U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

LIACC

M.EIC

Natural Language Processing

Henrique Lopes Cardoso

FEUP / LIACC

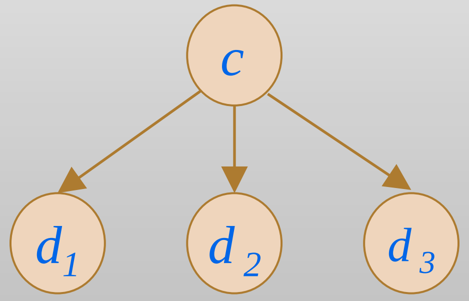
hlc@fe.up.pt

Logistic Regression

aka Maximum Entropy

Generative vs Discriminative Classifiers

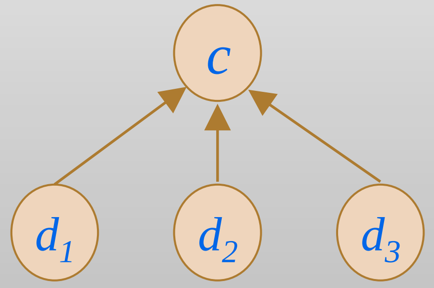
- A generative model makes use of a likelihood term: how to generate the features of a document if we knew it was of class c?



$$\hat{c} = \underset{c\in C}{\operatorname{argmax}} \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

- Naïve Bayes, Hidden Markov Models, …

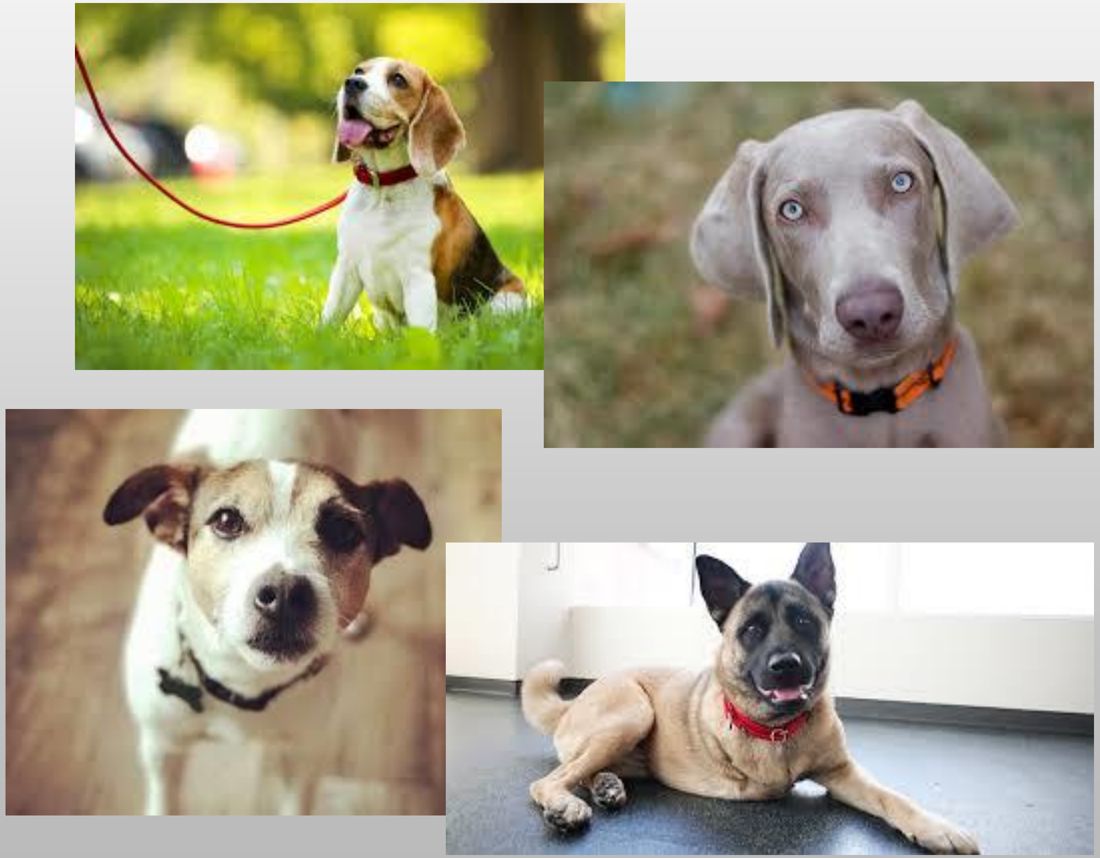
- A discriminative model tries to learn to distinguish the classes, and attempts to directly compute $P(c|d)$



