# Linear models for classification

Grzegorz Chrupała

Saarland University

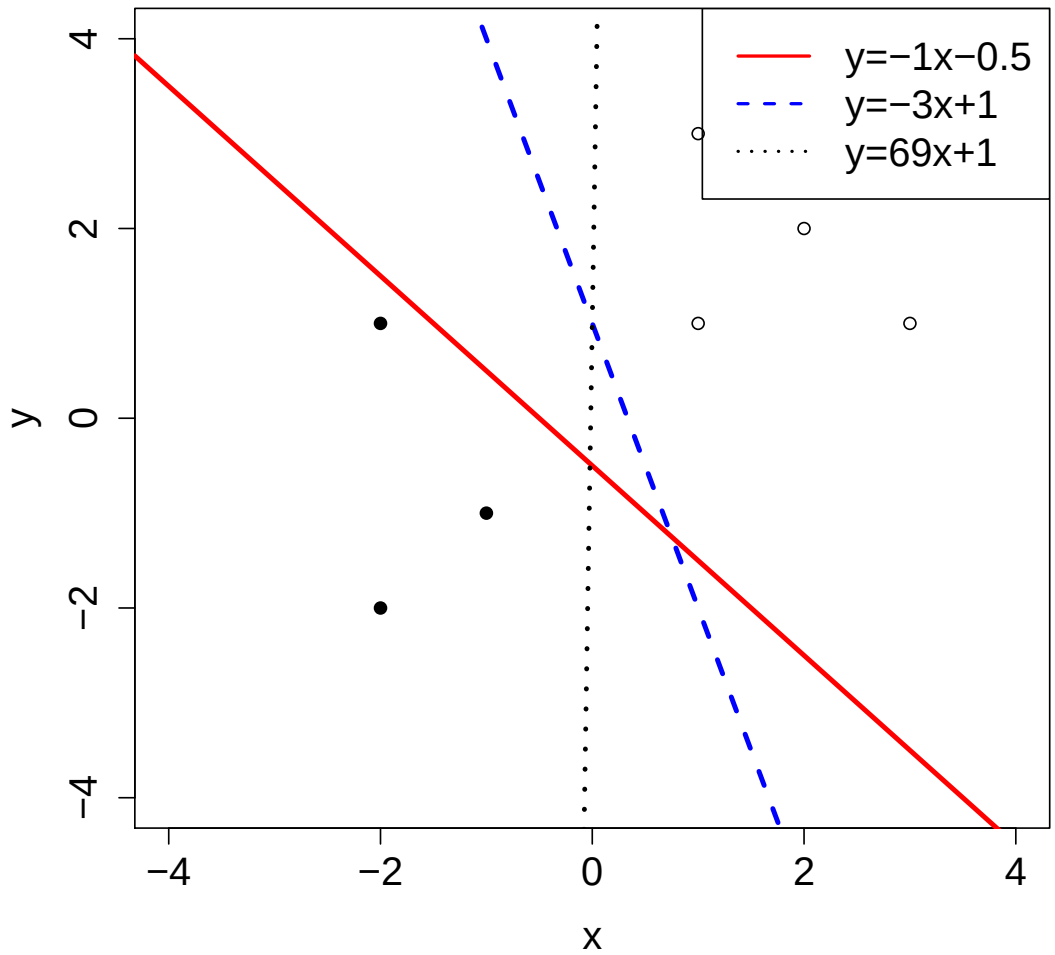
October 18, 2011

# Outline

# Outline

# An example

Positive examples are blank, negative are filled



# Linear models

Think of training examples as points in $d$-dimensional space. Each dimension corresponds to one feature.

A linear binary classifier defines a plane in the space which separates positive from negative examples.

# Linear decision boundary

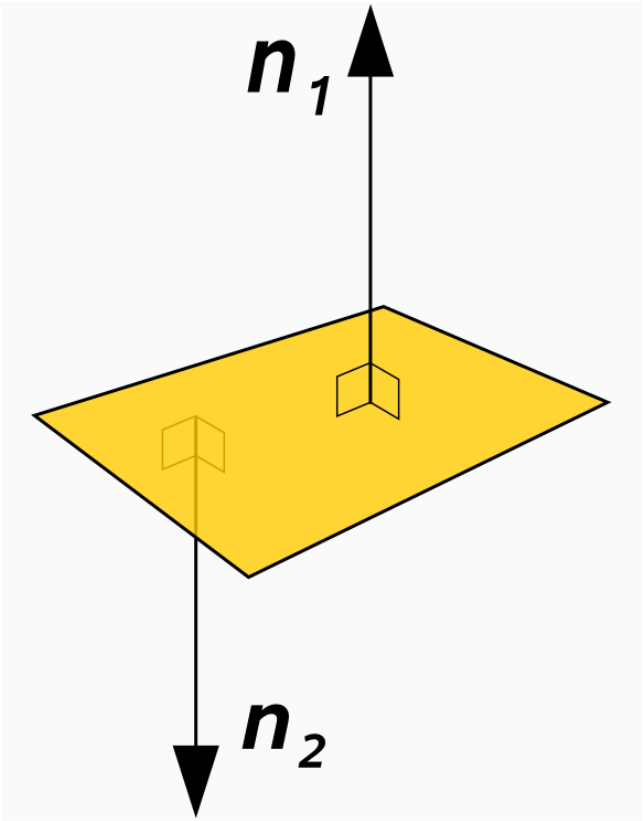
- A hyperplane is a generalization of a straight line to $> 2$ dimensions
- A hyperplane contains all the points in a $d$ dimensional space satisfying the following equation:

$$w_1 x_1 + w_2 x_2, \ldots, + w_d x_d + w_0 = 0$$

- Each coefficient $w_i$ can be thought of as a weight on the corresponding feature
- The vector containing all the weights $\mathbf{w} = (w_0, \ldots, w_d)$ is the parameter vector or weigth vector

