Linear Classifiers:
Naïve Bayes, Logistic Regression

CS 6120 Natural Language Processing

Northeastern University

Si Wu

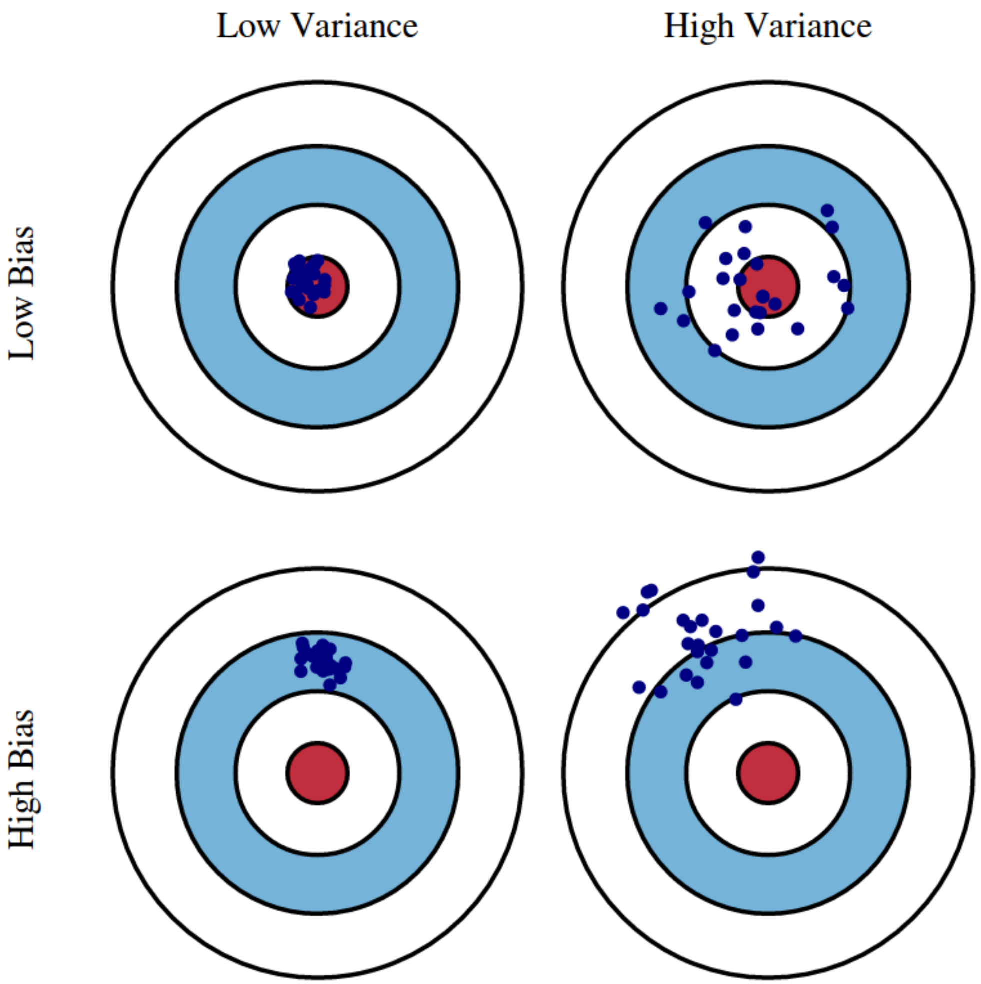
Logistics

- The first coding assignment was released.
  - You can find it on the class website under the syllabus
  - Due next Friday 11:59pm on Gradescope
  - Gradescope: make sure you select the corresponding pages for each question
- We just added the late policy for coding assignments on the course website.
  - The late policy doesn't apply to the in-class quizzes.
- Increased total seats from 59 to 64 for this session to accommodate a few students on the waitlist.
  - Watch out for emails for enrollment if you are on the waitlist

Review

- Last time we learned:
  - Regular expressions
  - Zipf’s law (rank & frequency)
  - Conditional probability and chain rule
  - First and second order Markov assumption
  - Naïve bayes, prior, likelihood, posterior
  - Maximum likelihood
  - Smoothing
- Today: continue more on naive Bayes, and introduce other linear classifiers
  - These are supervised machine learning models
  - Still little math heavy, but it’s built upon last lecture
- Helpful textbook chapter: [Jurafsky and Martin Chapter 4]
  - Some slides on gradient descent and loss function from this lectures are from this book chapter too

