FIT 3080: Intelligent Systems

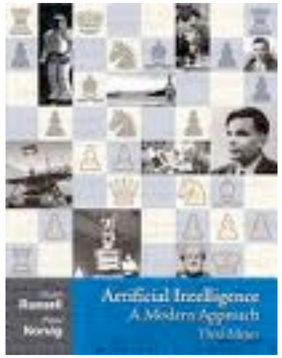
Supervised Machine Learning

Classification and Regression

Gholamreza Haffari – Monash University

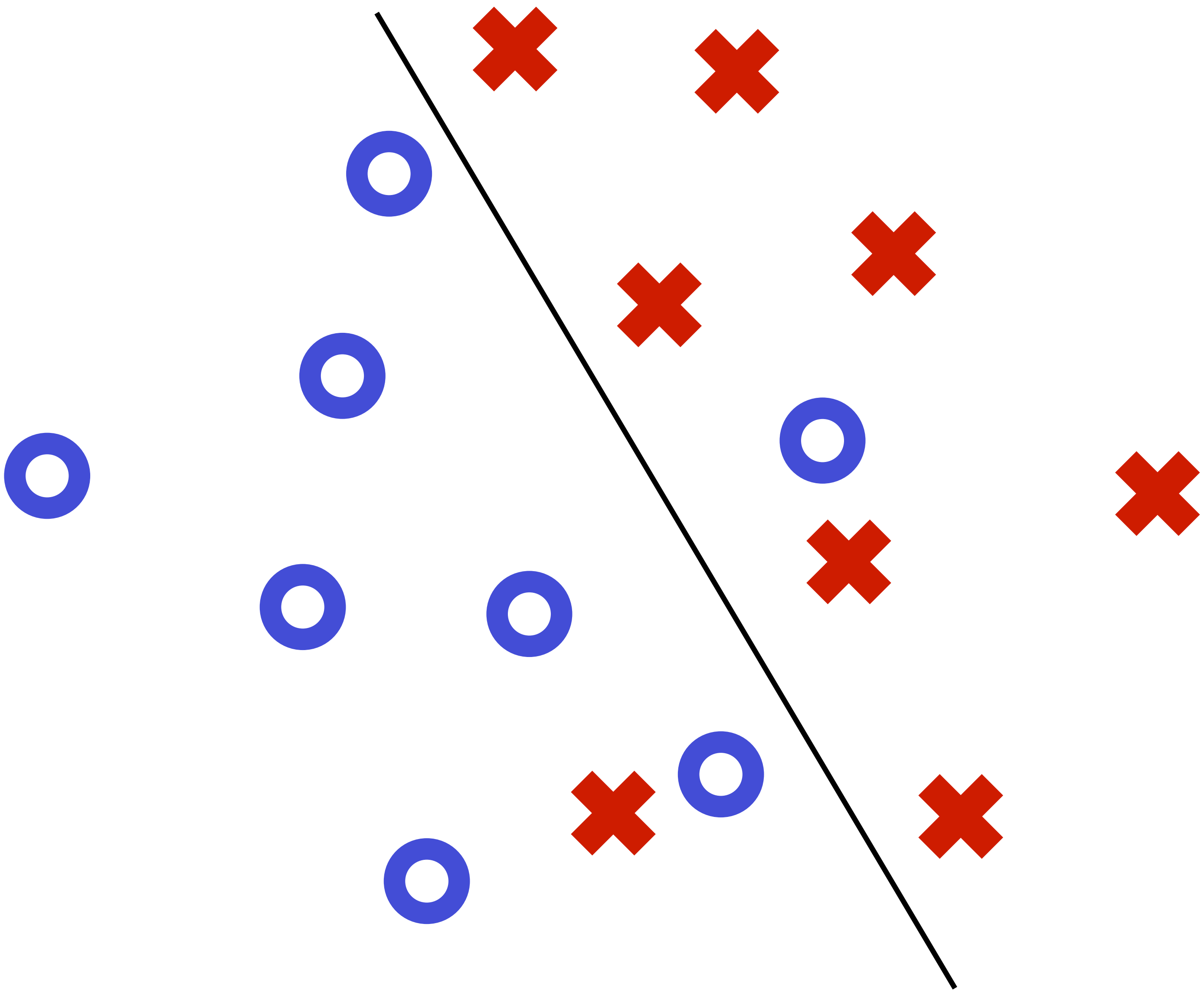
Many slides over the course adapted from Stuart Russell, Andrew Moore, or Dan Klein

Announcements



- Readings:
  - 18.6.1 (regression)
  - 18.6.3 (perceptron)
  - 18.8.1 (K Nearest Neighbor)
  - 13.5.2, 20.1, 20.2.1, 20.2.2 (Naïve Bayes & Maximum Likelihood)

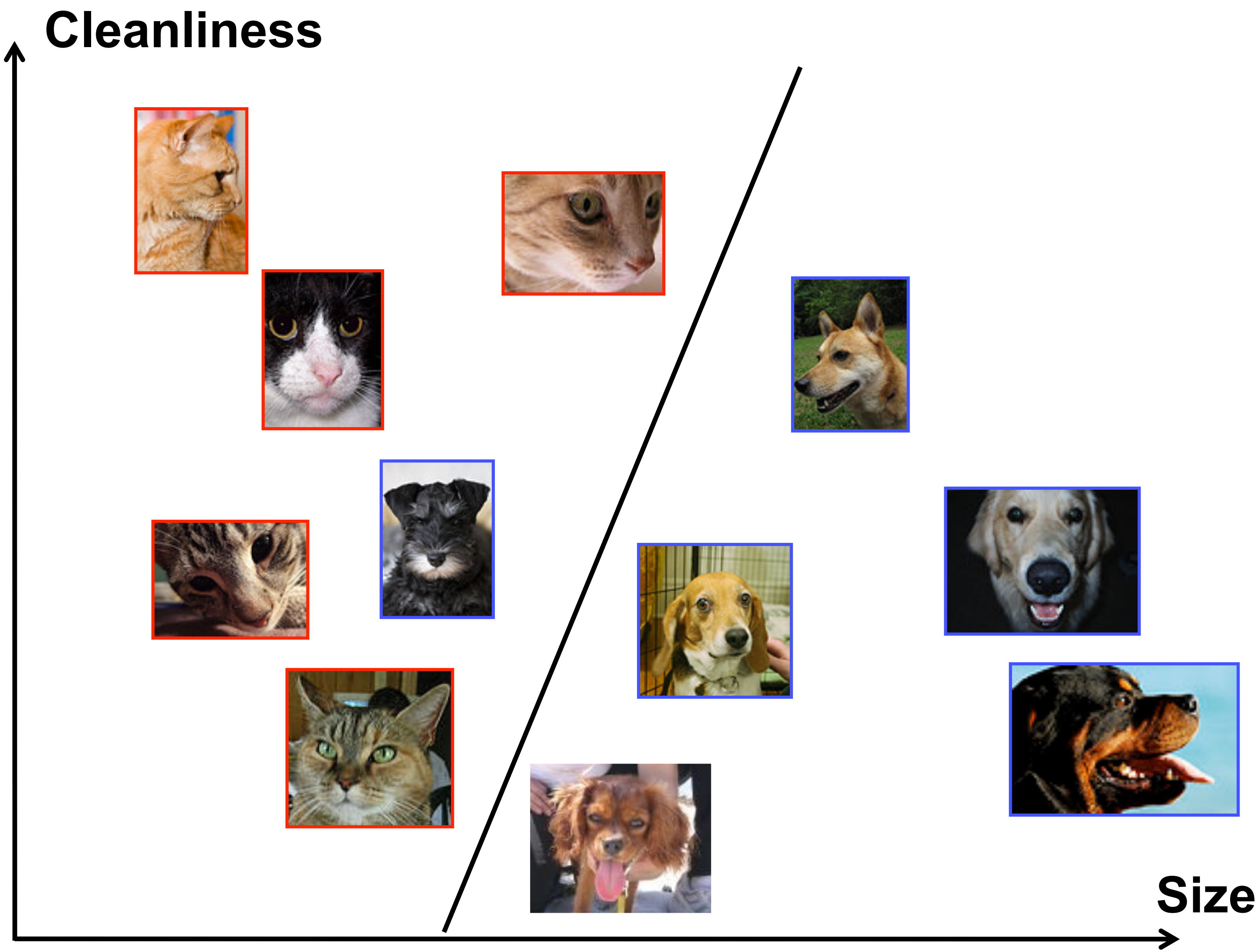
Classification



Cat

Dog

?