# Normal vector

- Geometrically, the weight vector $\mathbf{w}$ is a normal vector of the separating hyperplane
- A normal vector of a surface is any vector which is perpendicular to it



# Hyperplane as a classifier

- Let

$$g(\mathbf{x}) = w_1 x_1 + w_2 x_2, \ldots, + w_d x_d + w_0$$

- Then

$$y = \text{sign}(g(\mathbf{x})) = \begin{cases} +1 & \text{if } g(\mathbf{x}) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

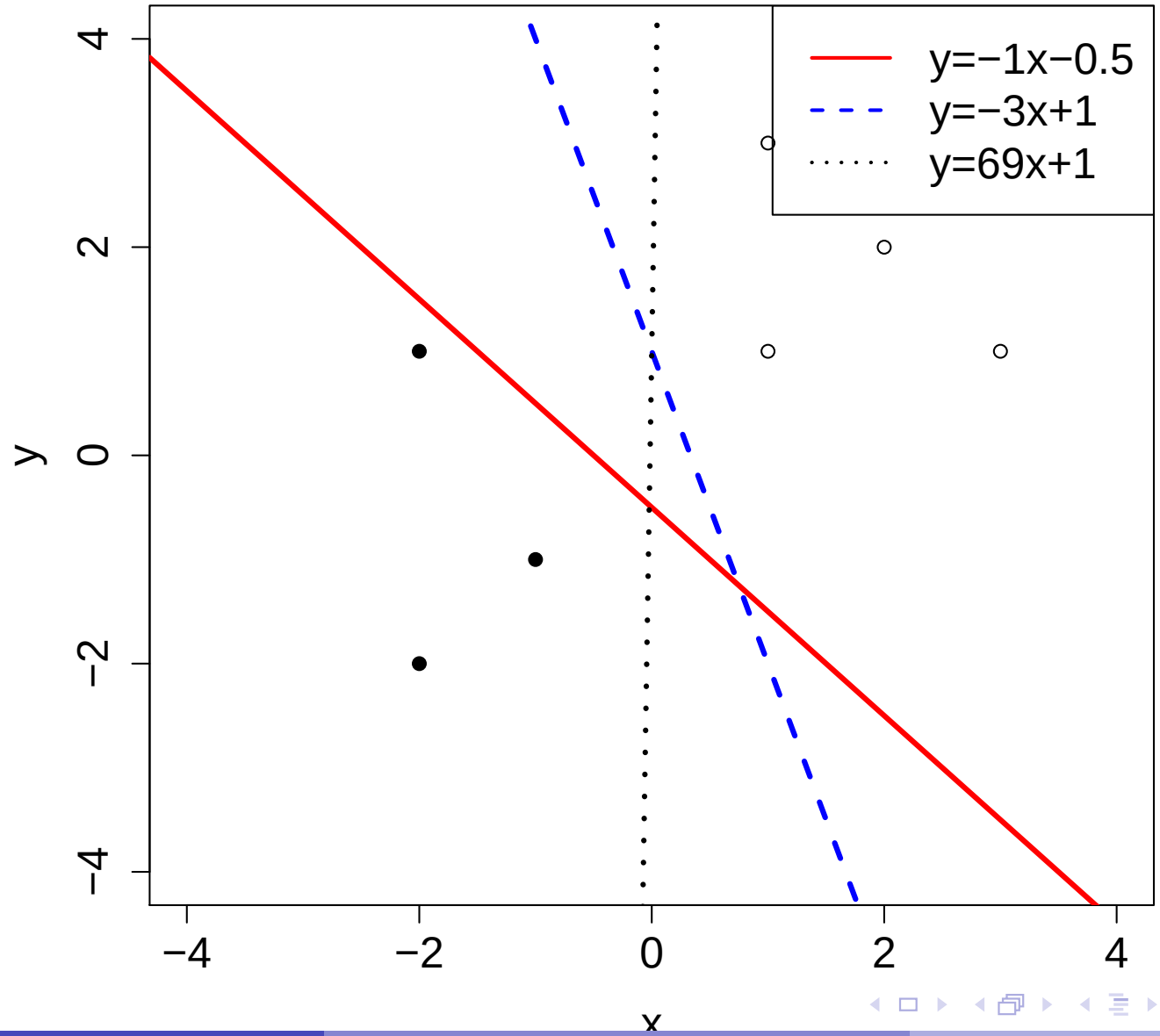
# Bias

- The slope of the hyperplane is determined by $w_1 ... w_d$. The location (intercept) is determined by bias $w_0$
- Include bias in the weight vector and add a dummy component to the feature vector
- Set this component to $x_0 = 1$
- Then

$$g(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i$$

$$g(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$$

# Separating hyperplanes in 2 dimensions



# Learning

- The goal of the learning process is to come up with a "good" weight vector $\mathbf{w}$
- The learning process will use examples to guide the search of a "good" $\mathbf{w}$
- Different notions of "goodness" exist, which yield different learning algorithms
- We will describe some of these algorithms in the following