$$\hat{c} = \underset{c\in C}{\operatorname{argmax}} \overbrace{P(c|d)}^{\text{posterior}}$$

- Logistic Regression, Decision Trees, Support Vector Machines, Neural Networks, Conditional Random Fields, …

Generative vs Discriminative Classifiers



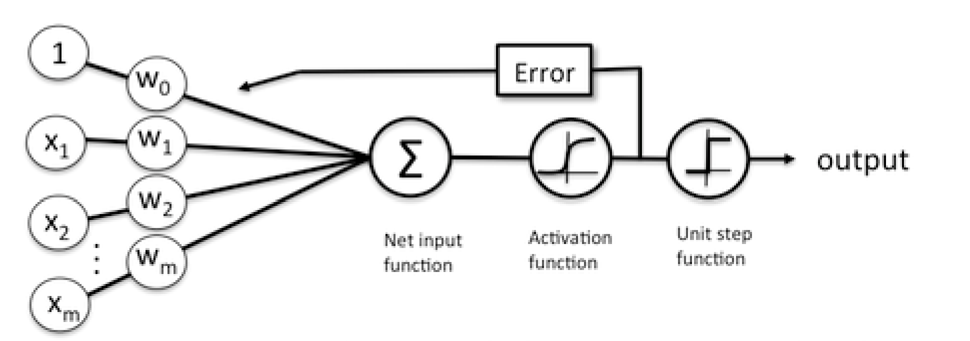
Generative vs Discriminative Classifiers

- Naïve Bayes has overly strong conditional independence assumptions
  - Edge case: two strongly correlated features, e.g. using the same feature twice
    - NB treats both copies of the feature as if they were separate
  - If multiple features tell mostly the same thing, such evidence is overestimated
- Discriminative classifiers (e.g. Logistic Regression) assign more accurate probabilities when there are many correlated features
- Naïve Bayes is easy to implement and very fast to train (there is no optimization step)
- Logistic Regression generally works better on larger documents or datasets

Why Logistic Regression?

- In NLP, Logistic Regression is a baseline supervised machine learning algorithm for classification

- Logistic Regression has a very close relationship with Neural Networks
  - a NN can be viewed as a series of logistic regression classifiers stacked on top of each other



Schematic of a logistic regression classifier.

https://www.datasciencecentral.com/logistic-regression-as-a-neural-network/

Components of a (Probabilistic) Classifier

1. A feature representation of the input
   - Given a document d, represent it as a feature vector $x = [x_1, x_2, \ldots, x_n]$
2. A classification function (e.g. sigmoid, softmax)
   - Compute the estimated class $\hat{y}$ from $p(y|x)$
3. An objective function (e.g. cross-entropy loss)
   - For learning through error minimization on training examples
4. An algorithm for optimizing the objective function (e.g. stochastic gradient descent (SGD))
   - By adjusting the model parameters (weights assigned to features)