Additional comments from last lecture

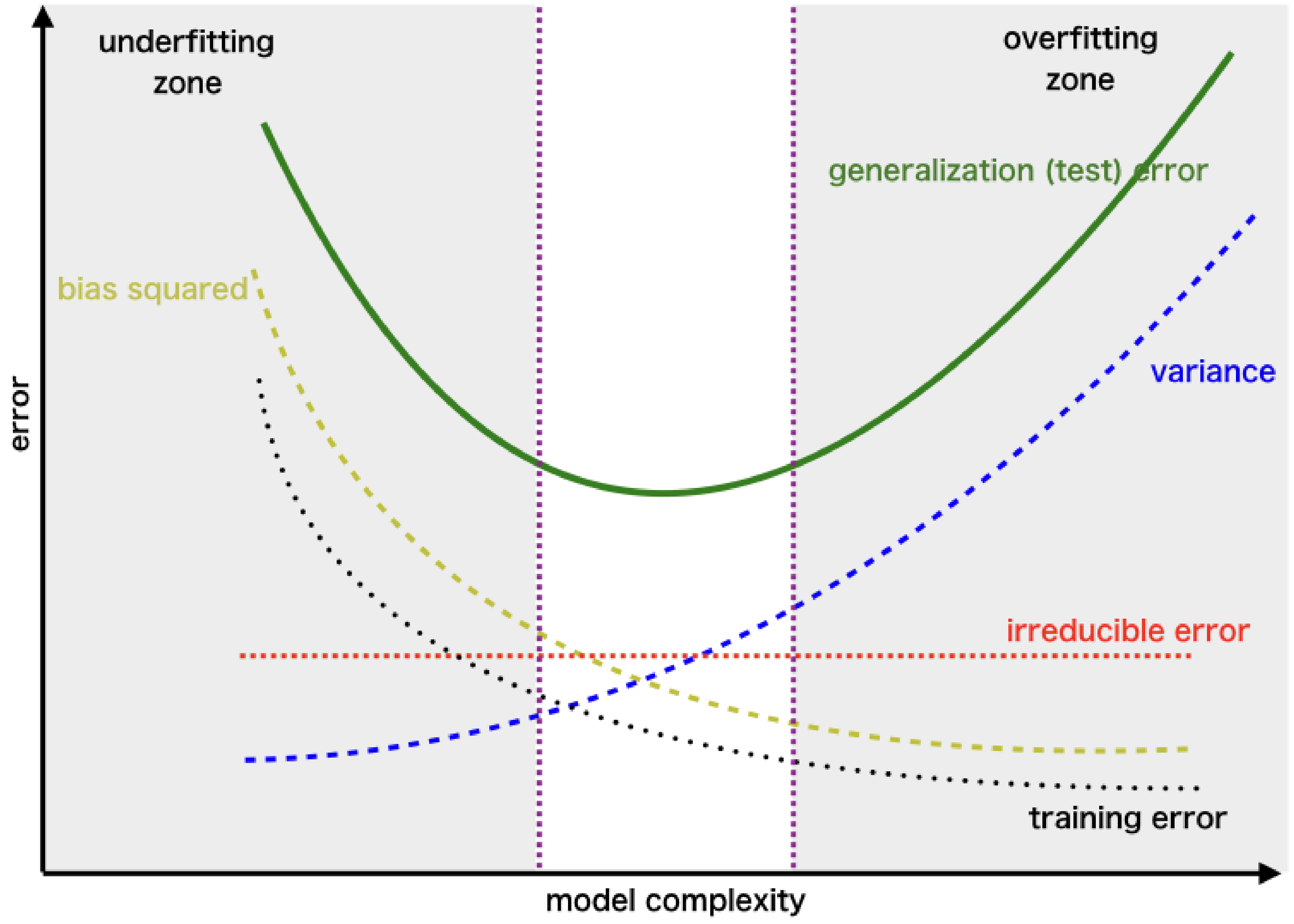

Low Variance
High Variance
Low Bias
High Bias

Bias-variance tradeoff

A little ML review, helpful for understanding MLE vs Smoothing

$$Expected\ Error = \ Bias^2 + Variance + Irreducible\ Error$$

- **Bias**: how far your model/estimator's average prediction is from the true value
  - High → model is too simple and not learning the pattern of the data → **underfitting**
  - Low → the model can capture the data well
- **Variance**: how much your model/estimator's prediction will fluctuate for different training sets
  - High → model is too sensitive to training data → **overfitting**
  - Low → predictions are stable; model can generalize well on different datasets
- You usually can't minimize both bias and variance at the same time


underfitting zone
overfitting zone
generalization (test) error
bias squared
variance
error
irreducible error
training error
model complexity

Examples of text classification tasks

- Sentiment analysis
- Spam detection
- Toxicity detection
- Etc.