# Outline

# Perceptron training

- How do we find a set of weights that separate our classes?
- **Perceptron**: A simple mistake-driven online algorithm

  - Start with a zero weight vector and process each training example in turn.
  - If the current weight vector classifies the current example incorrectly, move the weight vector in the right direction.
  - If weights stop changing, stop

- If examples are linearly separable, then this algorithm is guaranteed to converge to the solution vector

# Fixed increment online perceptron algorithm

- Binary classification, with classes $+1$ and $-1$
- Decision function $y' = \text{sign}(\mathbf{w} \cdot \mathbf{x})$

**PERCEPTRON**$(x^{1:N}, y^{1:N}, I)$:

1: $\mathbf{w} \leftarrow \mathbf{0}$
2: **for** $i = 1 \ldots I$ **do**
3:  **for** $n = 1 \ldots N$ **do**
4:  **if** $y^{(n)}(\mathbf{w} \cdot \mathbf{x}^{(n)}) \leq 0$ **then**
5:  $\mathbf{w} \leftarrow \mathbf{w} + y^{(n)}\mathbf{x}^{(n)}$
6: **return** $\mathbf{w}$

# Or more explicitly

1: $\mathbf{w} \leftarrow \mathbf{0}$
2: **for** $i = 1 \ldots I$ **do**
3:  **for** $n = 1 \ldots N$ **do**
4:  **if** $y^{(n)} = \text{sign}(\mathbf{w} \cdot \mathbf{x}^{(n)})$ **then**
5:  pass
6:  **else if** $y^{(n)} = +1 \wedge \text{sign}(\mathbf{w} \cdot \mathbf{x}^{(n)}) = -1$ **then**
7:  $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}^{(n)}$
8:  **else if** $y^{(n)} = -1 \wedge \text{sign}(\mathbf{w} \cdot \mathbf{x}^{(n)}) = +1$ **then**
9:  $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}^{(n)}$
10: **return** $\mathbf{w}$

# Weight averaging

- Although the algorithm is guaranteed to converge, the solution is not unique!
- Sensitive to the order in which examples are processed
- Separating the training sample does not equal good accuracy on unseen data
- Empirically, better generalization performance with weight averaging
  - A method of avoiding overfitting
  - As final weight vector, use the mean of all the weight vector values for each step of the algorithm
  - (cf. regularization in a following session)

# Outline

# Probabilistic model

- Instead of thinking in terms of multidimensional space...
- Classification can be approached as a probability estimation problem
- We will try to find a probability distribution which
  - Describes well our training data
  - Allows us to make accurate predictions
- We'll look at Naive Bayes as a simplest example of a probabilistic classifier

# Representation of examples

- We are trying to classify documents. Let's represent a document as a sequence of terms (words) it contains $\mathbf{t} = (t_1 ... t_n)$
- For (binary) classification we want to find the most probable class:

$$\hat{y} = \underset{y \in \{-1,+1\}}{\operatorname{argmax}} P(Y = y|\mathbf{t})$$

- But documents are close to unique: we cannot reliably condition $Y|\mathbf{t}$
- Bayes' rule to the rescue

# Bayes rule

Bayes rule determines how joint and conditional probabilities are related.

$$P(Y = y_i|X = x) = \frac{P(X = x|Y)P(Y = y_i)}{\sum_i P(X = x|Y = y_i)P(Y = y_i)}$$

That is:

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

# Prior and likelihood

- With Bayes' rule we can invert the direction of conditioning

$$\begin{aligned}\hat{y} &= \operatorname*{argmax}_y \frac{P(Y=y)P(\mathbf{t}|Y=y)}{\sum_y P(Y=y)P(\mathbf{t}|Y=y)} \\ &= \operatorname*{argmax}_y P(Y=y)P(\mathbf{t}|Y=y)\end{aligned}$$

- Decomposed the task into estimating the prior $P(Y)$ (easy) and the likelihood $P(\mathbf{t}|Y=y)$

# Conditional independence

- How to estimate $P(\mathbf{t}|Y = y)$?
- Naively assume the occurrence of each word in the document is independent of the others, when conditioned on the class

$$P(\mathbf{t}|Y = y) = \prod_{i=1}^{|\mathbf{t}|} P(t_i|Y = y)$$

# Naive Bayes

## Putting it all together

$$\hat{y} = \underset{y}{\operatorname{argmax}}\, P(Y = y) \prod_{i=1}^{|\mathbf{t}|} P(t_i | Y = y)$$

# Decision function

- For binary classification:

$$g(\mathbf{t}) = \frac{P(Y=+1)\prod_{i=1}^{|\mathbf{t}|} P(t_i|Y=+1)}{P(Y=-1)\prod_{i=1}^{|\mathbf{t}|} P(t_i|Y=-1)}$$

$$= \frac{P(Y=+1)}{P(Y=-1)} \prod_{i=1}^{|\mathbf{t}|} \frac{P(t_i|Y=+1)}{P(t_i|Y=-1)}$$

$$\hat{y} = \begin{cases} +1 \text{ if } g(\mathbf{t}) \geq 1 \\ -1 \text{ otherwise} \end{cases}$$