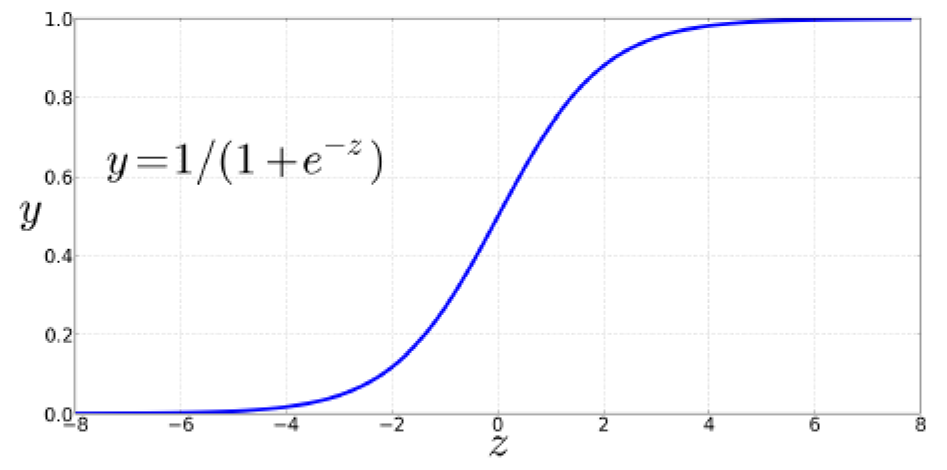
The Sigmoid Function

- Each feature x_i has an associated weight w_i
  - $w_i > 0$: feature i is associated with the class
  - $w_i < 0$: feature i is not associated
- A bias term is added to the weighted inputs

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b \qquad z = w \cdot x + b$$

- Sigmoid (or logistic) function maps z to the range $[0,1]$ (a probability)

$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$



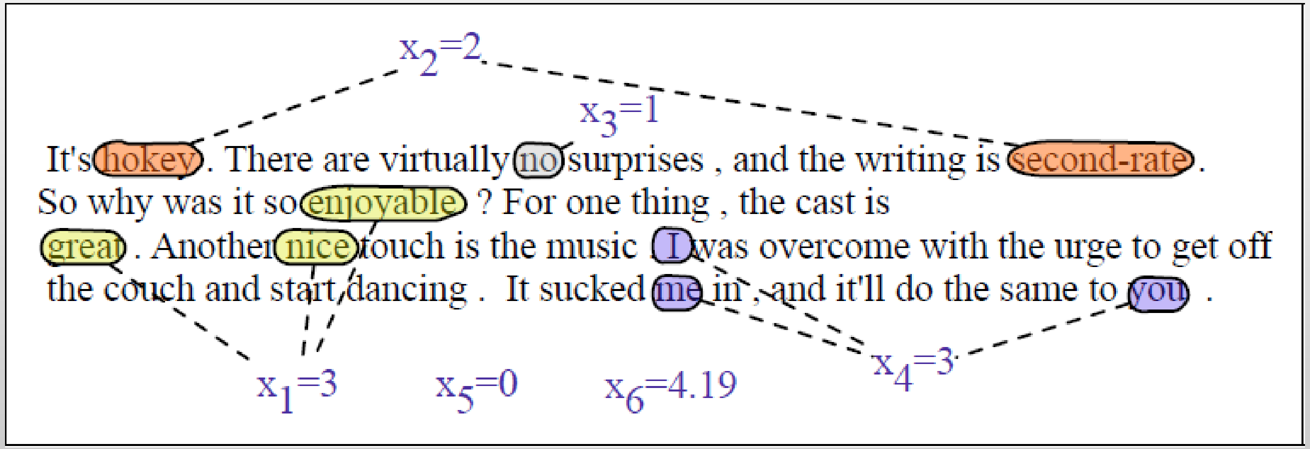
- Decision boundary:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Example: Sentiment Classification

Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

$w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$
$b = 0.1$



$$
\begin{aligned}
p(+|x) = P(y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= \sigma(.833) \\
&= 0.70 \\
p(-|x) = P(y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.30
\end{aligned}
$$

Standardizing Input Features

- Rescaling feature values so that they have comparable ranges
- Why?
  - Improved convergence (speed and stability), in particular when using gradient-based classifiers
  - Better comparison of feature importance
  - More effective distance calculations (for methods such as kNN)
  - Improved model generalization

