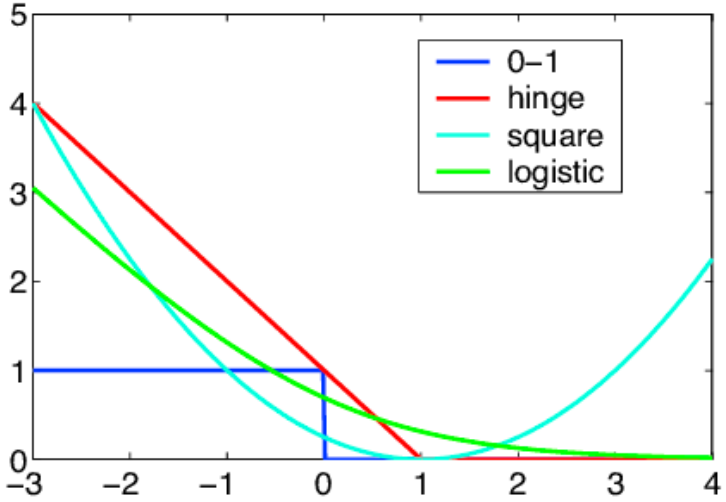
- Find parameters w that minimize the sum of a *regularization loss* and a *data loss*:

$$\hat{L}(w) = \lambda R(w) + \frac{1}{n}\sum_{i=1}^{n} l(w, x_i, y_i)$$

empirical loss regularization data loss

L2 regularization:

$$R(w) = \frac{1}{2}\|w\|_2^2$$



Closer look at L2 regularization

- Regularized objective: $\hat{L}(w) = \frac{\lambda}{2}\|w\|_2^2 + \sum_{i=1}^{n} l(w, x_i, y_i)$
- Gradient of objective:

$$\nabla\hat{L}(w) = \lambda w + \sum_{i=1}^{n} \nabla l(w, x_i, y_i)$$

- SGD update:

$$w \leftarrow w - \eta\left(\frac{\lambda}{n} w + \nabla l(w, x_i, y_i)\right)$$

$$w \leftarrow \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \nabla l(w, x_i, y_i)$$

- Interpretation: weight decay

General recipe

- Find parameters w that minimize the sum of a *regularization loss* and a *data loss*:

$$\hat{L}(w) = \lambda R(w) + \frac{1}{n}\sum_{i=1}^{n} l(w, x_i, y_i)$$

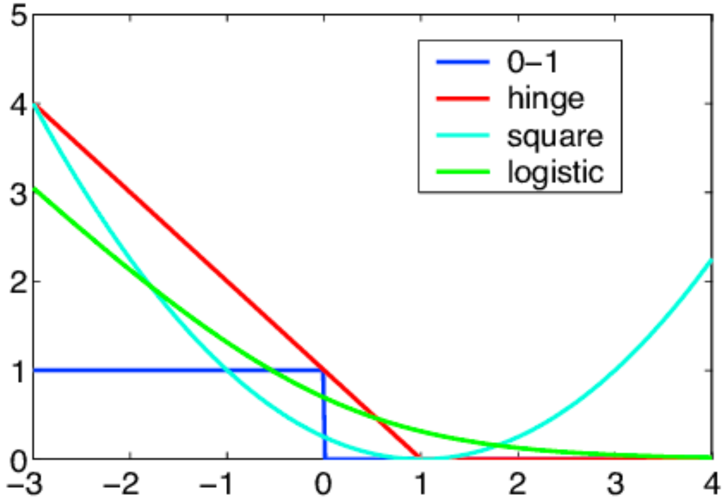
empirical loss — regularization — data loss

L2 regularization:

$$R(w) = \frac{1}{2}\|w\|_2^2$$

L1 regularization:

$$R(w) = \|w\|_1$$



Closer look at L1 regularization

- Regularized objective:

$$\hat{L}(w) = \lambda \|w\|_1 + \sum_{i=1}^{n} l(w, x_i, y_i)$$

$$= \lambda \sum_{d} \left|w^{(d)}\right| + \sum_{i=1}^{n} l(w, x_i, y_i)$$

- Gradient: $\nabla \hat{L}(w) = \lambda \operatorname{sgn}(w) + \sum_{i=1}^{n} \nabla l(w, x_i, y_i)$