# Documents in vector notation

- Let's represent documents as vocabulary-size-dimensional binary vectors

| | $V_1$ | $V_2$ | $V_3$ | $V_4$ | |
|---|---|---|---|---|---|
| | Obama | Ferrari | voters | movies | |
| $\mathbf{x} = ($ | 1 | 0 | 2 | 0 | $)$ |

- Dimension $i$ indicates how many times the $i^{th}$ vocabulary item appears in document $\mathbf{x}$

# Naive Bayes in vector notation

- Counts appear as exponents:

$$g(\mathbf{x}) = \frac{P(+1)}{P(-1)} \prod_{i=1}^{|V|} \left( \frac{P(V_i|+1)}{P(V_i|-1)} \right)^{x_i}$$

- If we take the logarithm of the threshold ($\ln 1 = 0$) and $g$, we'll get the same decision function

$$h(\mathbf{x}) = \ln \left( \frac{P(+1)}{P(-1)} \right) + \sum_{i=1}^{|V|} \ln \left( \frac{P(V_i|+1)}{P(V_i|-1)} \right) x_i$$

# Linear classifier

- Remember the linear classifier?

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i$$

$$h(\mathbf{x}) = \ln\left(\frac{P(+1)}{P(-1)}\right) + \sum_{i=1}^{|V|} \ln\left(\frac{P(V_i|+1)}{P(V_i|-1)}\right) x_i$$

- Log prior ratio corresponds to the bias term
- Log likelihood ratios correspond to feature weights

# What is the difference

Training criterion and procedure

## Perceptron

- Zero-one loss function

$$error(\mathbf{w}, D) = \sum_{(\mathbf{x},y)\in D} \begin{cases} 0 \text{ if } \operatorname{sign}(\mathbf{w}\cdot\mathbf{x}) = y \\ 1 \text{ otherwise} \end{cases}$$

- Error-driven algorithm

# Naive Bayes

- Maximum Likelihood criterion

$$P(D|\theta) = \prod_{(\mathbf{x},y)\in D} P(Y = y|\theta)P(x|Y = y, \theta)$$

- Find parameters which maximize the log likelihood

$$\hat{\theta} = \operatorname*{argmax}_{\theta} \log(P(D|\theta))$$

  Parameters reduce to relative counts
- $+$ Ad-hoc smoothing
- Alternatives (e.g. maximum *a posteriori*)

# Comparison

| Model | Naive Bayes | Perceptron |
|---|---|---|
| Model power | Linear | Linear |
| Type | Generative | Discriminative |
| Distribution modeled | $P(\mathbf{x}, y)$ | N/A |
| Smoothing | Crucial | Optional |
| Independence assumptions | Strong | None |