Classification: to put a **label** on a text, and the labels are from some **pre-defined categories**

- {positive, negative}
- Movie genre, e.g. {comedy, horror, romantic, scifi}
- Emotion/sentiment, e.g. {happy, sad, worried, sarcastic, despise}
- Language id, e.g. {English, French, Mandarin, Spanish}

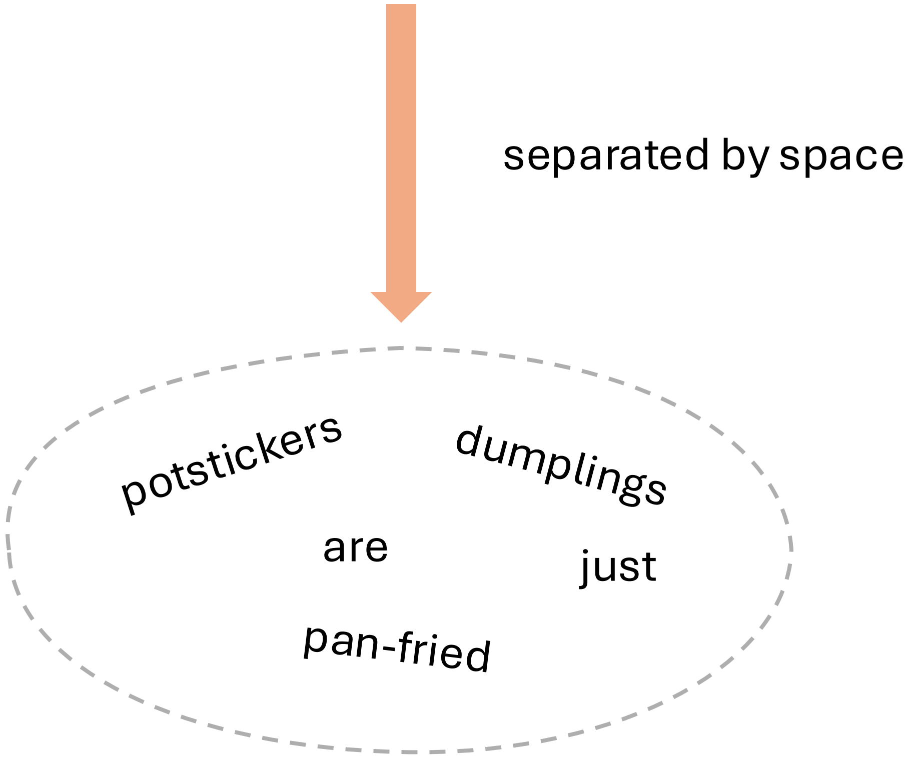
Pre-processing text for classification

- From a string of text, we can
  - Use them as is, sort of
    - Bag of words
      - Then use word count or frequency.
  - Turn it into a more abstract representation
    - Feature vectors
      - Designed features
      - Learned features (unsupervised learning). You need to define the number of dimensions.

Bag of words

Bag of words

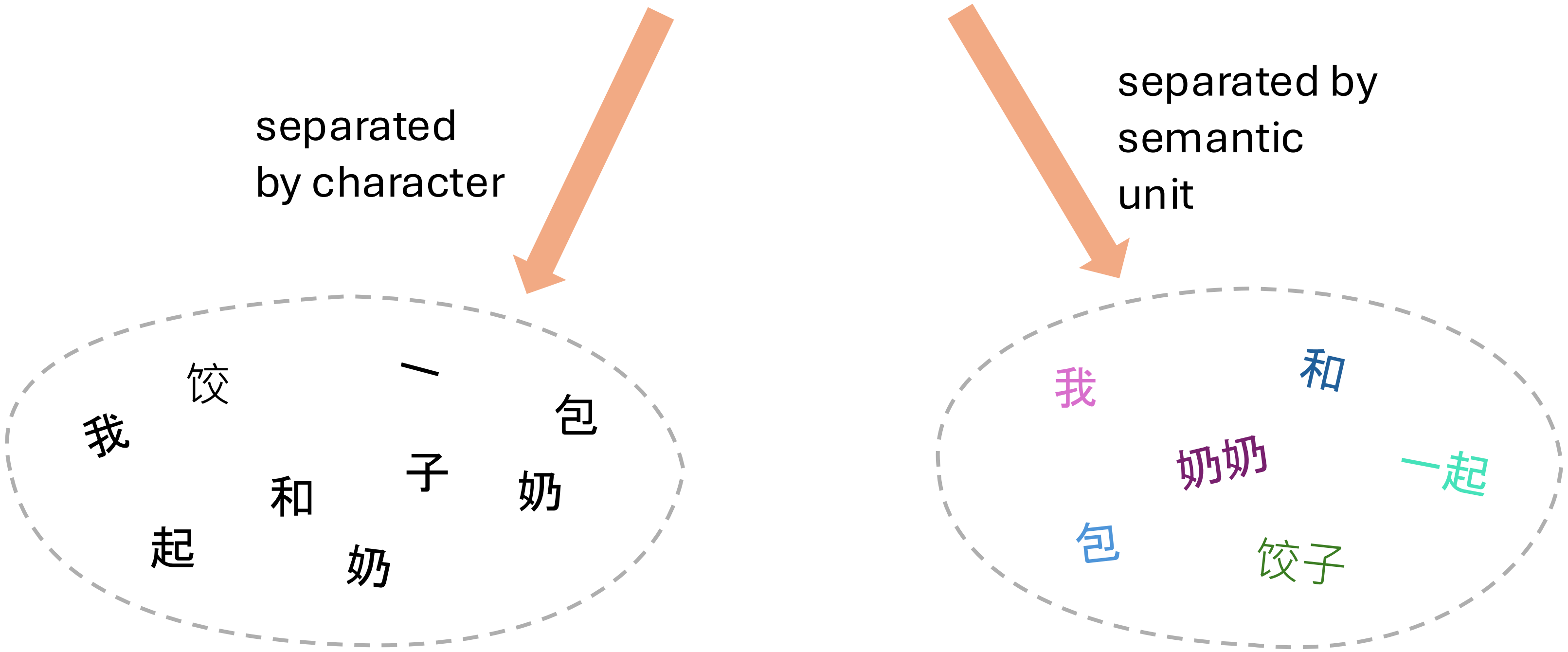
English: Potstickers ␣ are ␣ just ␣ pan-fried ␣ dumplings