  (here sgn is an elementwise function)

- SGD update:

$$w \leftarrow w - \frac{\eta \lambda}{n} \operatorname{sgn}(w) - \eta \nabla l(w, x_i, y_i)$$

- Interpretation: encouraging sparsity

General recipe

- Find parameters w that minimize the sum of a *regularization loss* and a *data loss*:

$$\hat{L}(w) = \lambda R(w) + \frac{1}{n}\sum_{i=1}^{n} l(w, x_i, y_i)$$

empirical loss — regularization — data loss

- Optimize by *stochastic gradient descent* (SGD): At each iteration, sample a single data point (x_i, y_i) and take a step in the direction *opposite* the gradient of the loss for that point:

$$w \leftarrow w - \eta \, \nabla_w \left[\frac{\lambda}{n} R(w) + l(w, x_i, y_i)\right]$$

Summary of SGD updates

- Linear regression:

$$w \leftarrow w + \eta\, (y_i - w^T x_i)\, x_i$$

- Logistic regression:

$$w \leftarrow w + \eta\, \sigma(-y_i w^T x_i)\, y_i x_i$$

- Perceptron:

$$w \leftarrow w + \eta\, \mathbb{I}[y_i w^T x_i < 0]\, y_i x_i$$

- SVM:

$$w \leftarrow \left(1 - \frac{\eta\lambda}{n}\right) w + \eta\, \mathbb{I}[y_i w^T x_i < 1]\, y_i x_i$$

Linear classifiers: Outline

- Examples of classification models: nearest neighbor, linear
- Empirical loss minimization framework
- Linear classification models
  1. Linear regression
  2. Logistic regression
  3. Perceptron training algorithm
  4. Support vector machines
- General recipe: data loss, regularization
- Multi-class classification with a Softmax Function

One-vs-all Classification with a Softmax

- Let $y \in \{1, \ldots, C\}$
- Learn C scoring functions $f_1, f_2, \ldots, f_C$

- We can squash the vector of responses $(f_1, \ldots, f_c)$ into a vector of "probabilities":

$$\mathrm{softmax}(f_1, \ldots, f_c) = \left(\frac{\exp(f_1)}{\sum_j \exp(f_j)}, \ldots, \frac{\exp(f_C)}{\sum_j \exp(f_j)} \right)$$

- The outputs are between 0 and 1 and sum to 1
- If one of the inputs (*logits*) is much larger than the others, then the corresponding softmax value will be close to 1 and others will be close to 0

Softmax and sigmoid

- For two classes:

$$\mathrm{softmax}(0, f) = \left(\frac{\exp(0)}{\exp(0) + \exp(f)}, \frac{\exp(f)}{\exp(0) + \exp(f)}\right)$$