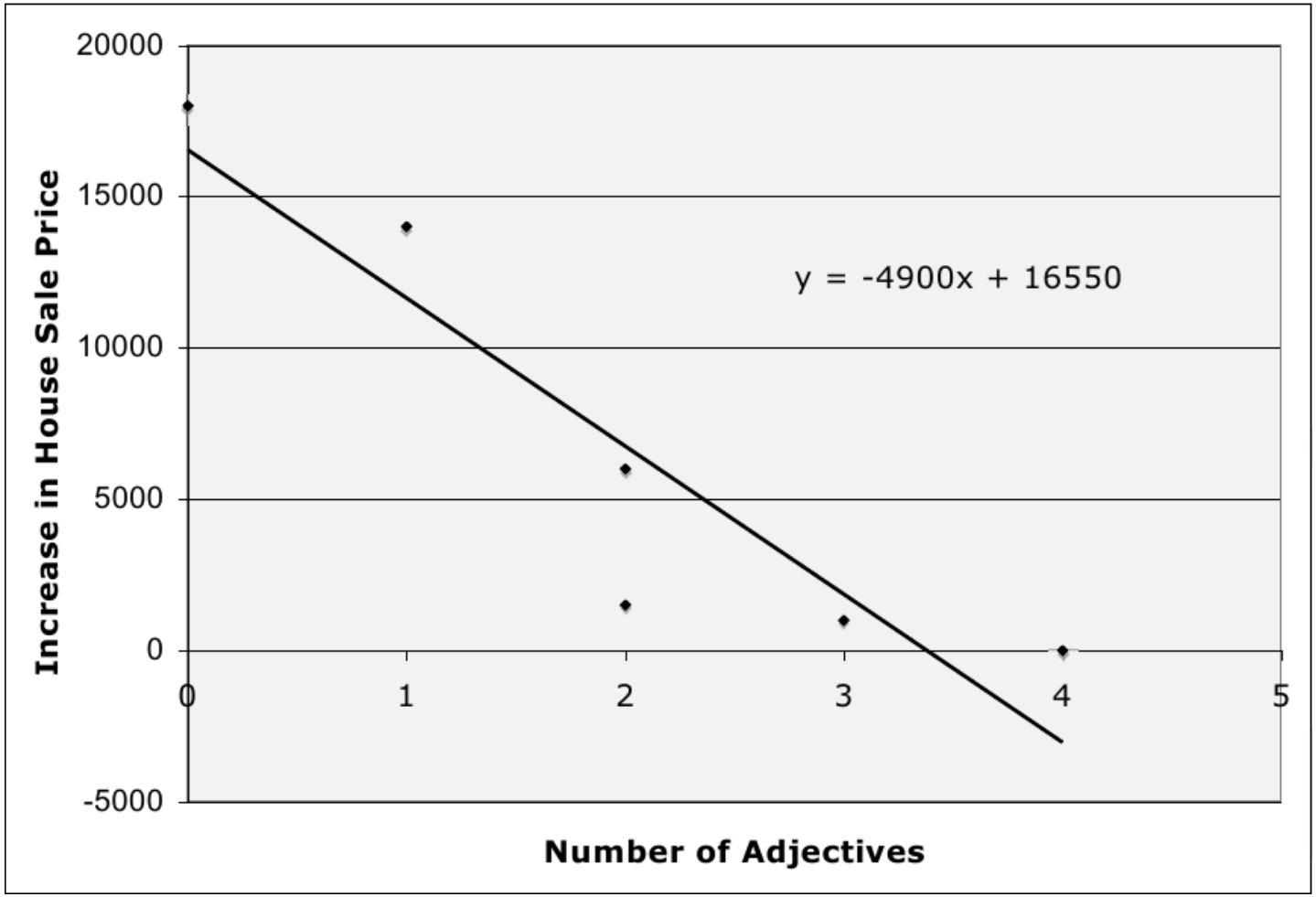
# Outline

# Probabilistic conditional model

- Let's try to come up with a probabilistic model which has some of the advantages of perceptron
- Model $P(y|\mathbf{x})$ directly, and not via $P(\mathbf{x}, y)$ and Bayes rule as in Naive Bayes
- Avoid issue of dependencies between features of $\mathbf{x}$
- We'll take linear regression as a starting point
  - The goal is to adapt regression to model class-conditional probability

# Linear Regression

- Training data: observations paired with outcomes ($n \in \mathbb{R}$)
- Observations have features (predictors, typically also real numbers)
- The model is a regression line $y = ax + b$ which best fits the observations
  - $a$ is the slope
  - $b$ is the intercept
  - This model has two parameters (or weigths)
  - One feature $= x$
  - Example:
    - $x =$ number of vague adjectives in property descriptions
    - $y =$ amount house sold over asking price


20000
15000
10000
5000
0
-5000
Increase in House Sale Price
y = -4900x + 16550
0
1
2
3
4
5
Number of Adjectives

# Multiple linear regression

- More generally $y = w_0 + \sum_{i=1}^{d} w_i x_i$, where
  - $y =$ outcome
  - $w_0 =$ intercept
  - $x_1..x_d =$ features vector and $w_1..w_d$ weight vector
  - Get rid of bias:

$$g(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

- Linear regression: uses $g(\mathbf{x})$ directly
- Linear classifier: uses $\text{sign}(g(\mathbf{x}))$

# Learning linear regression

- Minimize sum squared error over $N$ training examples

$$\text{Err}(\mathbf{w}) = \sum_{n=1}^{N} (g(\mathbf{x}^{(n)}) - y^{(n)})^2$$

- Closed-form formula for choosing the best weights $\mathbf{w}$:

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$$

where the matrix $X$ contains training example features, and $\mathbf{y}$ is the vector of outcomes.

# Logistic regression

- In logistic regression we use the linear model to assign probabilities to class labels
- For binary classification, predict $P(Y = 1|\mathbf{x})$. But predictions of linear regression model are $\in \mathbb{R}$, whereas $P(Y = 1|\mathbf{x}) \in [0, 1]$
- Instead predict logit function of the probability:

$$\ln\left(\frac{P(Y = 1|\mathbf{x})}{1 - P(Y = 1|\mathbf{x})}\right) = \mathbf{w} \cdot \mathbf{x}$$

$$\frac{P(Y = 1|\mathbf{x})}{1 - P(Y = 1|\mathbf{x})} = e^{\mathbf{w} \cdot \mathbf{x}}$$

Solving for $P(Y = 1|\mathbf{x})$ we obtain:

$$P(Y = 1|\mathbf{x}) = e^{\mathbf{w}\cdot\mathbf{x}}(1 - P(Y = 1|\mathbf{x}))$$
$$P(Y = 1|\mathbf{x}) = e^{\mathbf{w}\cdot\mathbf{x}} - e^{\mathbf{w}\cdot\mathbf{x}}P(Y = 1|\mathbf{x})$$
$$P(Y = 1|\mathbf{x}) + e^{\mathbf{w}\cdot\mathbf{x}}P(Y = 1|\mathbf{x}) = e^{\mathbf{w}\cdot\mathbf{x}}$$
$$P(Y = 1|\mathbf{x})(1 + e^{\mathbf{w}\cdot\mathbf{x}}) = e^{\mathbf{w}\cdot\mathbf{x}}$$
$$P(Y = 1|\mathbf{x}) = \frac{e^{\mathbf{w}\cdot\mathbf{x}}}{1 + e^{\mathbf{w}\cdot\mathbf{x}}}$$
$$= \frac{\exp\left(\sum_{i=0}^{d} w_i x_i\right)}{1 + \exp\left(\sum_{i=0}^{d} w_i x_i\right)}$$