Bag of words (not always separated by space)

Me and grandma (are) making dumplings together

Chinese: 我和奶奶一起包饺子



Unlike English, Chinese isn’t space-delimited.
Each character is an independent graphological unit, the more meaningful semantic unit could be made of one or more characters.

What about bag of words as… vector?

Raw text:

“a doctor has a lot of patients waiting.”

Bag-of-words vector

lot	**1**
a	**2**
hospital	**0**
dumpling	**0**
is	**0**
apple	**0**
doctor	**1**
of	**1**
…	**…**

This feature vector will be very **sparse** if the vocabulary of the corpus is huge. Most of the entry there will be zero.

Dealing with sparsity

- Dimensionality reduction: project sparse vector to a lower-dimensional dense space
  - Latent semantic analysis (LSA): document-term matrix then SVD
  - Principal component analysis (PCA)
- Feature selection:
  - Only keeping the most informative words
  - E.g. use a threshold for frequency, only keeping words that occur > 100 times
- Weighting schemes:
  - TF-IDF: BoW but instead of just word counts, it weights the importance of a word for a document.
- Using word embeddings and other embeddings
  - Much smaller in terms of dimension size
  - Next lecture!

Linear classifier

- **(Some)naïve Bayes, logistic regression**, support vector machine, **single-layer perceptron** are all linear classifiers

- There are two kinds of linear classifier:
  - **Generative**: try to model how the data was generated for each class (think likelihood from last lecture about naïve Bayes), learn the probability distribution for each class, then classify (think posterior)
    - Naive Bayes classifier
  - **Discriminative**: doesn’t care how the data was generated, instead, directly learn the boundaries that separate classes
    - “linear” classifier measn the decision boundary is linear
    - Logistic regression
    - Support vector machines (SVM)

Naïve Bayes Classifier

- It's a generative classifier
- Linear if Bernoulli or multinomial
- In the log space:

$$\hat{y} = \underset{k \in \{1,\dots,K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

$$\begin{aligned}\log p(C_k \mid \mathbf{x}) &\propto \log\left(p(C_k) \prod_{i=1}^{n} {p_{ki}}^{x_i}\right) \\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki} \\ &= b + \mathbf{w}_k^\top \mathbf{x}\end{aligned}$$

- Optional: you can read more about the math behind it here https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote05.html

Our goal for probabilistic classification

High-level:

Predicted probability for each class in $Y = \{y_1, y_2, y_3, y_4, y_5\}$

Input text

f

x

$P(y_1|x) = ?$

$P(y_2|x) = ?$

$P(y_3|x) = ?$

$P(y_4|x) = ?$

$P(y_5|x) = ?$

Example

Task: identifying language

x = "I am learning logistic regression"

f →

$y_1 = English$
$y_2 = Spanish$
…

$\boldsymbol{P(English|x)}$**= 0.99**

$P(Spanish|x)$ = 0.005

$P(Russian|x)$ = 0.003

$P(Mandarin|x)$ = 0.001

$P(Hindi|x)$ = 0.001

Discriminative linear classifier

- Generally, it can be expressed as:

$$f(x) = \sum_{i=1}^{d} w_i x_i = w^T x + b$$

or expressed as a dot product of vector $\boldsymbol{w}$ and $\boldsymbol{x}$:

$$f(x) = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

  - Where w is the vector of weights, and b is the bias term, and $\boldsymbol{x}$ is the feature vector
  - Each w_i is corresponding to an x_i, and w_i is a real number

- This is high-level how we learn from the training data

$$f(x) = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

- Each w_i learns how important the feature x_i
  - If $w_i > 0$, larger x_i will make $f(x)$ score go higher
  - If $w_i < 0$, larger x_i will make $f(x)$ score go lower
  - If $w_i = 0$, x_i is irrelevant to the decision

- However, since w_i is a real number, $-\infty < f(x) < \infty$
  - We need a special function to map/squash $f(x)$ between 0 and 1to get a valid probability value
    → **Sigmoid** and **softmax** functions!