Cleanliness
Size

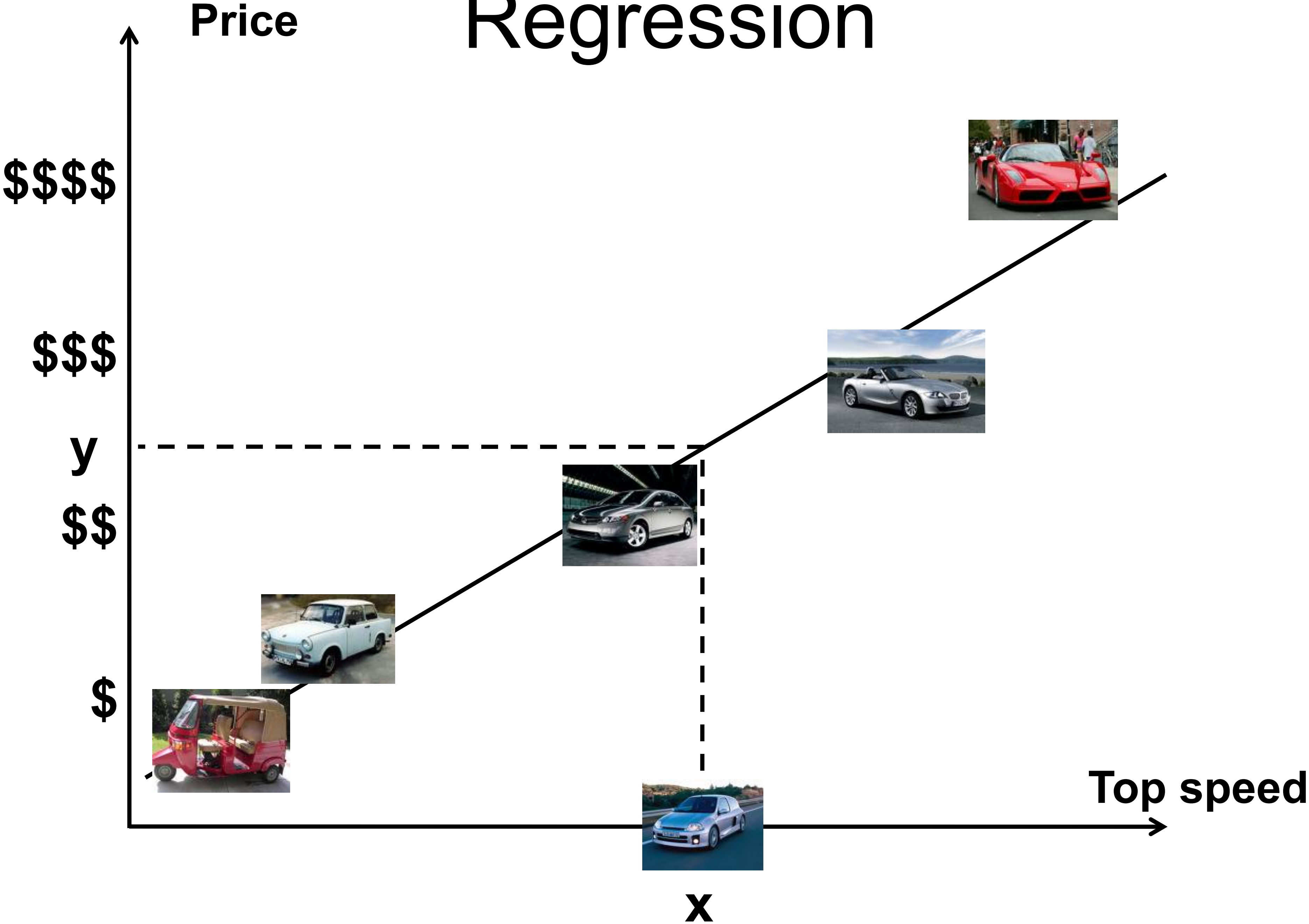
$ $$ $$$ $$$$





?

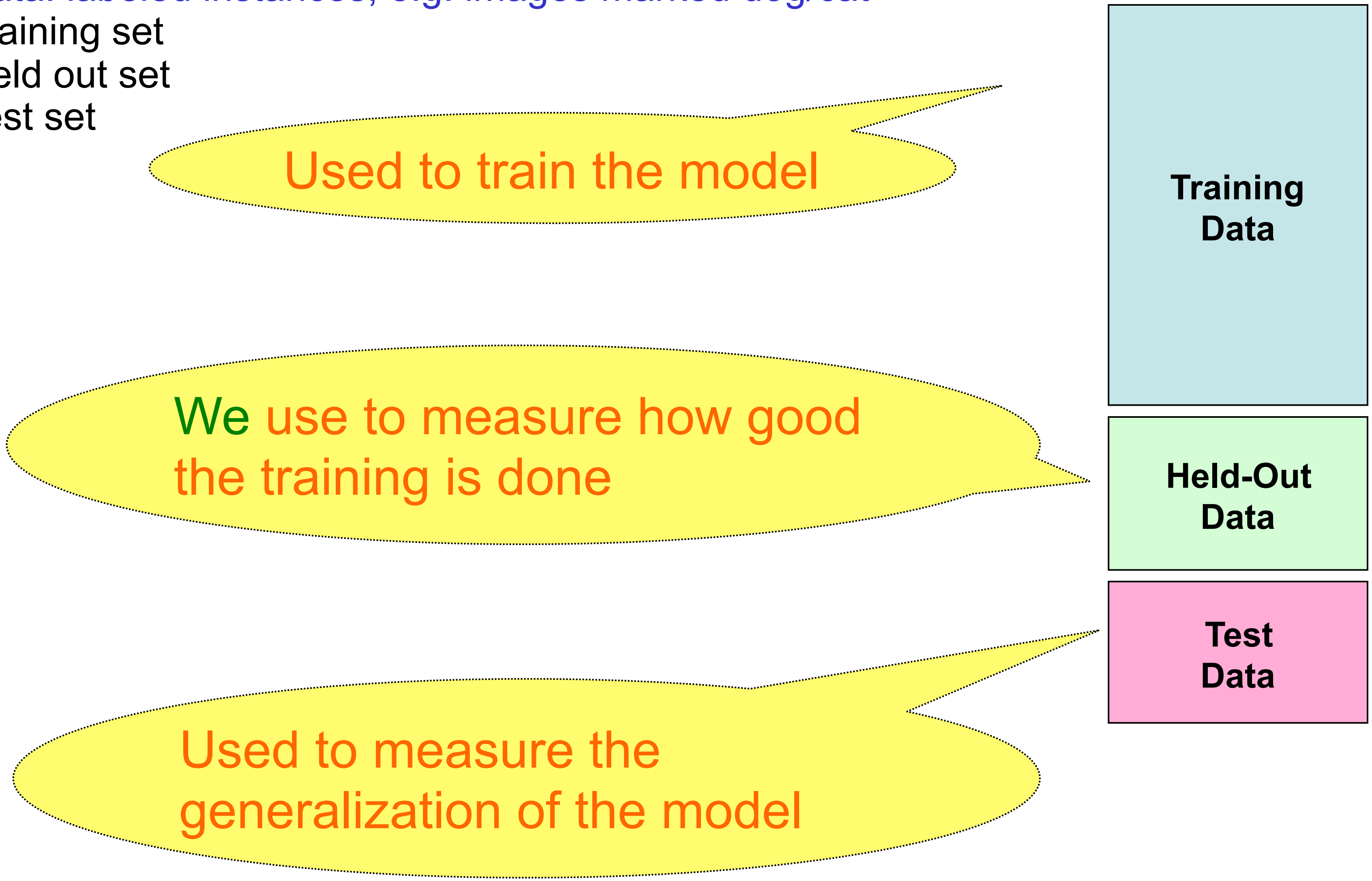
Regression



Important Concepts

- Data: labeled instances, e.g. images marked dog/cat

  - Training set
  - Held out set
  - Test set

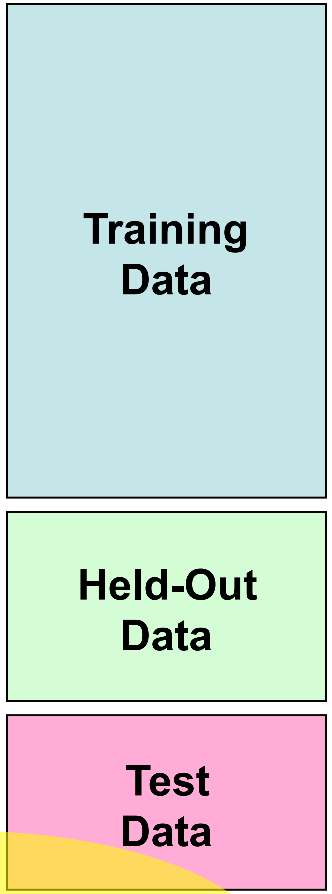


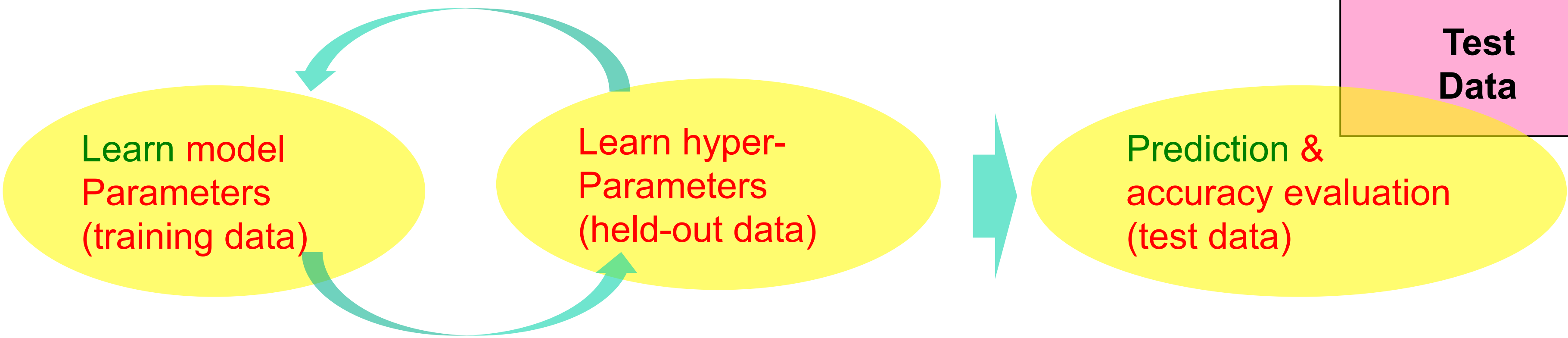
Important Concepts

- Data: labeled instances, e.g. images marked dog/cat
  - Training set
  - Held out set
  - Test set

- Features: attribute-value pairs which characterize each x

- Experimentation cycle
  - Learn model parameters on Training Data
  - Held-out Data for the hyper-parameters of the model (e.g. what should be the max height of the decision tree)
  - Compute accuracy of Test Data
  - Very important: never “peek” at the test set!





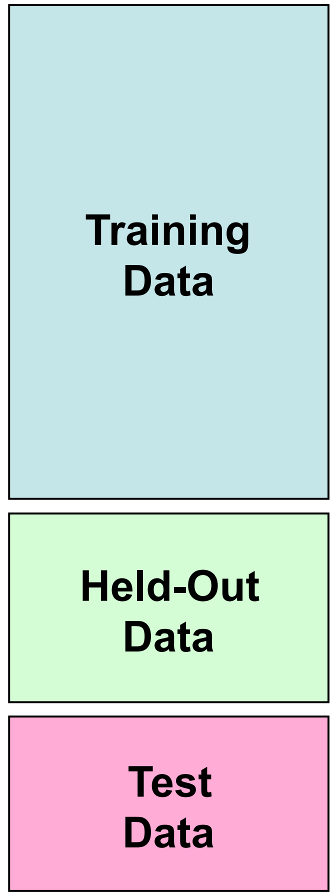
Important Concepts

- Data: labeled instances, e.g. images marked dog/cat
  - Training set
  - Held out set
  - Test set

- Features: attribute-value pairs which characterize each x

- Experimentation cycle
  - Learn model parameters on Training Data
  - Held-out Data for the hyper-parameters of the model (e.g. what should be the max height of the decision tree)
  - Compute accuracy of Test Data
  - Very important: never “peek” at the test set!