# Logistic regression - classification

- Example $\mathbf{x}$ belongs to class $1$ if:

$$\frac{P(Y=1|\mathbf{x})}{1-P(Y=1|\mathbf{x})} > 1$$

$$e^{\mathbf{w}\cdot\mathbf{x}} > 1$$

$$\mathbf{w}\cdot\mathbf{x} > 0$$

$$\sum_{i=0}^{d} w_i x_i > 0$$

- Equation $\mathbf{w}\cdot\mathbf{x} = 0$ defines a hyperplane with points above belonging to class $1$

# Multinomial logistic regression

Logistic regression generalized to more than two classes

$$P(Y = y|\mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{W}_{y\bullet} \cdot \mathbf{x})}{\sum_{y'} \exp(\mathbf{W}_{y'\bullet} \cdot \mathbf{x})}$$

# Learning parameters

- Conditional likelihood estimation: choose the weights which make the probability of the observed values $y$ be the highest, given the observations $\mathbf{x}_i$
- For the training set with $N$ examples:

$$\hat{\mathbf{W}} = \operatorname*{argmax}_{\mathbf{W}} \prod_{i=1}^{N} P(Y = y^{(n)} | \mathbf{x}^{(n)}, \mathbf{W})$$

$$= \operatorname*{argmax}_{\mathbf{W}} \sum_{i=1}^{N} \log P(Y = y^{(n)} | \mathbf{x}^{(n)}, \mathbf{W})$$

# Error function

Equivalently, we seek the value of the parameters which minimize the error function:

$$\mathrm{Err}(\mathbf{W}, D) = -\sum_{n=1}^{N} \log P(Y = y^{(n)} | \mathbf{x}^{(n)}, \mathbf{W})$$

where $D = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$

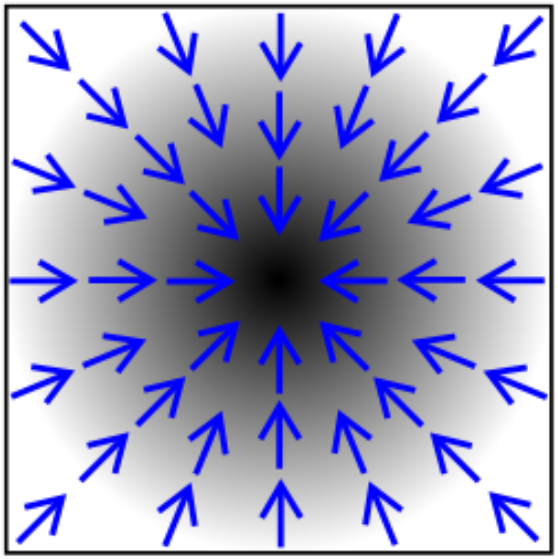
# A problem in convex optimization

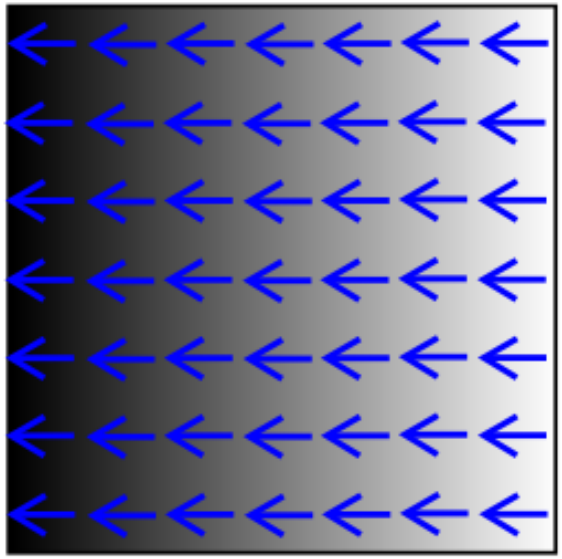
- L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno method)
- gradient descent
- conjugate gradient
- iterative scaling algorithms

# Stochastic gradient descent

## Gradient descent

- A gradient is a slope of a function
- That is, a set of partial derivatives, one for each dimension (parameter)
- By following the gradient of a convex function we can descend to the bottom (minimum)





