Training Models 4: Logistic Regression

Glenn Bruns

California State University
MONTEREY BAY

Learning outcomes

After this lecture you should be able to:

1. Define logistic regression as an optimization problem
2. Develop multi-class classifiers from binary classifiers
3. Define how to generalize logistic regression to more than 2 classes

Bonus:

- Compare training with the log loss function and with maximum likelihood

Recall: ML as an optimization problem

steps	example: linear regression
define a **model**, with parameters, that will be used to make predictions	$\hat{y} = \theta^T \cdot \mathbf{x}$
define a **cost function** to explain what it means for the model to fit the data well	$MSE(\mathbf{X}, \theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$
if possible, find the **partial derivatives of the cost function**	$\frac{\partial}{\partial \theta_j} MSE(\mathbf{X}, \theta) = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^T \cdot x^{(i)} - y^{(i)}\right) x_j^{(i)}$
fit the model to your training data by finding the parameters that **minimize the cost function, using gradient descent**	

Model for logistic regression

Logistic regression is used for classification.

To get class probabilities, first apply a linear model, then apply the logistic function:

$$\hat{p} = \sigma(\theta^T \cdot \boldsymbol{x})$$

Logistic function:

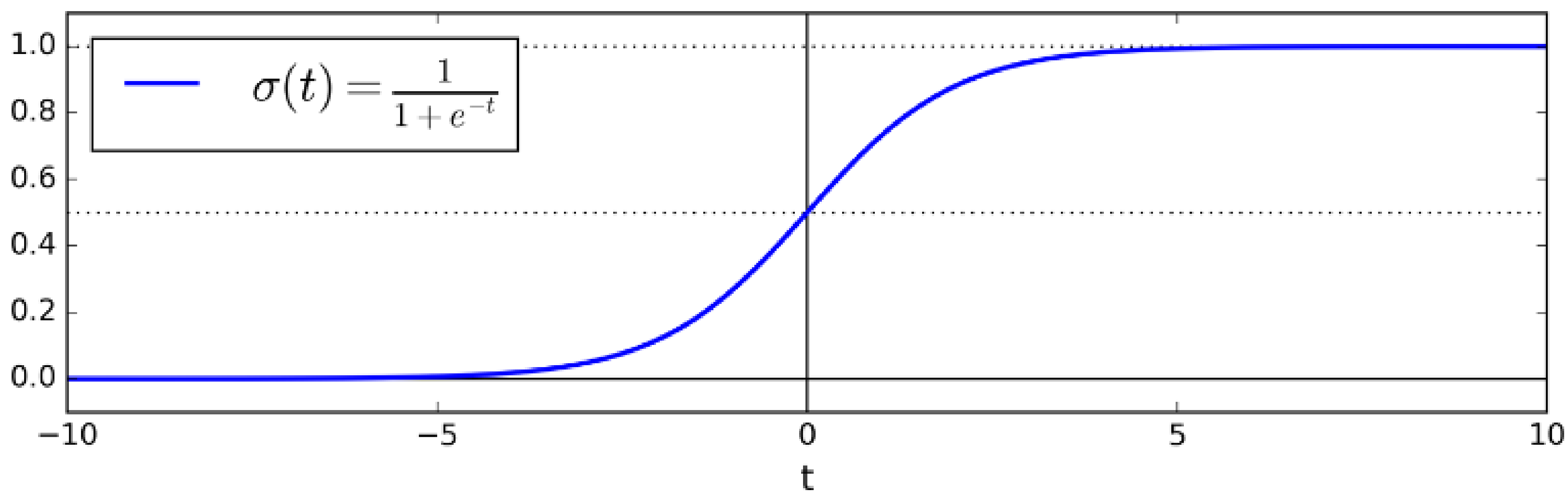


figure: Géron

Cost function for logistic regression?

A prediction $\hat{p}$ is between 0 and 1.

A target value y is exactly 0 or 1.

What should the cost function do?