- Z-score: mean μ_i, standard deviation σ_i

$$\mathbf{x}_i' = \frac{\mathbf{x}_i - \mu_i}{\sigma_i}$$

- Normalization: [0, 1]

$$\mathbf{x}_i' = \frac{\mathbf{x}_i - \min(\mathbf{x}_i)}{\max(\mathbf{x}_i) - \min(\mathbf{x}_i)}$$

Cross-entropy Loss

- Loss: $L(\hat{y}, y)$
  - Given $\hat{y} = \sigma(w \cdot x + b)$, how close are we from the correct output y?
- Classifier probability:

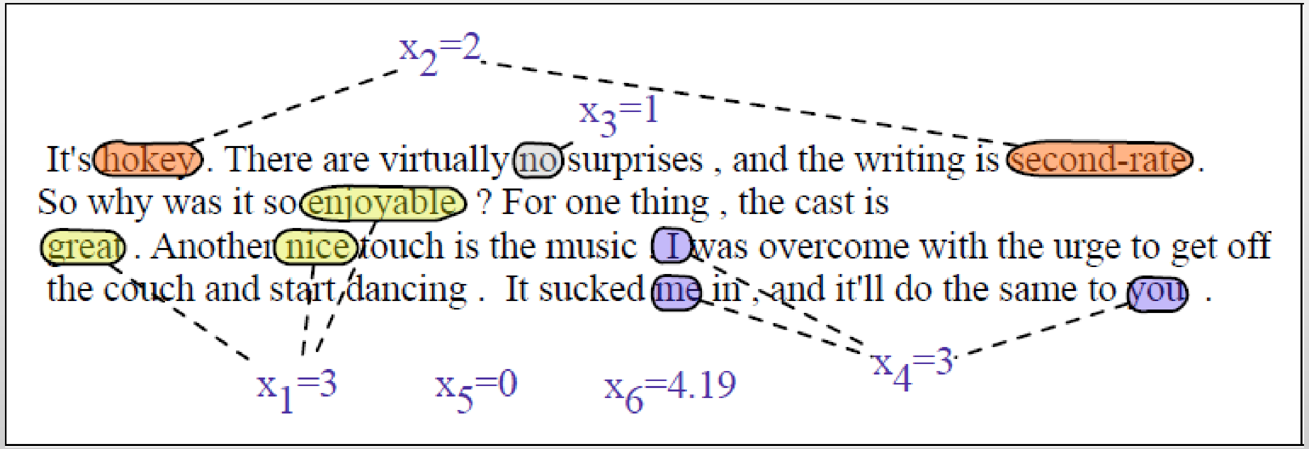
$$p(y|x) = \hat{y}^{y}(1-\hat{y})^{1-y} \longrightarrow \begin{aligned} \log p(y|x) &= \log\left[\hat{y}^{y}(1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y}) \end{aligned}$$

  - We want to maximize the probability of the correct label
    - If $y = 1$, $\mathrm{p}(y|x) = \hat{y}$; if $y = 0$, $\mathrm{p}(y|x) = 1 - \hat{y}$
- Cross-entropy loss (to minimize):

$$L_{\mathrm{CE}}(\hat{y}, y) = -\log p(y|x) = -[y\log\hat{y} + (1-y)\log(1-\hat{y})] = -[y\log\sigma(w\cdot x+b) + (1-y)\log(1-\sigma(w\cdot x+b))]$$

  - From $-\log(1) = 0$ to $-\log(0) = \infty$

Cross-entropy Loss



$$
\begin{aligned}
p(+|x) = P(y=1|x) &= \sigma(w\cdot x+b)\\
&= \sigma([2.5,-5.0,-1.2,0.5,2.0,0.7]\cdot[3,2,1,3,0,4.19]+0.1)\\
&= \sigma(.833)\\
&= 0.70\\
p(-|x) = P(y=0|x) &= 1-\sigma(w\cdot x+b)\\
&= 0.30
\end{aligned}
$$