- Evaluation
  - Accuracy: fraction of instances predicted correctly



If 80 predictions are correct out of 100 then
accuracy = 80/100
=0.8

Important Concepts

- Data: labeled instances, e.g. images marked dog/cat
  - Training set
  - Held out set
  - Test set

- Features: attribute-value pairs which characterize each x

- Experimentation cycle
  - Learn model parameters on Training Data
  - Held-out Data for the hyper-parameters of the model (e.g. what should be the max height of the decision tree)
  - Compute accuracy of Test Data
  - Very important: never “peek” at the test set!

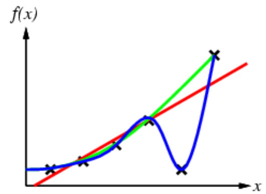
- Evaluation
  - Accuracy: fraction of instances predicted correctly

- Overfitting and generalization
  - Want a classifier/regressor which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well

Training Data

Held-Out Data

Test Data



Outline

- Classification:
  - Perceptron
  - Naïve-Bayes (a probabilistic model)
  - K nearest neighbor (KNN) classifier

- Regression:
  - Linear models based on least square errors

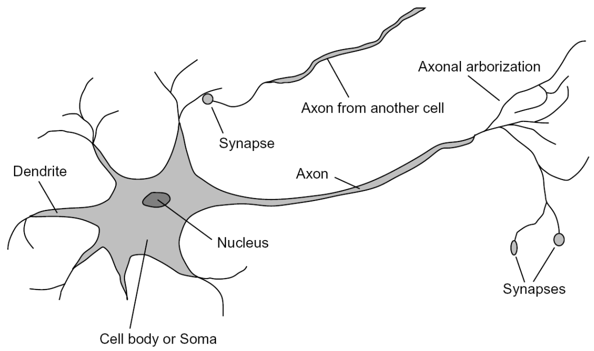
Outline

- Classification:
  - Perceptron
  - Naïve-Bayes (a probabilistic model)
  - K nearest neighbor (KNN) classifier

- Regression:
  - Linear models based on least square errors

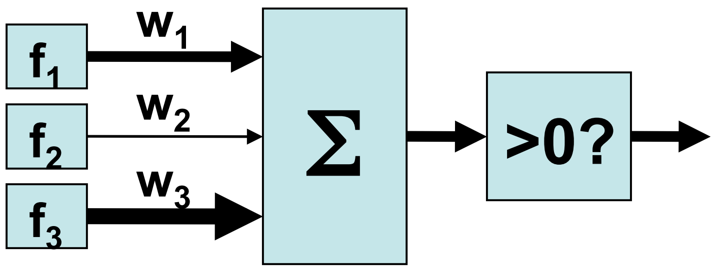
The Binary Perceptron

- Inputs are feature values $f_i(x)$
- Each feature has a weight w_i
- Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x)$$

- If the activation is:
  - Positive, output 1
  - Negative, output 0



Example: Spam Filter

- Input: email
- Output: spam/ham
- Setup:
  - -Get a large collection of example emails, each labeled “spam” or “ham”
  - - Note: Somebody has provided the example labels
  - - Want to learn to predict labels for new, future emails

- Features: The attributes used to make prediction
  - - words in the email’s text

Dear Sir.

First, I must solicit your confidence in this transaction, this is by virture of its nature as being utterly confidencial and top secret. …

TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.

99 MILLION EMAIL ADDRESSES FOR ONLY $99

Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

Example: Spam Filter

- Imagine 4 features:
  - Free (number of occurrences of "free")
  - Money (occurrences of "money")
  - BIAS (always has value 1)

$$\overbrace{\sum_i w_i \cdot f_i(x)}^{w \cdot f(x)}$$

x	$f(x)$	w
"free money"	**BIAS : 1** **free : 1** **money : 1** **the : 0** **...**	**BIAS : -3** **free : 4** **money : 2** **the : 0** **...**

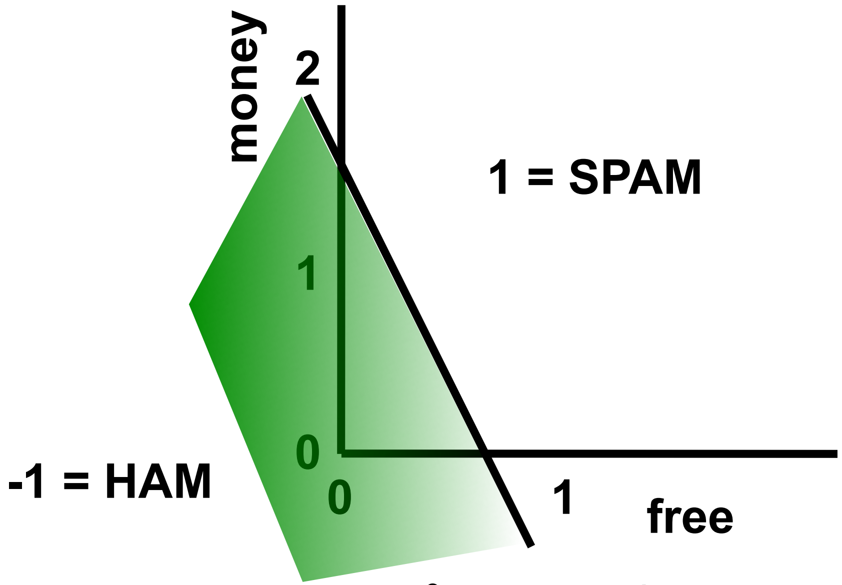
$$(1)(4) \quad +$$
$$(1)(2) \quad +$$
$$(0)(0) \quad +$$
$$\ldots$$
$$= 3$$

Prediction: Binary Decision Rule

- **In the space of feature vectors**
  - **Any weight vector is a hyperplane**
  - **One side will be class 1**
  - **Other will be class -1**

$$w$$

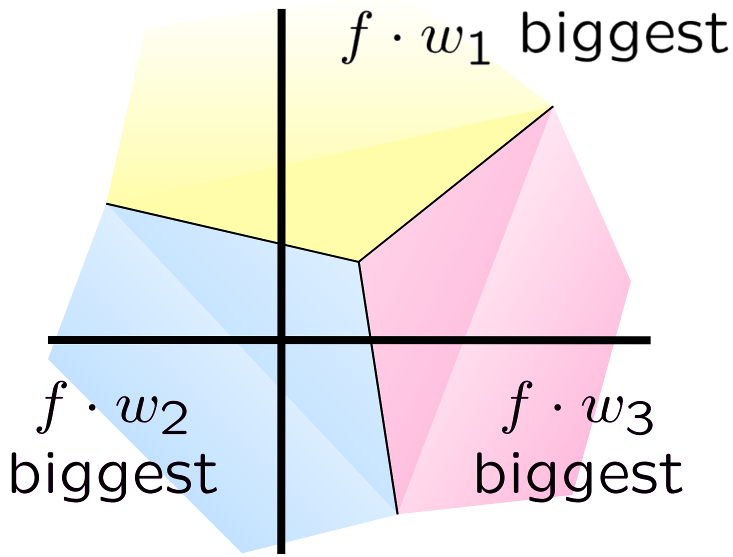
BIAS : -3
free : 4
money : 2
the : 0
...



$$f \cdot w = 0$$

Prediction: Multiclass Decision Rule

- If we have more than two classes:



- Have a weight vector for each class

- Calculate an activation for each class

$$\text{activation}_w(x, c) = \sum_i w_{c,i} \cdot f_i(x)$$

- Highest activation wins

$$c = \arg\max_c \left(\text{activation}_w(x, c)\right)$$

Example

“win the vote”

BIAS	:	1
win	:	1
game	:	0
vote	:	1
the	:	1
...		

w_{SPORTS}

BIAS	:	-2
win	:	4
game	:	4
vote	:	0
the	:	0
...		

$w_{POLITICS}$

BIAS	:	1
win	:	2
game	:	0
vote	:	4
the	:	0
...		

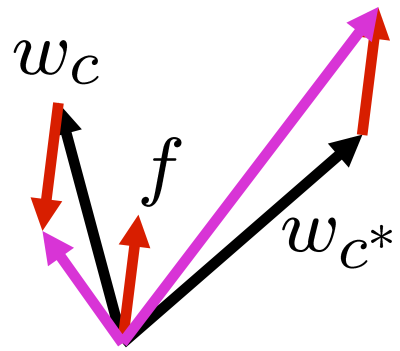
w_{TECH}

BIAS	:	2
win	:	0
game	:	2
vote	:	0
the	:	0
...		