Logistic regression

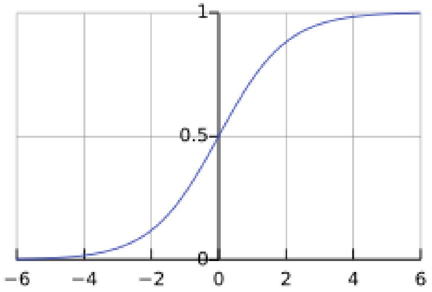
Logistic regression

- It’s a type of **probabilistic classifier**
  - Each label will get a probability for the text we are classifying on
- It’s discriminative
- It’s a linear classifier: decision boundary is linear
- **Supervised learning**:
  - Training: learn the pattern
  - Test: how good was the learning?
  - All data are labeled

With sigmoid function

Predicted probability for each class in $Y = \{y_1, y_2\}$

$f =$



Input text

x

$P(y_1|x)$

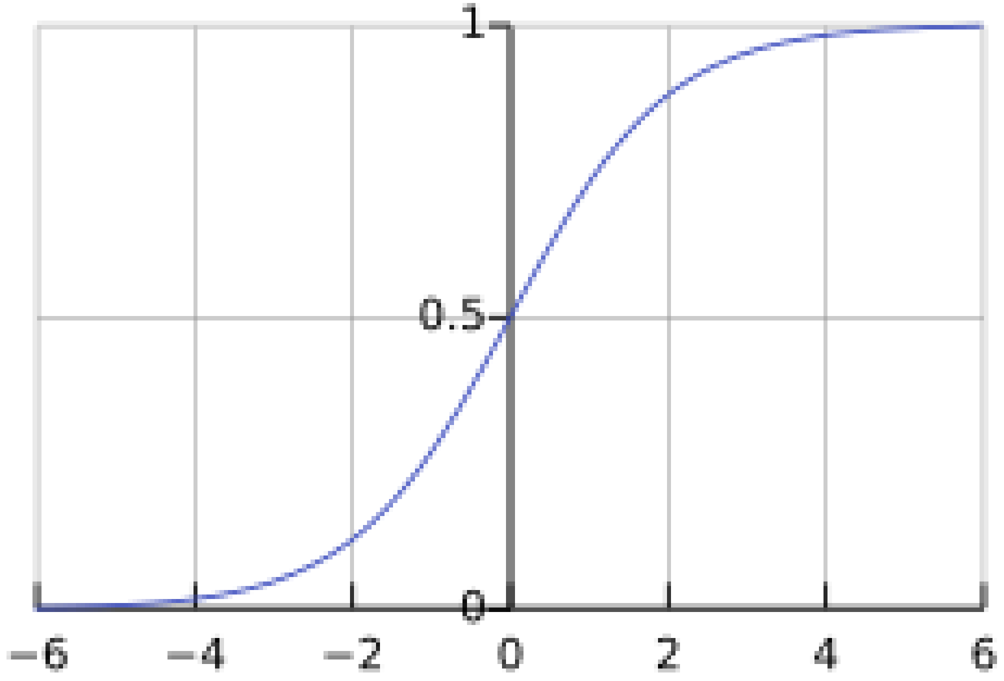
$P(y_2|x)$

Smooth and differentiable
Good for gradient-based optimization and gradient descent for neural network (future lecture)

Sigmoid function

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

- For when number of labels is 2, i.e. binary classification
  - E.g. Y = {negative, positive}
- This function helps us make the decision between two classes and output the probability in the range (0,1)



e.g.

$$0 < p(y = \text{“}positive\text{”}\,|x) < 1$$

$$0 < p(y = \text{“}negative\text{”}\,|x) < 1$$

$$p(y = \text{“}positive\text{”}\,|x) + p(y = \text{“}negative\text{”}\,|x) = 1$$

Let's go from try to squash f(x) into range (0,1) with sigmoid function

$$f(x) = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

Apply Sigmoid σ to $f(x)$, and we get:

$$\mathrm{p}(\mathrm{y} = \text{“}positive\text{”} | \, x) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

Note that sigmoid function $\sigma = \frac{1}{1+ e^{-x}}$.
We use $\exp(x)$ to denote e^x

Then

$$\mathrm{p}(\mathrm{y} = \text{“}positive\text{”} | \, x) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$
$$= \frac{1}{1 + \exp(\, -(\boldsymbol{w} \cdot \boldsymbol{x} + b \,) \,)}$$

- Now we know $\mathrm{p}(\mathrm{y} = \text{“}positive\text{”} | \, x)$, What about $\mathrm{p}(\mathrm{y} = \text{“}negative\text{”} | \, x)$?

$$\mathrm{p}(\mathrm{y} = \text{“}positive\text{”} | \, x)$$

(Optional whiteboard activity)
We can prove this!