- We want the loss to be:
  - smaller if the model estimate is close to correct
  - bigger if the model is confused
- If $y = 1$:
  - $L = -[\log(0.70)] = 0.36$
- If $y = 0$:
  - $L = -[\log(0.30)] = 1.20$

Gradient Descent

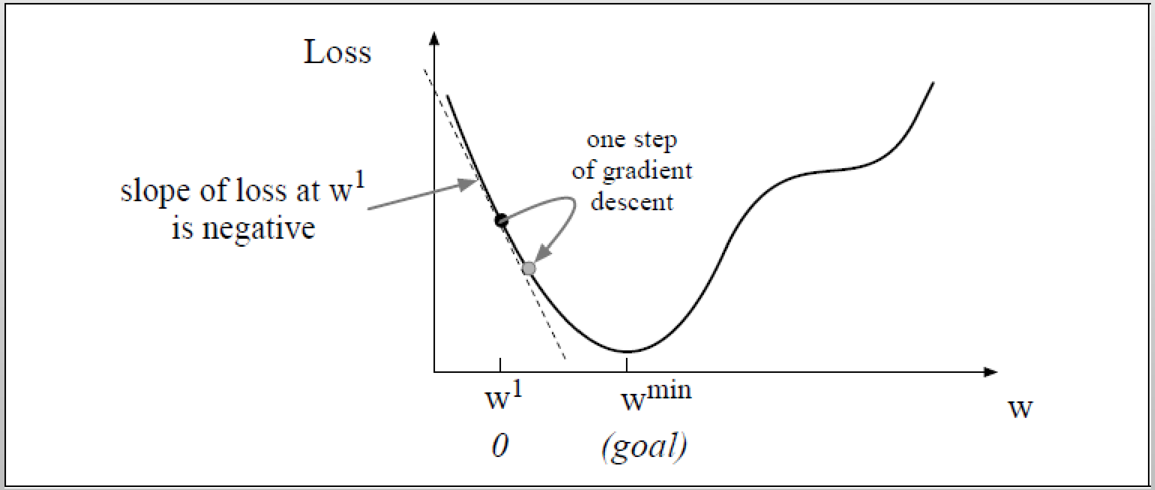
- Optimizing the objective function = finding the optimal weights to minimize the loss function, averaged over all examples

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)};\theta), y^{(i)})$$

- Gradient descent
    - Figure out the highest slope (the gradient of the loss function at the current point)
    - Move downhill in the opposite direction

Gradient Descent

- In logistic regression the loss function is convex
- Considering a single weight:



- If slope is negative, move positive!

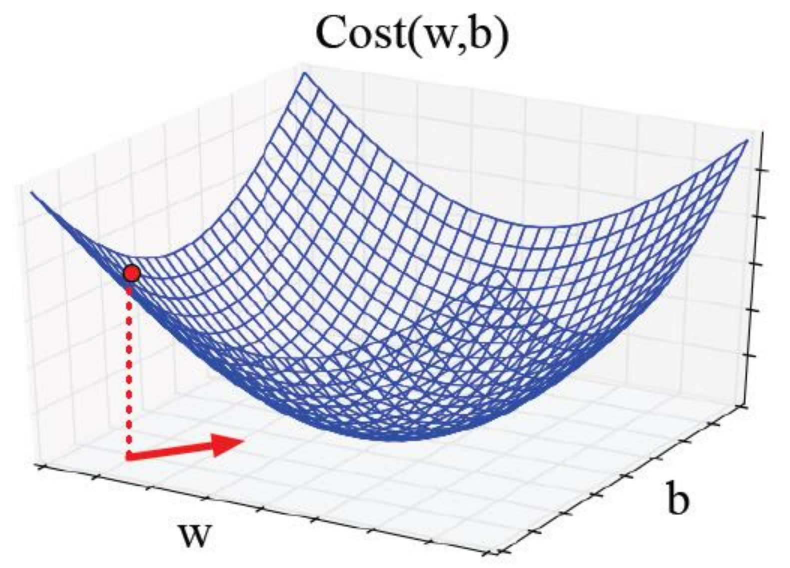
- Updating the weight:

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w), y)$$

  - where η is the learning rate

Gradient Descent

- 2 dimensions



  - The gradient vector has two dimensions, shown in the w-b plane

- N-dimensional space
  - In each dimension w_i, the slope is a partial derivative of the loss function:

$$\nabla_{\theta} L(f(x;\theta),y) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

  - Update:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta),y)$$

U.PORTO FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

LIACC

Gradient for Logistic Regression

- Cross-entropy loss:

$$L_{\mathrm{CE}}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1 - y) \log (1 - \sigma(w \cdot x + b))]$$

- Derivative:

$$\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

  - Intuitive value: gradient w.r.t. w_j is the difference between the true y and $\hat{y} = \sigma(w \cdot x + b)$ multiplied by the input x_j

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

LIACC

Stochastic Gradient Descent Algorithm

- Online algorithm that minimizes the loss function by computing its gradient after each training example

function STOCHASTIC GRADIENT DESCENT($L(), f(), x, y$) **returns** θ
  # where: L is the loss function
  #   f is a function parameterized by θ
  #   x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$
  #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$

$\theta \leftarrow 0$
repeat til done
  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
    1. Optional (for reporting): # How are we doing on this tuple?
       Compute $\hat{y}^{(i)} = f(x^{(i)};\theta)$ # What is our estimated output $\hat{y}$?
       Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
    2. $g \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$ # How should we move θ to maximize loss?
    3. $\theta \leftarrow \theta - \eta g$ # Go the other way instead
return θ

- Learning rate η (hyperparameter)
  - It set too high, the learner will take big steps and possibly overshoot the minimum
  - If set too low, the learner will take too long
- Setting a learning rate schedule η_k
  - A function of the training iteration k
  - Start at a higher value, and slowly decrease it

Example

(simplified sentiment classification)

- True value $y = 1$
- Features: $x_1 = 3$ (count of positive lexicon words); $x_2 = 2$ (count of negative lexicon words)
- Parameters: $w_1 = w_2 = b = 0$ Learning rate: $\eta = 0.1$
- Gradient:

$$\nabla_{w,b}L = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

- Update step:

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Batch and Mini-batch Training

- Batch training: compute the gradient over the entire dataset
- Mini-batch training: train on a group of *m* examples, less than the whole dataset
  - Computational efficiency: mini-batches can be vectorized for parallel processing – choose the size of the minibatch based on computational resources
- Cost function (average loss for each example):

$$Cost(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(\hat{y}^{(i)}, y^{(i)})$$

- Mini-batch gradient (average of the individual gradients):

$$\frac{\partial Cost(\hat{y}, y)}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \left[\sigma(w \cdot x^{(i)} + b) - y^{(i)} \right] x_j^{(i)}$$

Regularization

- Overfitting: giving importance to features that only accidentally correlate with the class
  - An overfitted model will not generalize well to test data
    - 4-grams in small dataset will tend to memorize the training set, but will probably be useless in the test set
    - Features not really related with the target concept, but which accidently occur in the training data

- Regularization: a technique to avoid overfitting by penalizing excessive feature weights

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)};\theta), y^{(i)}) + \alpha R(\theta)$$

regularization term

Regularization

- L2 regularization

$$R(\theta) \;=\; ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

  - Uses the square of the L2 norm $\|\theta\|_2$ of the weight values
    - L2 norm is the Euclidean distance of θ from the origin
  - Prefers weight vectors with many small weights

- L1 regularization

$$R(\theta) \;=\; ||\theta||_1 = \sum_{i=1}^{n} |\theta_i|$$

  - Uses the L1 norm $\|\theta\|_1$ of the weight values
    - Sum of absolute weight values: Manhattan distance
  - Prefers sparse vectors with some larger weights but many weights set to zero
    - Sparser weight vectors = fewer features