Learning: The Perceptron Update Rule

- Start with zero weights
- Pick up training instances one by one
- Try to classify

$$
\begin{aligned}
c &= \arg\max_c \; w_c \cdot f(x) \\
&= \arg\max_c \; \textstyle\sum_i w_{c,i} \cdot f_i(x)
\end{aligned}
$$



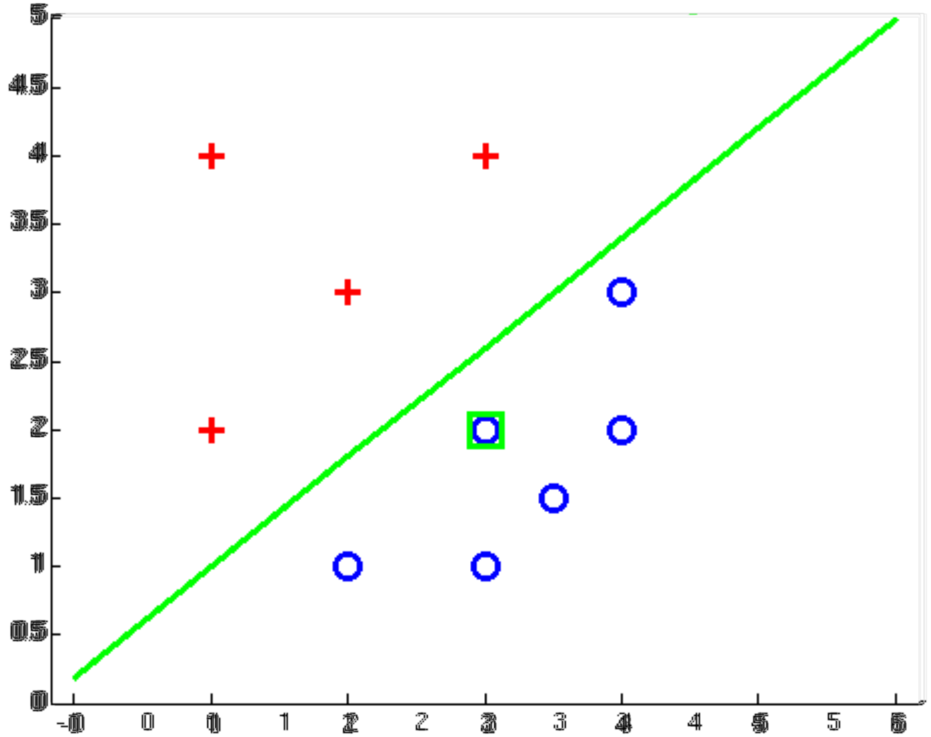
- If correct: no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_c = w_c - f(x)$$

$$w_{c^*} = w_{c^*} + f(x)$$

Examples: Perceptron

- **Separable Case**



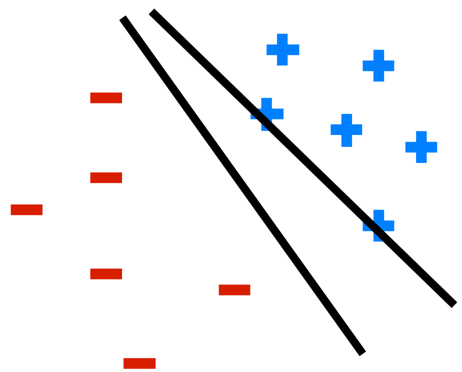
Properties of Perceptron

- Separability: some parameters get the training set perfectly correct

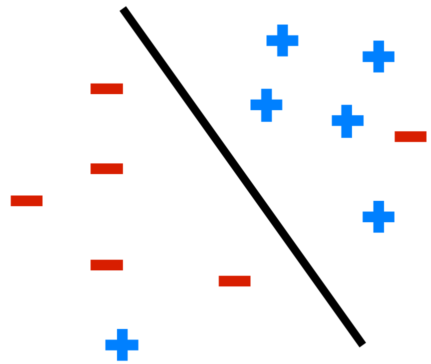
- Convergence: if the training is separable, perceptron will eventually converge (binary case)

  - The convergence can be very slow

Separable

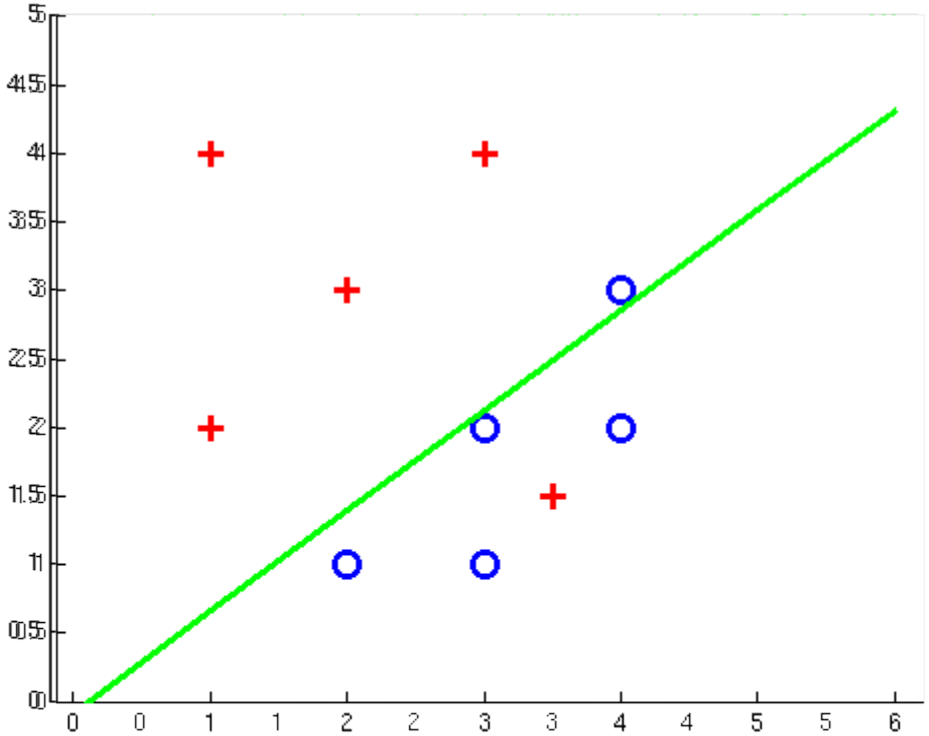


Non-Separable



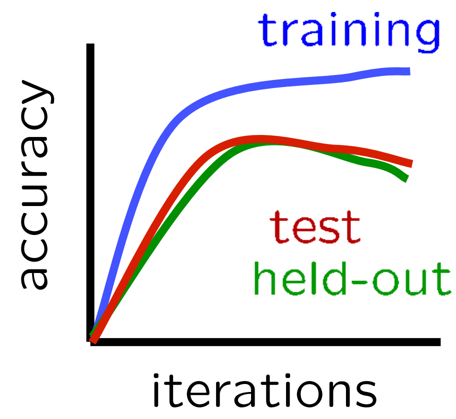
Examples: Perceptron

- **Non-Separable Case**

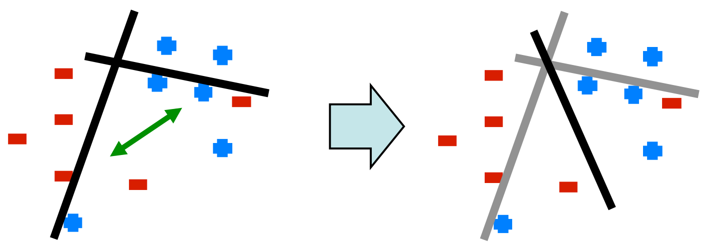


Issues with Perceptrons

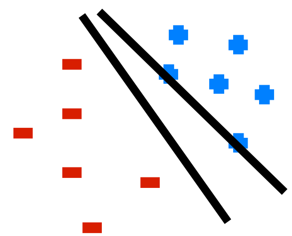
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining isn't quite as bad as overfitting, but is similar



- Non-separability: If the data isn't separable, weights might thrash around
  - Averaging weight vectors over time can help (averaged perceptron)

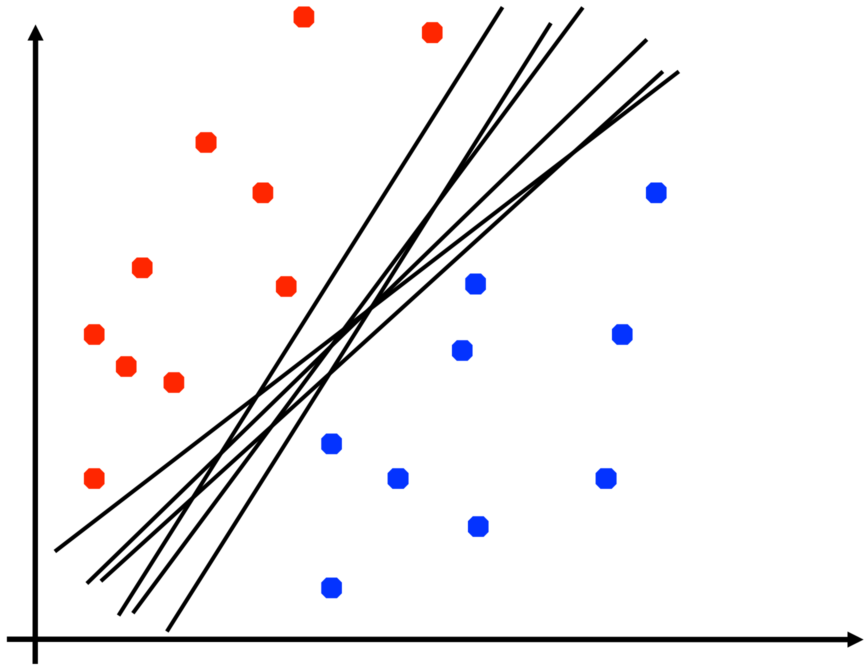


- Mediocre generalization:
  - finds a "barely" separating solution



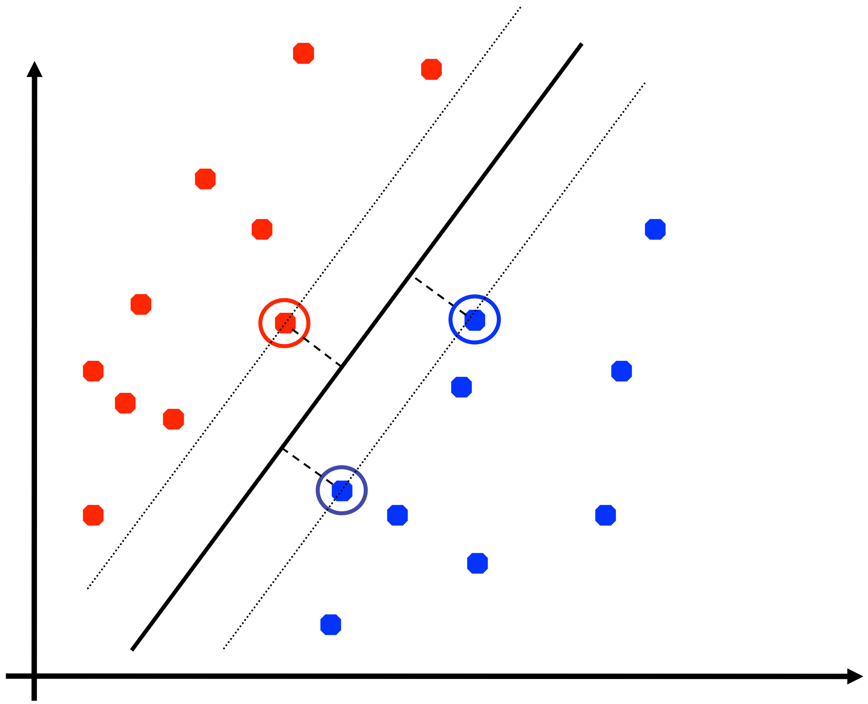
Linear Separators

- Which of these linear separators is optimal?