Since sigmoid has the property of

$$\sigma(-x) = 1 - \sigma(x)$$

It’s perfect for binary classification, because now

$$\mathrm{p}(\mathrm{y} = \text{“}negative\text{”} | \, x) = 1 - \mathrm{p}(\mathrm{y} = \text{“}positive\text{”} | \, x)$$

Scaling up!

- So far we are only talking about using one input
- What if we have 1000 input that we need to classify? E.g. a batch of reviews from different people, each review has 100 words?
- This is where we use matrix instead of vector

Before, with vector:

$$y = \boldsymbol{\sigma}(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

With matrix:

$$\hat{y} = \boldsymbol{\sigma}(\boldsymbol{X} \cdot \boldsymbol{w} + \boldsymbol{b})$$

With multiple input X: $\hat{y} = \boldsymbol{\sigma}(\boldsymbol{X} \cdot \boldsymbol{w} + \boldsymbol{b})$

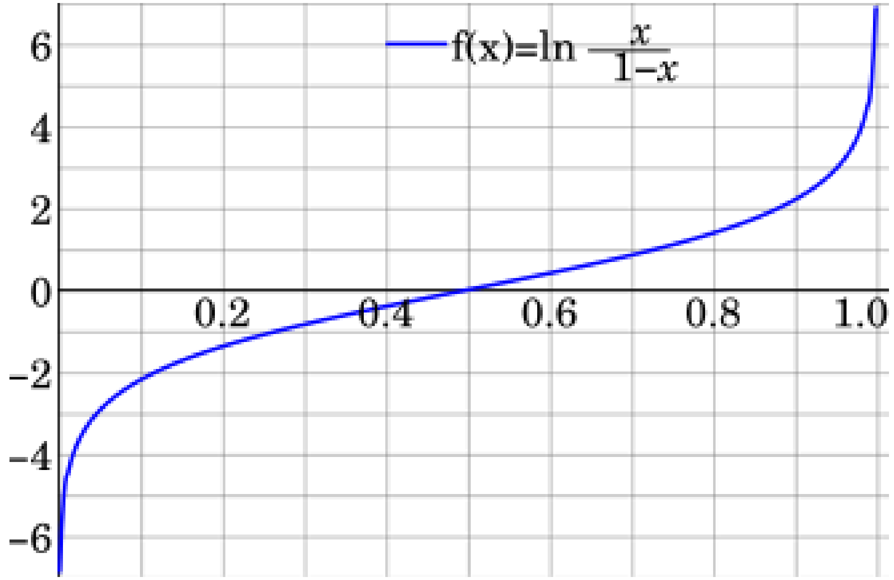
$$z = Xw + b = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b = \begin{bmatrix} w_1 x_{11} + w_2 x_{12} + b \\ w_1 x_{21} + w_2 x_{22} + b \\ w_1 x_{31} + w_2 x_{32} + b \end{bmatrix}$$

Apply sigmoid

$$\hat{y} = \sigma(z) = \begin{bmatrix} \sigma(w_1 x_{11} + w_2 x_{12} + b) \\ \sigma(w_1 x_{21} + w_2 x_{22} + b) \\ \sigma(w_1 x_{31} + w_2 x_{32} + b) \end{bmatrix}$$

- Each row of X is an input, of 2 features, and there are 3 input
- Each row of w modify a feature
- Notice that b is a scalar

Useful note for the future



- We just talked about

$$f(x) = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

- Let the score $z = f(x)$, z is called the **logit**, because it is the input of sigmoid, and the inverse of sigmoid function is called the logit function

$$logit(p) = \sigma^{-1}(p) = \ln\left(\frac{p}{1-p}\right) \text{ for } p \in (0,1)$$

Softmax function

- For **multiclass classification,** we use softmax instead of sigmoid.
  - In the case of logistic regression, it's call **multinomial logistic regression**

- Same as sigmoid, it can map values to the range (0,1)

- Given a vector $z = [z_1, z_2, z_3, \ldots, z_K]\ where\ K > 1$

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)} \quad 1 \leq i \leq K$$

$$\text{softmax}(\mathbf{z}) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)}, \ldots, \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right]$$

But exactly how do we learn with weights and bias terms?