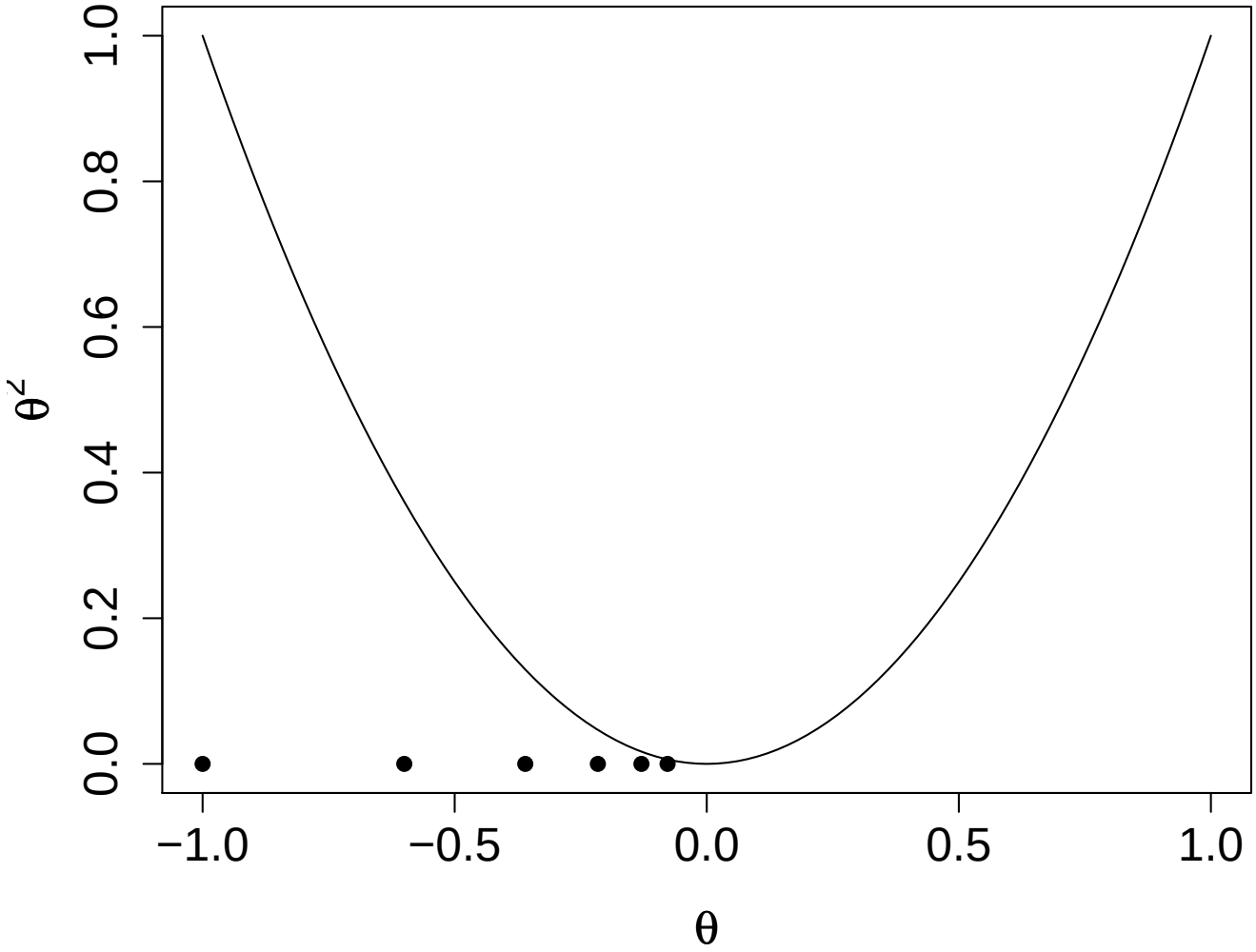
# Gradient descent example

- Find $\operatorname{argmin}_\theta f(\theta)$ where $f(\theta) = \theta^2$
- Initial value of $\theta_1 = -1$
- Gradient function: $\nabla f(\theta) = 2\theta$
- Update: $\theta^{(n+1)} = \theta^{(n)} - \eta \nabla f(\theta^{(n)})$
- The learning rate $\eta$ $(= 0.2)$ controls the speed of the descent

# Gradient descent example

- Find $\operatorname{argmin}_\theta f(\theta)$ where $f(\theta) = \theta^2$
- Initial value of $\theta_1 = -1$
- Gradient function: $\nabla f(\theta) = 2\theta$
- Update: $\theta^{(n+1)} = \theta^{(n)} - \eta \nabla f(\theta^{(n)})$
- The learning rate $\eta$ $(= 0.2)$ controls the speed of the descent
- After first iteration: $\theta^{(2)} = -1 - 0.2(-2) = -0.6$

# Five iterations of gradient descent



# Stochastic

- We could compute the gradient of error for the full dataset before each update
- Instead
  - Compute the gradient of the error for a single example
  - update the weight
  - Move on to the next example
- On average, we'll move in the right direction
- Efficient, online algorithm

# Error gradient

- The gradient of the error function is the set of partial derivatives of the error function with respect to the parameters $\mathbf{W}_{yi}$

$$\nabla_{y,i}\mathrm{Err}(D,\mathbf{W}) = \frac{\partial}{\partial \mathbf{W}_{yi}}\left(-\sum_{n=1}^{N}\log P(Y=y|\mathbf{x}^{(n)},\mathbf{W})\right)$$
$$= -\sum_{n=1}^{N}\frac{\partial}{\partial \mathbf{W}_{yi}}\log P(Y=y|\mathbf{x}^{(n)},\mathbf{W})$$

# Single training example

$$\frac{\partial}{\partial \mathbf{W}_{yi}} \log P(Y = y|\mathbf{x}^{(n)}, \mathbf{W}) = \frac{\partial}{\partial \mathbf{W}_{yi}} \log P(Y = y|\mathbf{x}^{(n)}, \mathbf{W})$$

$$= \frac{\partial}{\partial \mathbf{W}_{yi}} \log \frac{\exp(\mathbf{W}_{y\bullet} \cdot \mathbf{x}^{(n)})}{\sum_{y'} \exp(\mathbf{W}_{y'\bullet} \cdot \mathbf{x}^{(n)})}$$

$$\ldots$$

$$\ldots$$

$$= x_i^{(n)}(\mathbf{1}(y = y^{(n)}) - P(Y = y|\mathbf{x}^{(n)}, \mathbf{W}))$$

# Update

## Stochastic gradient update step

$$\mathbf{W}_{yi}^{(n)} = \mathbf{W}_{yi}^{(n-1)} + \eta x_i^{(n)} (\mathbf{1}(y = y^{(n)}) - P(Y = y|\mathbf{x}^{(n)}, \mathbf{W}))$$