Support Vector Machines

- Maximizing the margin: good according to intuition and theory.
- Only support vectors matter; other training examples are ignorable.
- Support vector machines (SVMs) find the separator with max margin



Summary

- **A Linear model for classification w.f(x)**

  - prediction is done by thresholding w.f(x)

- **w can be learned by the Perceptron learning alg.**

  - SVM or other algs can be used too

Outline

- Classification:
  - Perceptron
  - Naïve-Bayes (a probabilistic model)
  - K nearest neighbor (KNN) classifier
- Regression:
  - Linear models based on least square errors

Naïve Bayes Classifier

- Based on Bayes rule

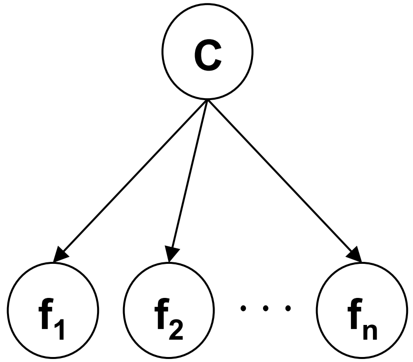
$$\Pr(C_i = c \mid f_{i1},\ldots,f_{in}) = \frac{\Pr(f_{i1},\ldots,f_{in} \mid C_i = c)\Pr(C_i = c)}{\Pr(f_{i1},\ldots,f_{in})}$$

where

C_i is the class of item i

$x_i=\{f_{i,1},\ldots,f_{i,n}\}$ is a set of features

f_{ij} is the value of feature j for item i



- Assuming conditional independence of the features for different classes

$$\Pr(C_i = c \mid f_{i1},\ldots,f_{in}) = \alpha \prod_{k=1}^{n} \Pr(f_{ik} \mid C_i = c)\Pr(C_i = c)$$

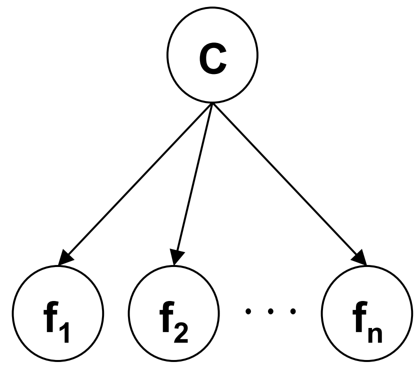
- where α is a normalizing constant

Prediction: Naïve Bayes Classifier

- The predicted class for a new $x=(f_1,..,f_n)$ is:

$$c^* = \arg\max_c \Pr(c \mid f_1,\ldots,f_n)$$

$$= \arg\max_c \prod_{k=1}^{n} \Pr(f_k \mid c)\Pr(c)$$



Example: Naïve Bayes Classifier

- 4 features – outlook, temperature, humidity, wind
- 2 target classes – YES/NO
- Probability of a class:

$c =$	YES	NO
$\Pr(C_i=c)$	9/14	5/14

obtained from the data

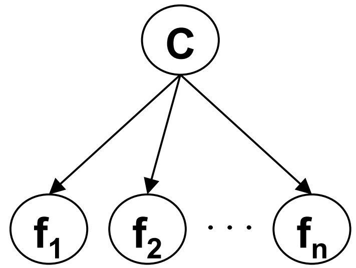
- Calculating *$Pr(C_i=YES|v_{i1}=sunny, v_{i2}=hot, v_{i3}=high, v_{i4}=weak)$*

$$
\begin{aligned}
&\Pr(C_i = YES \mid f_{i1} = sunny, f_{i2} = hot, f_{i3} = high, f_{i4} = weak) \\
&= \alpha \Pr(f_{i1} = sunny \mid C_i = YES) \times \Pr(f_{i2} = hot \mid C_i = YES) \times \\
&\quad \Pr(f_{i3} = high \mid C_i = YES) \times \Pr(f_{i4} = weak \mid C_i = YES) \times \Pr(C_i = YES) \\
&= \alpha \frac{2}{9} \times \frac{2}{9} \times \frac{3}{9} \times \frac{6}{9} \times \frac{9}{14} = \alpha \times 0.007
\end{aligned}
$$

- This calculation is repeated for all values of *C*

Learning the Parameters

- Estimating the CPTs P(c) and P(f_k|c)



1. Empirically: use training data

- For each outcome x, look at the *empirical rate* of that value:

$P_{\mathsf{ML}}(\textcolor{red}{\mathsf{r}}) = 1/3$

$$P_{\mathsf{ML}}(x) = \frac{\mathsf{count}(x)}{\mathsf{total\ samples}}$$

- This is the estimate that maximizes the *likelihood of the data*

2. Elicitation: ask a human!

- Usually need domain experts, and sophisticated ways of eliciting probabilities (e.g. betting games)

- Trouble calibrating

Maximum Likelihood Principle

- Suppose you have data D, and a probabilistic model parameterized by Θ
  - Need to learn parameters Θ from data D

- Likelihood: The probability of data based on the model

- Maximum likelihood: Choose Θ^* which maximizes (the log of) the likelihood function:

$$\Theta^* := \operatorname{argmax}_{\Theta} \log P_{\Theta}(D)$$

Likelihood

Example: Maximum Likelihood

- There is a coin where its probability to come Head is Θ
- The coin has been tossed 3 times: H T T
- What's the maximum likelihood estimate for Θ ?

- Likelihood L(Θ) := **P(H) * P(T) *P(T)**

$$= \Theta \quad * \quad (1-\Theta) \quad * \quad (1-\Theta)$$

- To maximize, set the derivative to zero:

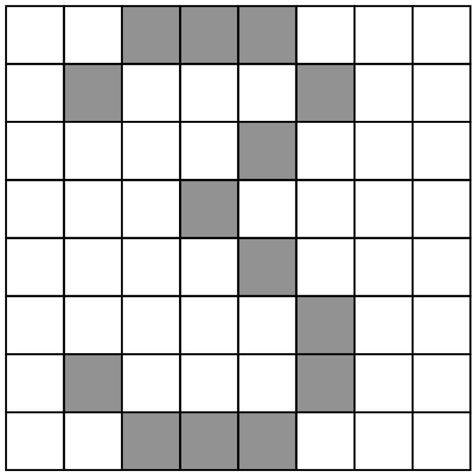
$$\frac{\partial}{\partial\theta}[\log L(\theta)] = \frac{\partial}{\partial\theta}[\log\theta + 2\log(1-\theta)]$$

$$= \frac{1}{\theta} - \frac{2}{1-\theta} = 0 \Rightarrow \theta = \frac{1}{1+2}$$

$$\frac{\text{count}(x)}{\text{total samples}}$$

Example: A Digit Recognizer

- **Input: pixel grids**



- **Output: a digit 0-9**

Naïve Bayes for Digits

- Features from images :

  - One feature f_{ij} for each grid position <i,j>

  - Possible feature values are on / off,

    > based on whether intensity is more or less than 0.5 in underlying image

  - Each input image is mapped to a feature vector, e.g.

 $\rightarrow \langle F_{0,0} = 0 \;\; F_{0,1} = 0 \;\; F_{0,2} = 1 \;\; F_{0,3} = 1 \;\; F_{0,4} = 0 \;\; \ldots F_{15,15} = 0 \rangle$

> lots of features, each is binary

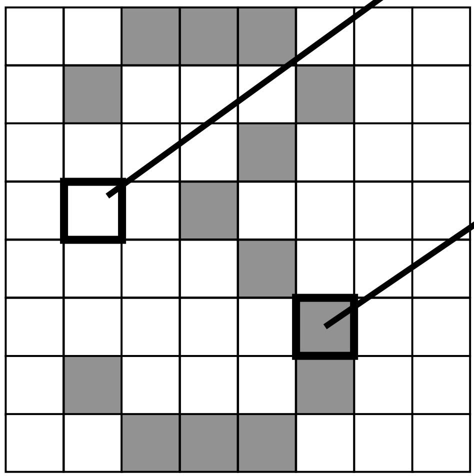
- Naïve Bayes model:

$$P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$$

Examples: CPTs

$P(Y)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



$P(F_{3,1} = on|Y)$

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

$P(F_{5,5} = on|Y)$

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

Example: Overfitting

$P(\text{features}, C = 2)$

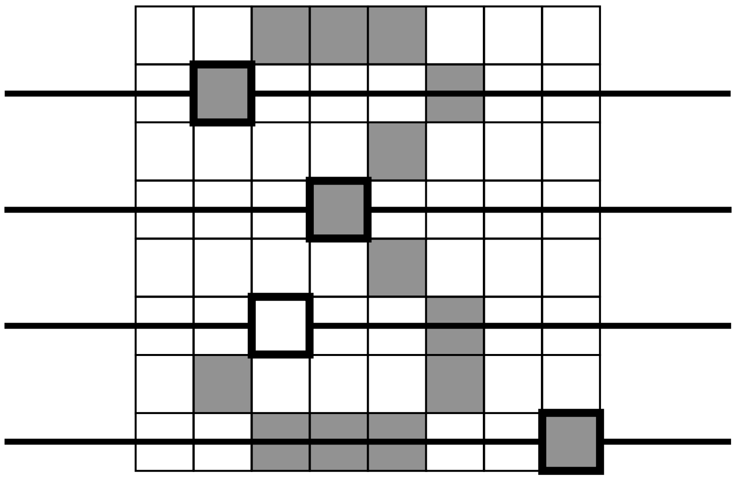
$P(C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.8$

$P(\text{on}|C = 2) = 0.1$

$P(\text{off}|C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.01$



$P(\text{features}, C = 3)$

$P(C = 3) = 0.1$

$P(\text{on}|C = 3) = 0.8$

$P(\text{on}|C = 3) = 0.9$

$P(\text{off}|C = 3) = 0.7$

$P(\text{on}|C = 3) = 0.0$

2 wins!!