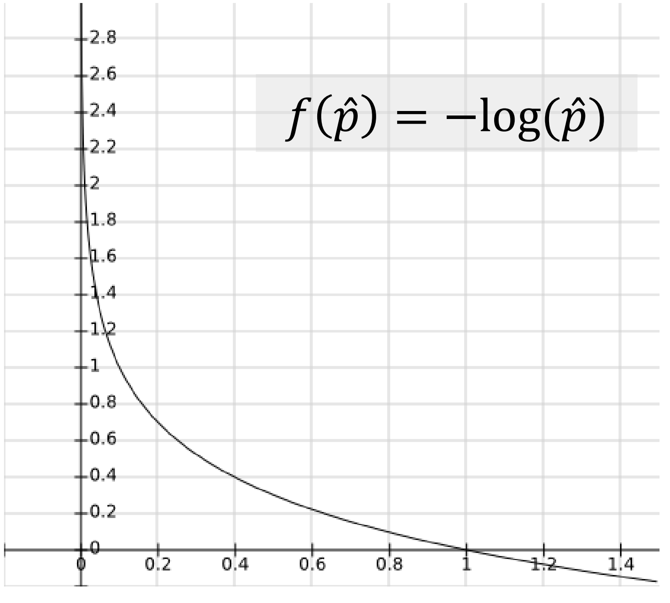
For example, what if the prediction is 0.4 and the correct label is 1?

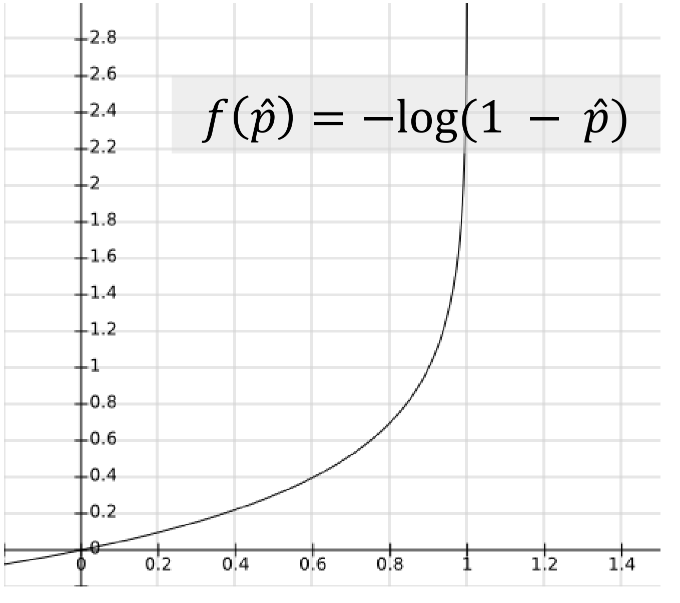
Cost function for a single prediction

$$c(\theta) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$

if y = 1, cost is high when $\hat{p}$ is 0

if y = 0, cost is high when $\hat{p}$ is 1





Cost function for entire training set

Cost for a single training example:

$$c(\theta) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$

Cost for training set is the average cost per training example:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - \hat{p}^{(i)}\right)\right]$$

m : number of training examples
$\hat{p}^{(i)}$: predicted probability of ith training example
$y^{(i)}$: label (0 or 1) or ith training example

"log loss"

Logistic regression decision boundary

Classify irises using logistic regression:

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris["data"][:, 3:]   # petal width
y = (iris["target"] == 2).astype(np.int) # 1 if Iris-Virginica, else 0