# Update: Explicit

- For the correct class $(y = y^{(n)})$

$$\mathbf{W}_{yi}^{(n)} = \mathbf{W}_{yi}^{(n-1)} + \eta x_i^{(n)}(1 - P(Y = y|\mathbf{x}^{(n)}, \mathbf{W}))$$

where $1 - P(Y = y|\mathbf{x}^{(n)}, \mathbf{W})$ is the residual

- For all other classes $(y \neq y^{(n)})$

$$\mathbf{W}_{yi}^{(n)} = \mathbf{W}_{yi}^{(n-1)} - \eta x_i^{(n)} P(Y = y|\mathbf{x}^{(n)}, \mathbf{W})$$

# Logistics Regression SGD vs Perceptron

$$\mathbf{w}^{(n)} = \mathbf{w}^{(n-1)} + 1\mathbf{x}^{(n)}$$
$$\mathbf{W}_{yi}^{(n)} = \mathbf{W}_{yi}^{(n-1)} + \eta x_i^{(n)} \quad (1 - P(Y = y|\mathbf{x}^{(n)}, \mathbf{W}))$$

- Very similar update!
- Perceptron is simply an instantiation of SGD for a particular error function
- The perceptron criterion: for a correctly classified example zero error; for a misclassified example $-y^{(n)}\mathbf{w} \cdot \mathbf{x}^{(n)}$

# Maximum entropy

Multinomial logistic regression model is also known as the Maximum entropy model. What is the connection?

# Entia non sunt multiplicanda praeter necessitatem

# Maximum Entropy principle

## Jaynes, 1957

... in making inferences on the basis of partial information we must use that probability distribution which has maximum entropy subject to whatever is known. This is the only unbiased assignment we can make; to use any other would amount to arbitrary assumption of information which we by hypothesis do not have.

# Entropy

- Out of all possible models, choose the simplest one consistent with the data
- Entropy of the distribution of $X$:

$$H(X) = -\sum_{x} P(X = x) \log_2 P(X = x)$$

- The uniform distribution has the highest entropy

- Finding the maximum entropy distribution:

$$p* = \underset{p \in C}{\operatorname{argmax}} H(p)$$

- Berger et al. (1996) showed that solving this problem is equivalent to finding the multinomial logistic regression model whose weights maximize the likelihood of the training data.

# Maxent principle – simple example

- Find a Maximum Entropy probability distribution $p(a, b)$ where $a \in \{x, y\}$ and $b \in \{0, 1\}$
- The only thing we know are is the following constraint: $p(x, 0) + p(y, 0) = 0.6$

| $p(a, b)$ | 0 | 1 | |
|---|---|---|---|
| $x$ | ? | ? | |
| $y$ | ? | ? | |
| total | 0.6 | | 1 |

# Maxent principle – simple example

- Find a Maximum Entropy probability distribution $p(a, b)$ where $a \in \{x, y\}$ and $b \in \{0, 1\}$
- The only thing we know are is the following constraint: $p(x, 0) + p(y, 0) = 0.6$

| $p(a,b)$ | 0 | 1 | |
|---|---|---|---|
| $x$ | ? | ? | |
| $y$ | ? | ? | |
| total | 0.6 | | 1 |

| $p(a,b)$ | 0 | 1 | |
|---|---|---|---|
| $x$ | 0.3 | 0.2 | |
| $y$ | 0.3 | 0.2 | |
| total | 0.6 | | 1 |

# Comparison

| Model | Naive Bayes | Perceptron | Log. regression |
|---|---|---|---|
| Model power | Linear | Linear | Linear |
| Type | Generative | Discriminative | Discriminative |
| Distribution | $P(\mathbf{x}, y)$ | N/A | $P(y\|\mathbf{x})$ |
| Smoothing | Crucial | Optional[1] | Optional[1] |
| Independence | Strong | None | None |

[1] Aka regularization

# The end

# Efficient averaged perceptron algorithm

**PERCEPTRON**$(x^{1:N}, y^{1:N}, I)$:

$\mathbf{w} \leftarrow \mathbf{0}$ ; $\mathbf{w_a} \leftarrow \mathbf{0}$
$b \leftarrow 0$ ; $b_a \leftarrow 0$
$c \leftarrow 1$
**for** $i = 1 \ldots I$ **do**
  **for** $n = 1 \ldots N$ **do**
    **if** $y^{(n)}(\mathbf{w} \cdot \mathbf{x}^{(n)} + b) \leq 0$ **then**
      $\mathbf{w} \leftarrow \mathbf{w} + y^{(n)}\mathbf{x}^{(n)}$ ; $b \leftarrow b + y^{(n)}$
      $\mathbf{w_a} \leftarrow \mathbf{w_a} + cy^{(n)}\mathbf{x}^{(n)}$ ; $b_a \leftarrow b_a + cy^{(n)}$
    $c \leftarrow c + 1$
**return** $(\mathbf{w} - \mathbf{w_a}/c, b - b_a/c)$