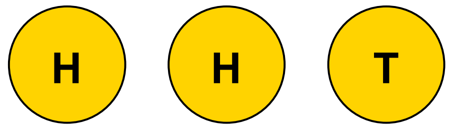
Estimation: Smoothing

- Problems with maximum likelihood estimates:
  - If I flip a coin once, and it's heads, what's the estimate for P(heads)?
  - What if I flip 10 times with 8 heads?
  - What if I flip 10M times with 8M heads?

- Basic idea:
  - We have some prior expectation about parameters (here, the probability of heads)
  - Given little evidence, we should skew towards our prior
  - Given a lot of evidence, we should listen to the data

Estimation: Laplace Smoothing

- Pretend you saw every outcome once more than you actually did



$$P_{LAP}(x) = \frac{c(x)+1}{\sum_x [c(x)+1]}$$

$$= \frac{c(x)+1}{N+|X|}$$

$$P_{ML}(X) =$$

$$P_{LAP}(X) =$$

- Use Laplace smoothing when estimating the parameters of Naïve Bayes, i.e. P(C) and P(f|C)

Summary

- **Naïve Bayes: A probabilistic model for classification**
  - features are independent given the class
  - $c^* = \text{argmax}_c \ P(c|f_1,..,f_n)$
    $= \text{argmax}_c \ P(c) * P(f_1|c) \ .. * P(f_n|c)$

- **Parameters are the CPTs, i.e. P(c) and $P(c|f_k)$**
  - can be learned by maximum likelihood
  - Laplace smoothing is recommended to prevent overfitting

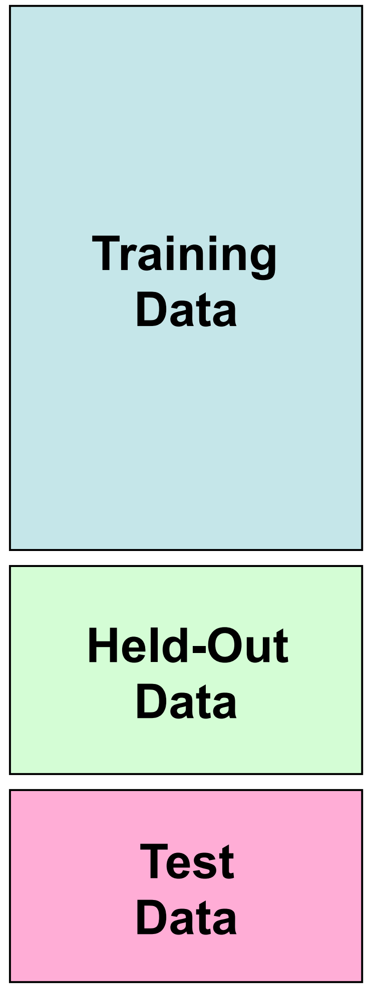
Naïve Bayes vs Perceptron

- In naïve Bayes, parameters:

  - From data statistics
  - Have a causal interpretation
  - One pass through the data

- For the perceptron parameters:

  - From reactions to mistakes
  - Have a discriminative interpretation
  - Go through the data until held-out accuracy maxes out



Outline

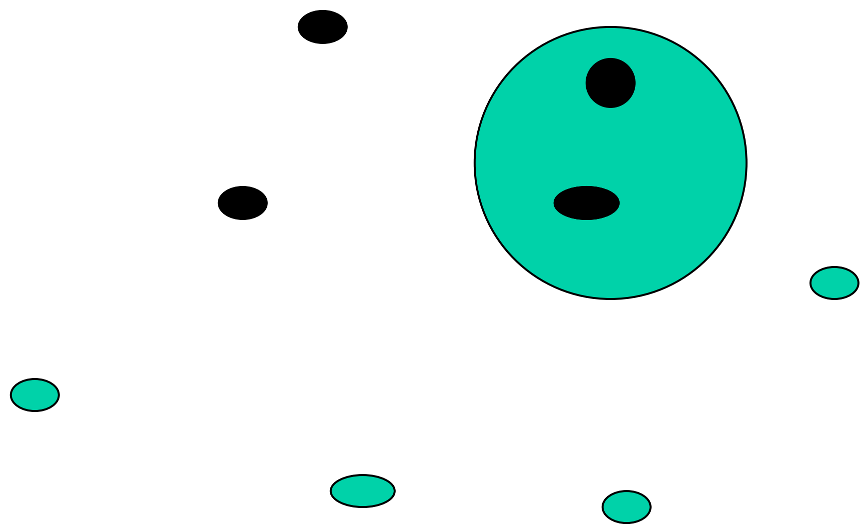
- Classification:
  - Perceptron
  - Naïve-Bayes (a probabilistic model)
  - K nearest neighbor (KNN) classifier

- Regression:
  - Linear models based on least square errors

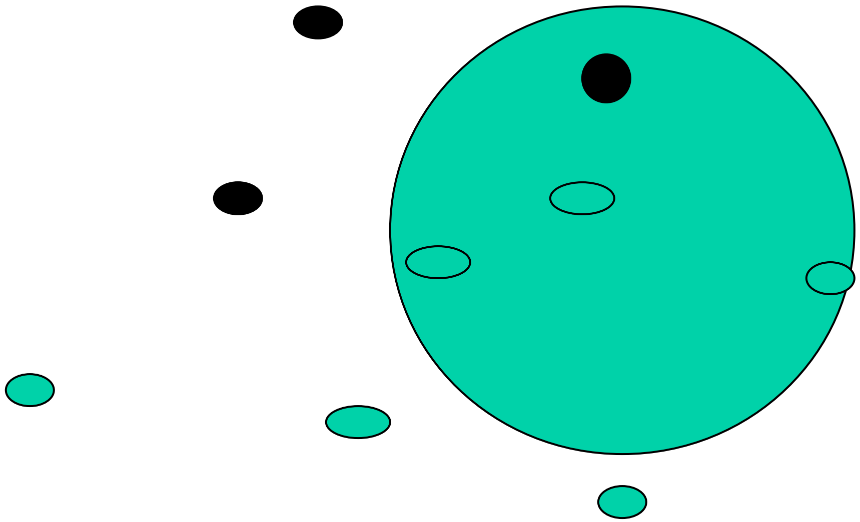
K-Nearest Neighbor

- All instances correspond to points in an n-dimensional Euclidean space
- Classification is delayed till a new instance arrives
- Classification done by comparing to features of the training points
- Target function may be discrete or real-valued

1-Nearest Neighbor



3-Nearest Neighbor



K-Nearest Neighbor

- An instance x_i is represented by ($f_{i,1}$, $f_{i,1}$, .., $f_{i,n}$)

  - $f_{i,k}$ is the value of the "k"th feature for x_i

- Euclidean distance between two instances x_i and x_j is:

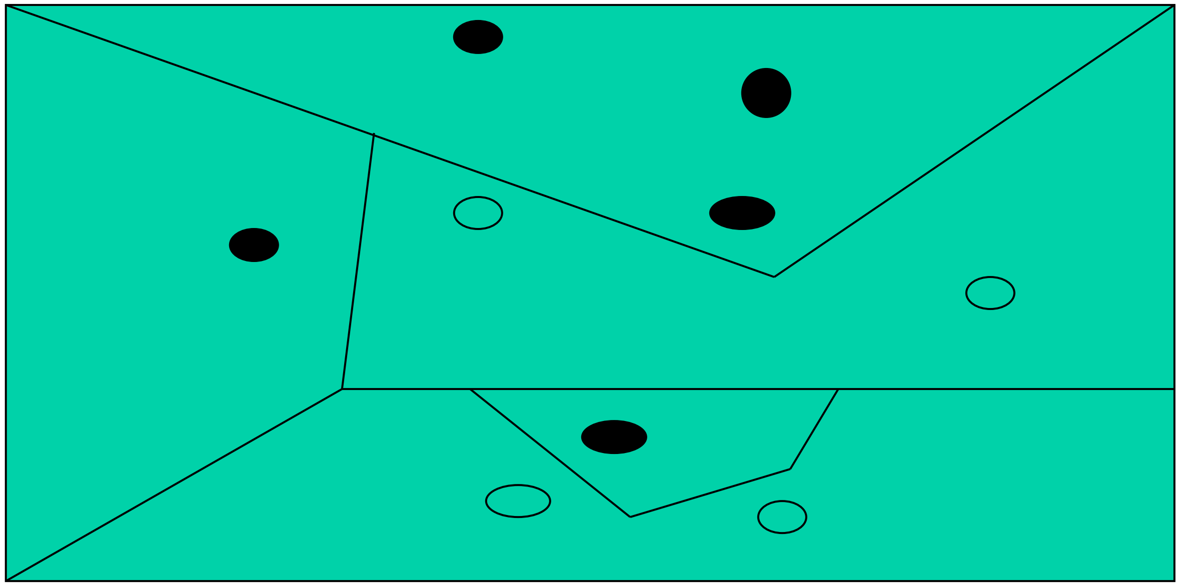
$$d(x_i, x_j) = [\sum_{k=1}^{n} (f_{i,k} - f_{j,k})^2]^{1/2}$$

- Continuous valued target function

  - mean value of the k nearest training examples

Voronoi Diagram
(for 1-Nearest Neighbor)

- Decision surface formed by the training examples



Parameter Learning

- Given the training data and the distance function, there is no training
  - It memorizes the training data
  - As data comes in, the model size grows
  - Hence it's a non-parametric model: infinitely many parameters