Loss function and optimization

- Supervised classification:
  - We know the correct label y (either 0 or 1) for each *x*.
  - But what the system produces is an estimate, $\hat{y}$
- We want to set *w* and *b* to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.
  - We need a distance estimator: a **loss function** or a **cost function**
  - We need an **optimization** algorithm to update *w* and *b* to minimize the loss.

Loss function

- The goal of loss function is to make the predicted results more similar to the gold label
  - By minimizing the distance between the predicted output and ground truth
  - This is how we measure how well the model is learning

- A common loss function is **cross-entropy loss function**
- More on this later

Loss function: the distance between $\hat{y}$ and y

We want to know how far is the classifier output:

$\hat{y}$ = σ(w·x+b)

from the true output:

y [= either 0 or 1]

We'll call this difference:

$L(\hat{y}, y)$ = how much $\hat{y}$ differs from the true y

Deriving cross-entropy loss for a single observation x

- **Goal**: maximize probability of the correct label $p(y|x)$
- Since there are only 2 discrete outcomes (0 or 1) we can express the probability $p(y|x)$ from our classifier (the thing we want to maximize) as

$$p(y|x) = \hat{y}^{y} (1-\hat{y})^{1-y}$$

noting:

if y=1, this simplifies to $\hat{y}$

if y=0, this simplifies to $1-\hat{y}$

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Maximize:

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

Now take the log of both sides (mathematically handy)

Maximize:

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

Whatever values maximize log p(y|x) will also maximize p(y|x)

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

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

Now flip sign to turn this into a loss: something to minimize

Cross-entropy loss (because is formula for cross-entropy(y, $\hat{y}$))

Minimize:
$$L_{\mathrm{CE}}(\hat{y},y)=-\log p(y|x) \;=\; -[y\log\hat{y}+(1-y)\log(1-\hat{y})]$$

Or, plugging in definition of $\hat{y}$:

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

Our goal: minimize the loss

Let's make explicit that the loss function is parameterized by weights θ=(w,b)

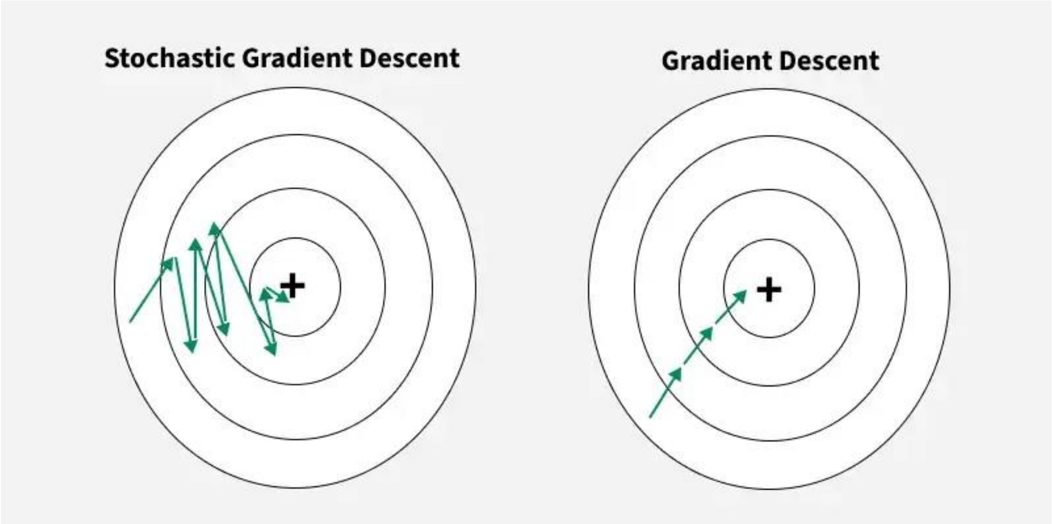
- And we'll represent $\hat{y}$ as f (x; θ) to make the dependence on θ more obvious

We want the weights that minimize the loss, averaged over all examples:

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

Optimization

- Now that we have a loss function, we need to minimize it
  - We minimize by updating weights
- Most common optimization technique is **gradient descent**, where we iteratively update our weights



Our goal: minimize the loss

For logistic regression, loss function is **convex**

- A convex function has just one minimum
- Gradient descent starting from any point is guaranteed to find the minimum
  - (Loss for neural networks is non-convex)

Gradients

- The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.
- **Gradient Descent**: Find the gradient of the loss function at the current point and move in the **opposite** direction.

How much do we move in that direction ?