# fit the logistic regression model
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression()
log_reg.fit(X, y)
```

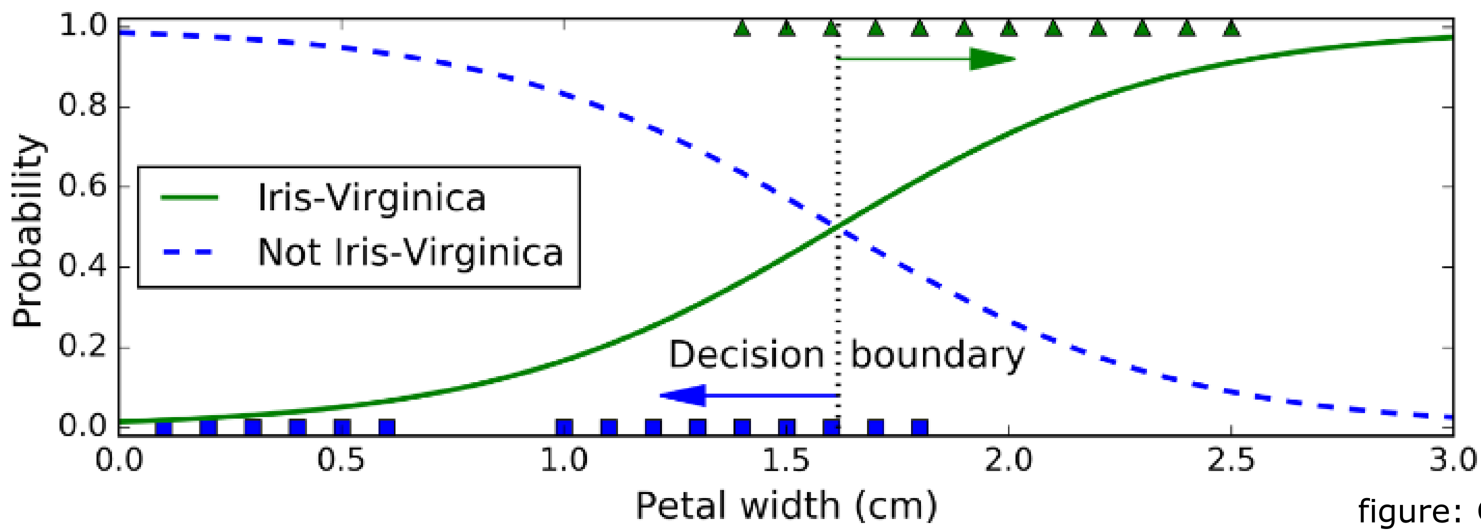
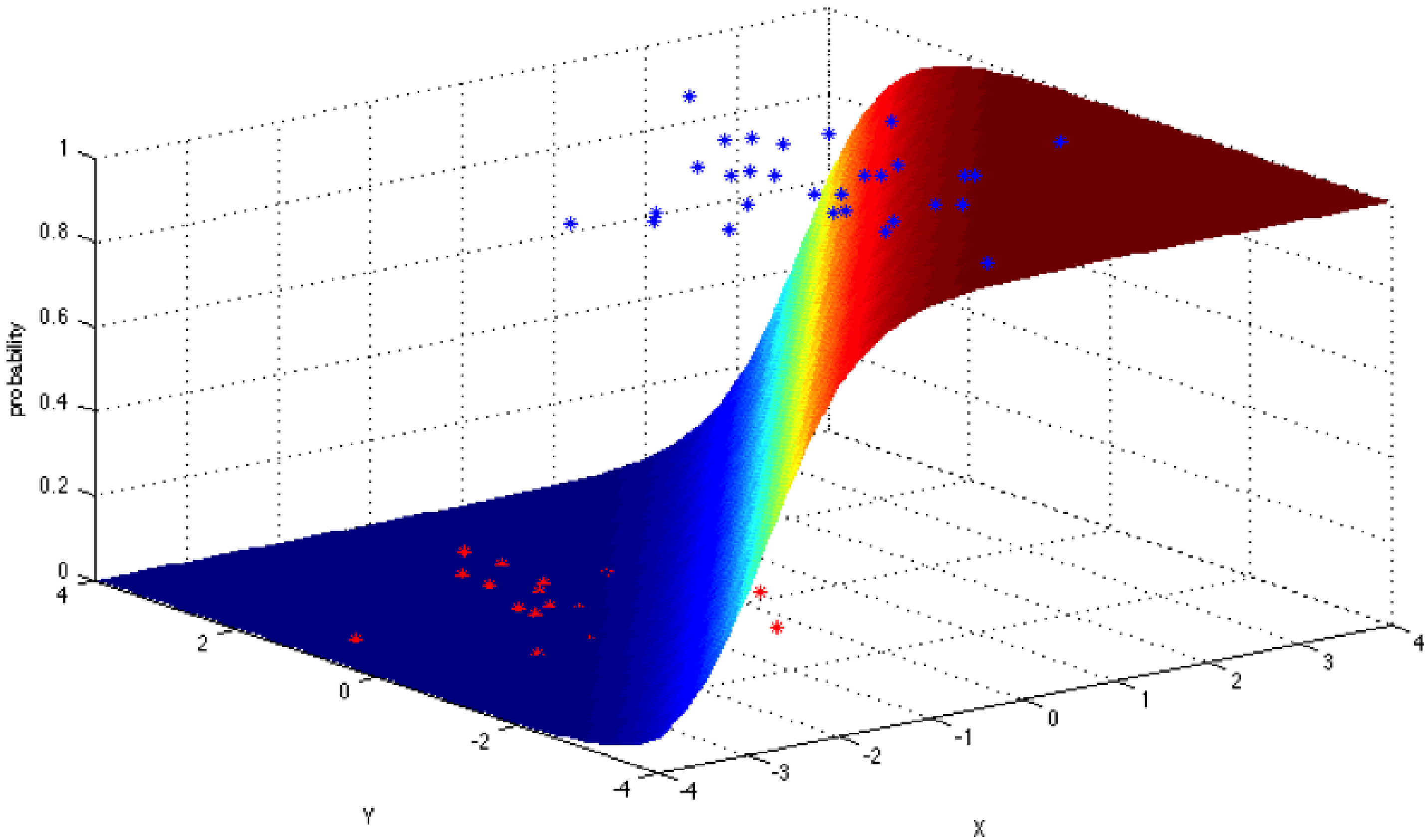


figure: Géron

Logistic curve with two predictors



Logistic regression with >2 classes?

Here are two ways to build a k-way classifier from a bunch of binary classifiers:

- One-vs-all:
  - Train one binary classifier for every class
  - Run every classifier; select class with highest decision score
- One-vs-one:
  - Train a binary classifier for every pair of classes
  - Run every classifier; select class that wins the most "duels"

	C1	C2	C3
score on x	0.5	0.7	0.2

which class do we predict?

	C12	C13	C23
score on x	0.2	0.6	0.7

which class do we predict?

Softmax regression

We can modify logistic regression to directly support k-way classification.

Idea: Each class k has its own vector θ_k of coefficients used to compute a score for class k:

$$s_k(\mathbf{x}) = {\theta_k}^T \cdot \mathbf{x}$$

Suppose we have 3 classes.

From a training example $\mathbf{x}$, compute a score for each class:

$s_1(\mathbf{x})$	$s_2(\mathbf{x})$	$s_3(\mathbf{x})$
1.7	2.4	0.5

How to convert these scores into probabilities for each class?