Multinomial Logistic Regression

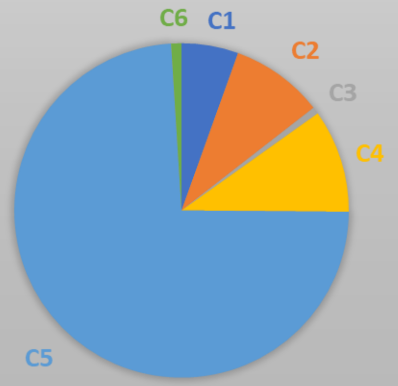
- More than two classes: we want to know the probability of each potential class
- Softmax can be seen as a generalization of the sigmoid function
  - Take a vector of k arbitrary values and map them to probabilities that sum up to 1

$$\operatorname{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

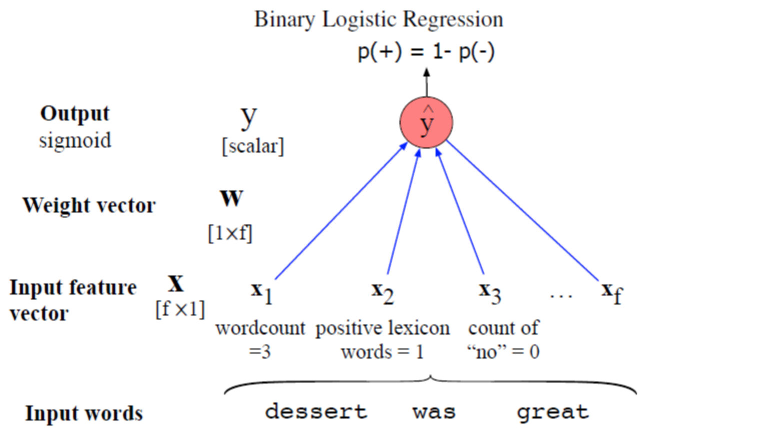
$$\operatorname{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \le i \le k$$

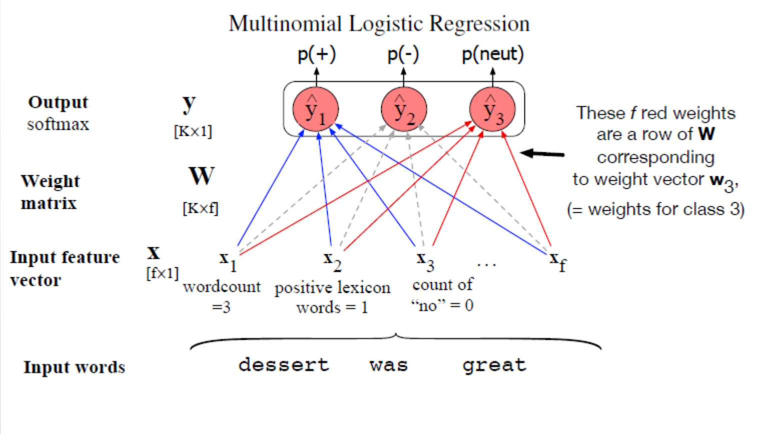
- $z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1] \rightarrow [0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$

C1 C2 C3 C4 C5 C6



Multinomial Logistic Regression





- Need separate weight vectors for each of the classes
  - In binary classification, a positive (negative) weight for a feature influences the classifier toward $y = 1$ ($y = 0$), and its absolute value indicates how important the feature is
  - In multinomial logistic regression, a feature can be evidence for or against each individual class

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	−5.3

Multinomial Logistic Regression Loss

- Classifier probability assigned to each class using softmax (sum up to 1)

$$p(y=c|x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)}$$

- Loss function

$$\begin{aligned} L_{\mathrm{CE}}(\hat{y}, y) &= -\log \hat{y}_k, \quad \text{(where } k \text{ is the correct class)} \\ &= -\log \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)} \end{aligned}$$

  - The cross-entropy loss is simply the log of the output probability corresponding to the correct class

The Python Notebook

regularization-sgd_.ipynb

- Regularization: L1 and L2
- SGD: loss, regularization
- Mini-batch training

https://scikit-learn.org/