- In some cases, the distance function is parameterized and its parameters are learnt
  - E.g. instead of Euclidean distance, use Mahalanobis distance d_M

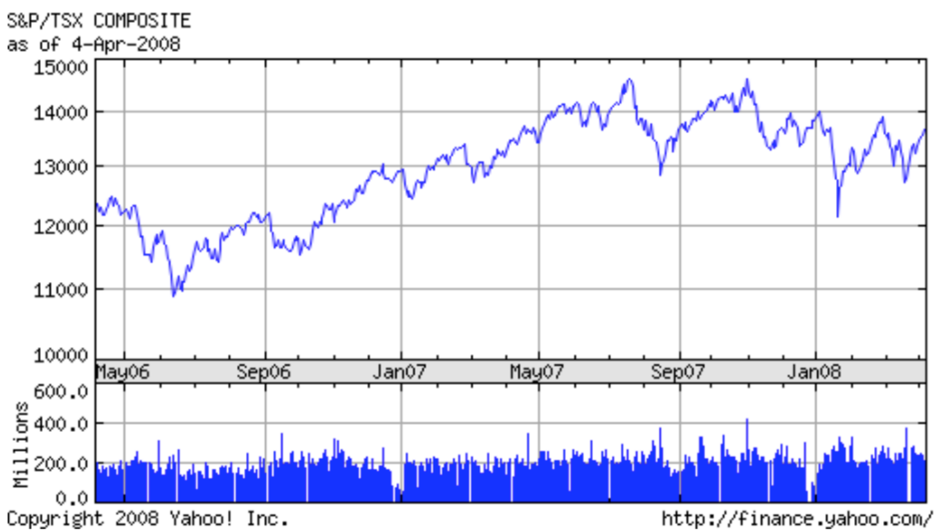
$$d_M(x_i, x_j) := (x_i - \mu)^T M (x_j - \mu)$$

  - The matrix M and the vector μ are the parameters

Outline

- Classification:
  - Perceptron
  - Naïve-Bayes (a probabilistic model)
  - K nearest neighbor (KNN) classifier
- Regression:
  - Linear models based on least square errors

Regression



- Given training set $\{(x_1,t_1),..,(x_m,t_m)\}$
  - t_i is continuous: regression
  - assume $x_i \in R^n, t_i \in R$
- E.g. t_i is stock price, x_i contains company profit, debt, cash flow, …

Error Function

- Given training set $\{(\mathbf{x_1},t_1),\ldots,(\mathbf{x_m},t_m)\}$

  - assume $x_i \in R^n, t_i \in R$

vector

bias

- Linear regression model: $t = \mathbf{w} \cdot \mathbf{x} + w_0$

  - want to learn the parameters $w \in R^n, w_0 \in R$

- Error: Square of the difference between the true and predicted target value for $\mathbf{x}_i$

$$E(w) := \sum_{i=1}^{m} (t_i - w.x_i - w_0)^2$$

truth

prediction

Learning the Parameters

- Look for w^* that minimizes the error function:

$$w^* := \arg\min_w E(w)$$

  - Note E(w) is a convex function, so w^* is unique

- How to find w^*?

  1 - Set the derivatives to zero: $\frac{\partial}{\partial w_k} E(w) = 0$

  2- Or, use an iterative algorithm such as gradient descent

Gradient Descent Algorithm

> initialize w^0 arbitrarily

Learning rate

> for t = 1,2,…

> $w^t \leftarrow w^{t-1} - \alpha \nabla_w E(w)$

> if $\| w^t - w^{t-1} \| < \varepsilon$ then break

The gradient vector

- Stack up the partial derivatives $\frac{\partial}{\partial w_k} E(w)$ in a vector