- The value of the gradient (slope in our example) $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η
- Higher learning rate means move *w* faster

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

Real gradients

- Are much longer; lots and lots of weights
- For each dimension w_i the gradient component i tells us the slope with respect to that variable.
  - "How much would a small change in w_i influence the total loss function L?"
  - We express the slope as a partial derivative ∂ of the loss ∂w_i
- The gradient is then defined as a vector of these partials.

The gradient

We'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious:

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

The final equation for updating θ based on the gradient is thus

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

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 $x^{(1)}, x^{(2)}, ..., x^{(m)}$
    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, ..., y^{(m)}$

$\theta \leftarrow 0$
repeat til done   # see caption
  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

Regularization

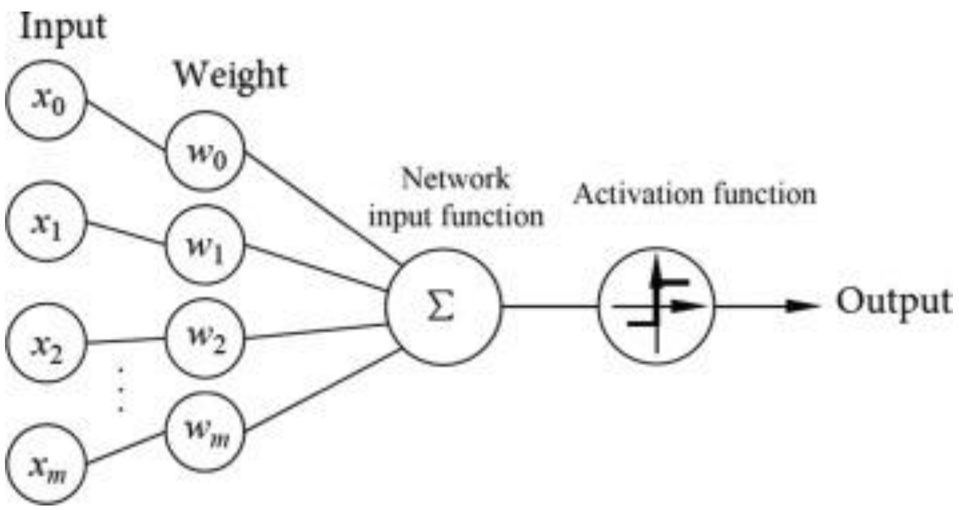
- Add to loss function to prevent overfitting
  - A penalty term for better generalization
  - $Loss = Error + \lambda \cdot Penalty$
- L1 regularization (lasso).
  - $Penalty = \sum_i |w_i|$
- L2 regularization (ridge)
  - $Penalty = \sum_i w_i^2$

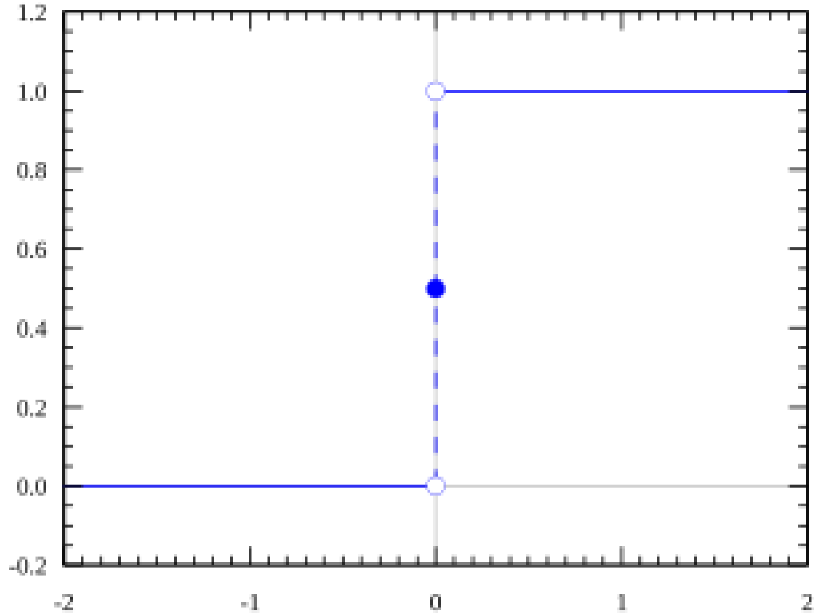
Perceptron

We will talk briefly about this, but more in the future lecture

Single-layer perceptron

- A single-layer perceptron is also a linear classifier!
- The activation function is a step-function, which is non-linear, but the output decision boundary is linear
- In the future, we will talk about multi-layer perceptron





$$H(x) := \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

First figure from https://www.sciencedirect.com/topics/engineering/perceptron

Additional resources on ML

https://introml.mit.edu/notes/

https://www.cs.cornell.edu/courses/cs4780/2024sp/

https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/

- These are helpful if you want to review ML