$$= \left(\frac{1}{1+\exp(f)}, \frac{\exp(f)}{1+ \exp(f)}\right)$$

$$= (1 - \sigma(f), \sigma(f))$$

- Thus, softmax is the generalization of sigmoid for more than two classes

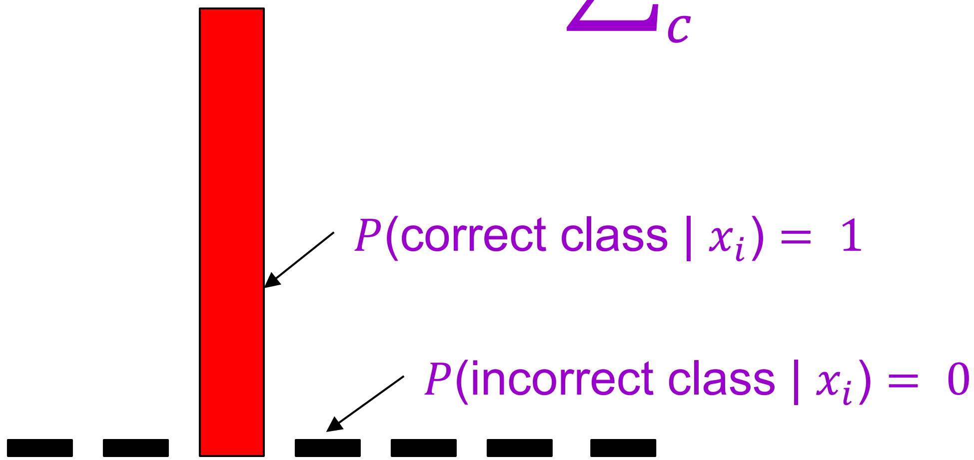
Cross-entropy loss

- It is natural to use negative log likelihood loss with softmax:

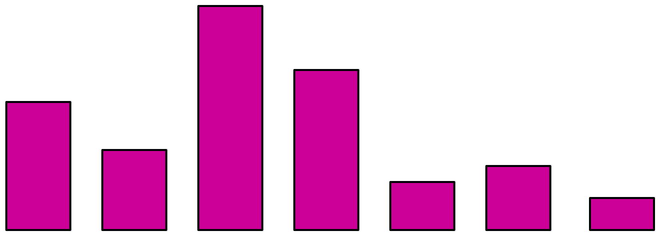
$$l(W, x_i, y_i) = -\log P_W(y_i|x_i) = -\log\left(\frac{\exp(w_{y_i}^T x_i)}{\sum_j \exp(w_j^T x_i)}\right)$$

- This is also the *cross-entropy* between the "empirical" distribution $\hat{P}(c|x_i) = \mathbb{I}[c = y_i]$ and "estimated" distribution $P_W(c|x_i)$:

$$-\sum_c \hat{P}(c|x_i) \log P_W(c|x_i)$$



Empirical distribution $\hat{P}(c|x_i)$



Estimated distribution $P_W(c|x_i)$

SGD with cross-entropy loss

$$l(W, x_i, y_i) = -\log P_W(y_i|x_i) = -\log\left(\frac{\exp(w_{y_i}^T x_i)}{\sum_j \exp(w_j^T x_i)}\right)$$

$$= -w_{y_i}^T x_i + \log\left(\sum_j \exp(w_j^T x_i)\right)$$

- Gradient w.r.t. w_{y_i}:

$$-x_i + \frac{\exp(w_{y_i}^T x_i) x_i}{\sum_j \exp(w_j^T x_i)} = (P_W(y_i|x_i) - 1) x_i$$

- Gradient w.r.t. w_c, $c \neq y_i$:

$$\frac{\exp(w_c^T x_i) x_i}{\sum_j \exp(w_j^T x_i)} = P_W(c|x_i) x_i$$

SGD with cross-entropy loss

- Gradient w.r.t. w_{y_i}: $(P_W(y_i|x_i) - 1)x_i$
- Gradient w.r.t. w_c, $c \neq y_i$: $P_W(c|x_i)x_i$
- Update rule:
  - For y_i:

$$w_{y_i} \leftarrow w_{y_i} + \eta\big(1 - P_W(y_i|x_i)\big)x_i$$

  - For $c \neq y_i$:

$$w_c \leftarrow w_c - \eta P_W(c|x_i)x_i$$

Softmax trick: Avoiding overflow

- Exponentiated values $\exp(f_c)$ can become very large and cause overflow
- Note that adding the same constant to all softmax inputs (*logits*) does not change the output of the softmax:

$$\frac{\exp(f_c + K)}{\sum_j \exp(f_j + K)} = \frac{\exp(K)\exp(f_c)}{\sum_j \exp(K)\exp(f_j)} = \frac{\exp(f_c)}{\sum_j \exp(f_j)}$$

- Then we can let $\mathrm{K} = -\max_j f_j$ (i.e., make largest input to softmax be 0)

Linear classifiers: Outline

- Examples of classification models: nearest neighbor, linear
- Empirical loss minimization framework
- Linear classification models
  1. Linear regression
  2. Logistic regression
  3. Perceptron training algorithm
  4. Support vector machines
- General recipe: data loss, regularization
- Multi-class classification with a Softmax Function