Illustration of Gradient Descent

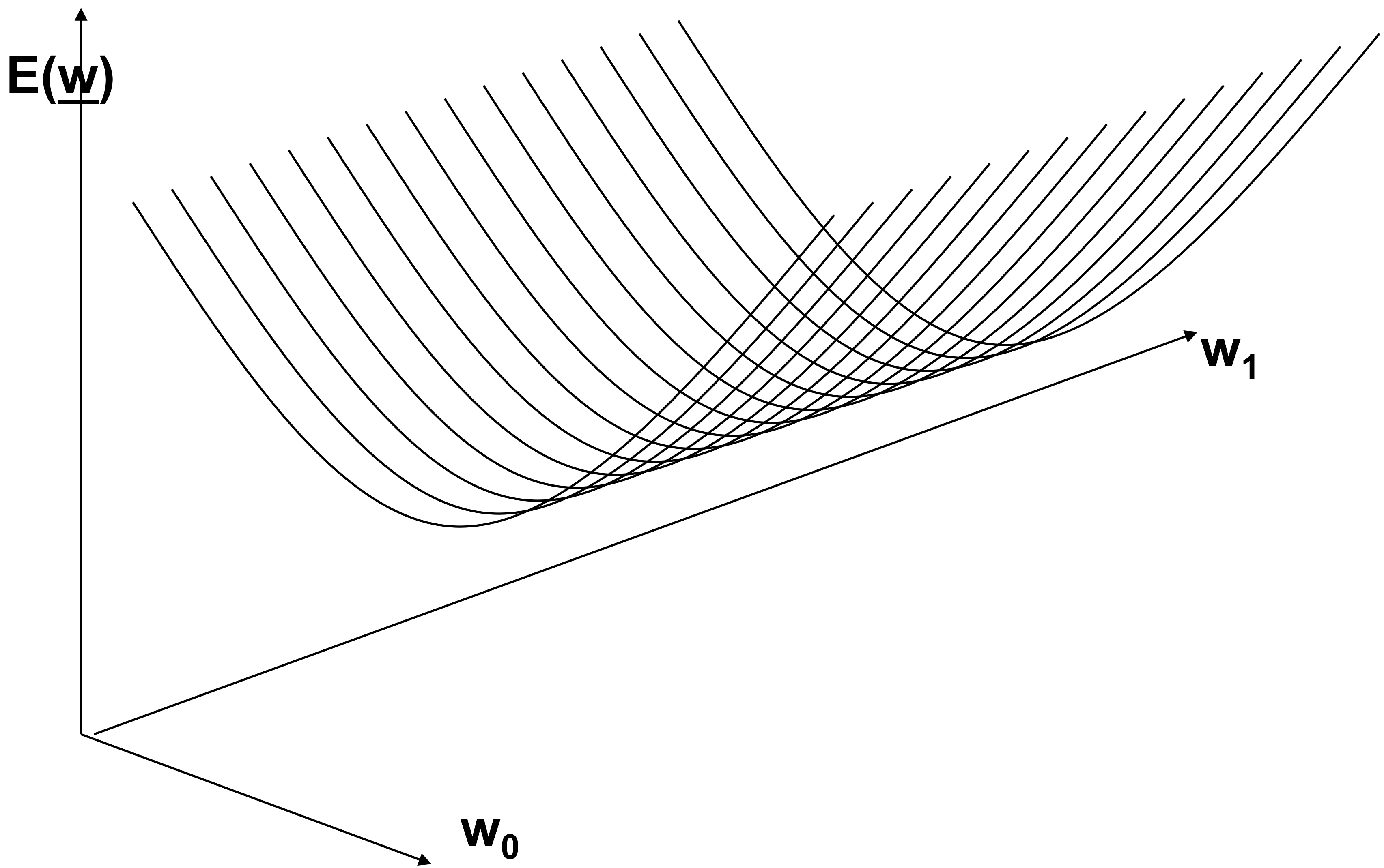


Illustration of Gradient Descent

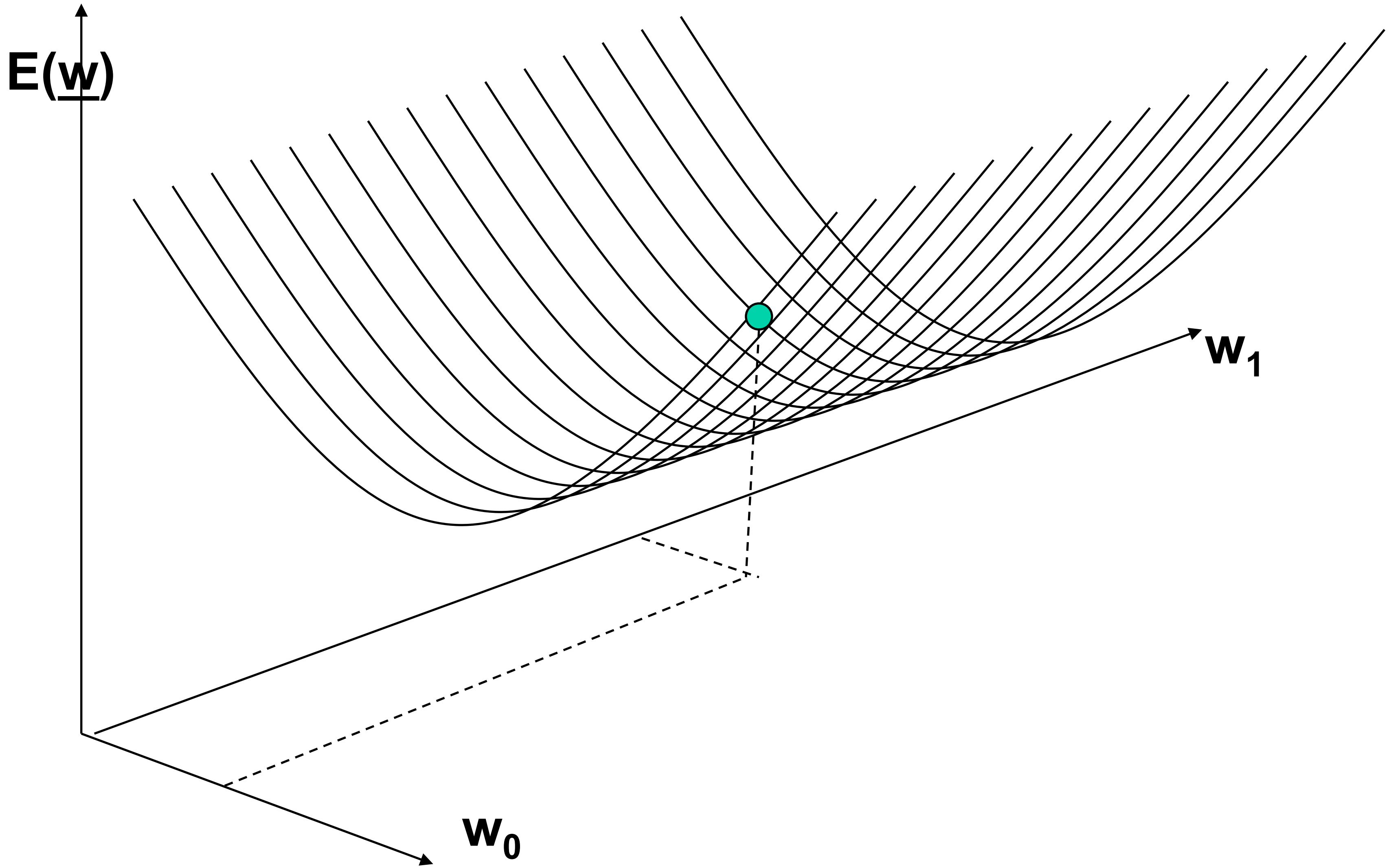
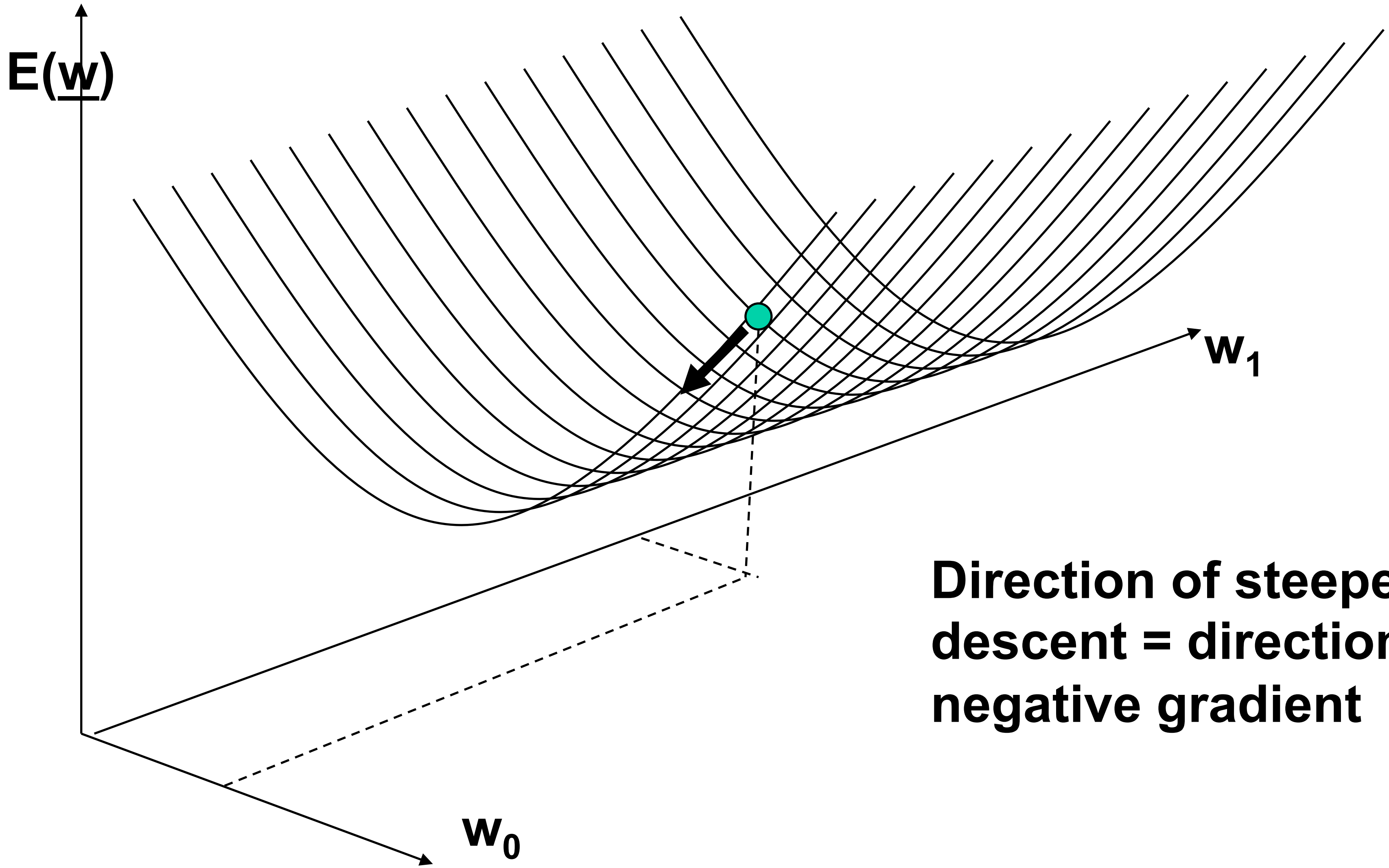
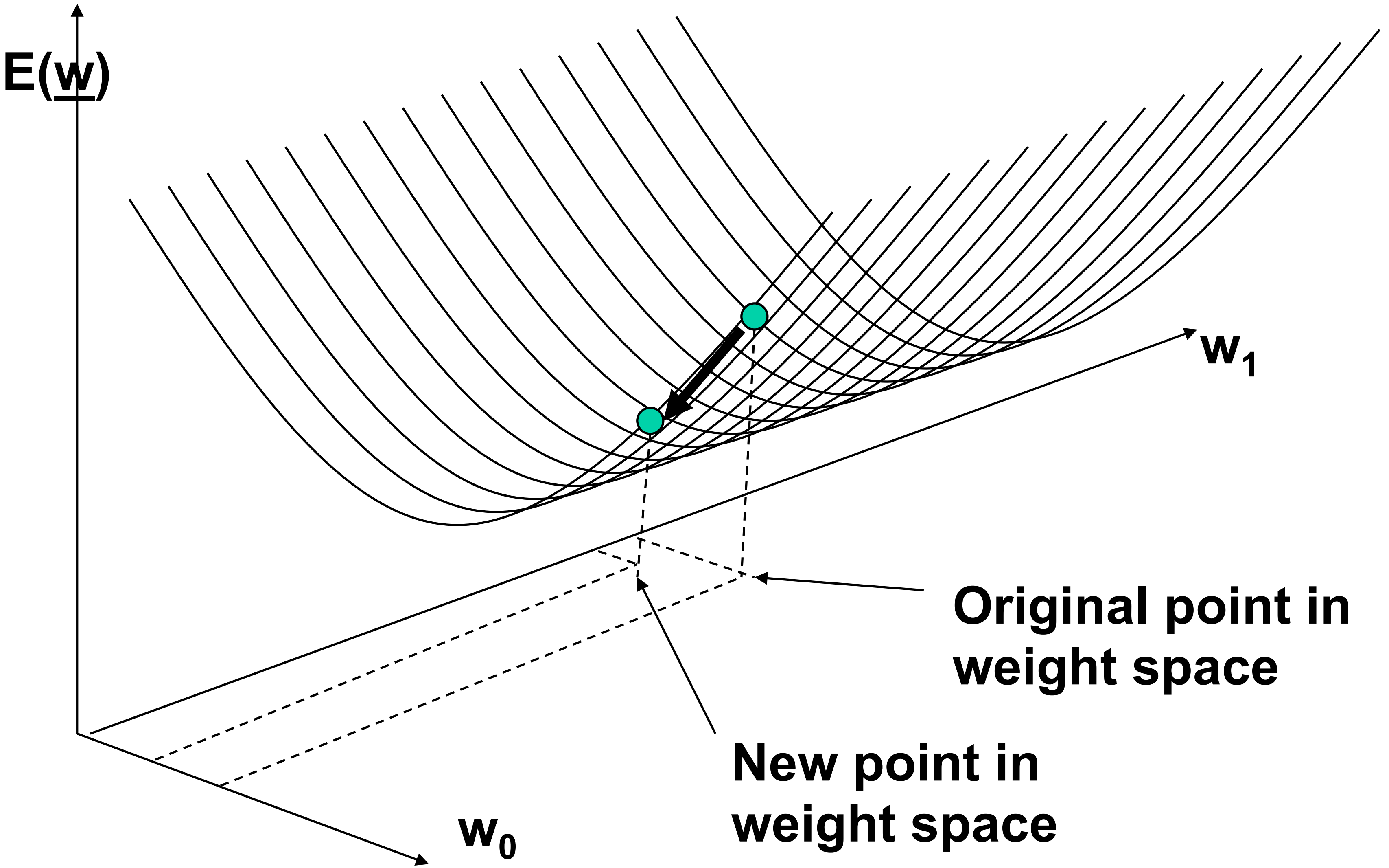


Illustration of Gradient Descent



Direction of steepest descent = direction of negative gradient

Illustration of Gradient Descent



Example: Linear Regression

- Training data: {(1,3),(2.1,.5),(-5,6.2)}

- Linear regression model: $t = w_1 x + w_0$

- The error function: $E(w) = (3 - w_1 \times 1 - w_0)^2 + (5 - w_1 \times 2.1 - w_0)^2 + (6.2 - w_1 \times (-5) - w_0)^2$

- The gradient:

$$\nabla_w E(w) = \begin{bmatrix} \dfrac{\partial E}{\partial w_0} \\ \dfrac{\partial E}{\partial w_1} \end{bmatrix} = -2 * \begin{bmatrix} (3 - w_1 - w_0) + (5 - 2.1 w_1 - w_0) + (6.2 + 5 w_1 - w_0) \\ (3 - w_1 - w_0) + 2.1(5 - 2.1 w_1 - w_0) - 5(6.2 + 5 w_1 - w_0) \end{bmatrix}$$

Summary

- We saw various techniques for supervised machine learning
  - Where we are given a labeled training data

- Classification:
  - Perceptron (parametric)
  - Naïve-Bayes (parametric)
  - K nearest neighbor (non-parametric)

- Regression:
  - Linear models based on least square errors (parametric)