Softmax function

We can use the softmax function to get a probability value for every class k:

$$\hat{p}_k = \sigma(s(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(s_j(\mathbf{x}))}$$

Example:

$s_1(\mathbf{x})$	$s_2(\mathbf{x})$	$s_3(\mathbf{x})$
1.7	2.4	0.5

Step 1:

$\exp(1.7) = e^{1.7} = 5.5$

$\exp(2.4) = e^{2.4} = 11$

$\exp(0.5) = e^{1.6} = 1.6$

Probability of class 1: $\frac{5.5}{5.5+11+1.6} = 0.30$

Probability of class 2: $\frac{11}{5.5+11+1.6} = 0.61$

Probability of class 3: $\frac{1.6}{5.5+11+1.6} = 0.09$

What is the predicted class?

scores:

$s_1(\mathbf{x})$	$s_2(\mathbf{x})$	$s_3(\mathbf{x})$
1.7	2.4	0.5

probabilities:

$\boldsymbol{\hat{p}_1}$	$\boldsymbol{\hat{p}_2}$	$\boldsymbol{\hat{p}_3}$
1.7	2.4	0.5

1. use class with highest probability $\underset{k}{\operatorname{argmax}}\, \sigma\big(s(x)\big)_k$
2. use class with highest score $\underset{k}{\operatorname{argmax}}\, s_k(x)$
3. use class with highest value of ${\theta_k}^T \cdot x$ $\underset{k}{\operatorname{argmax}}({\theta_k}^T \cdot x)$

Cost function for softmax regression?

How to compute how well the model is doing on a training set?

actual class	predicted prob. of class 1	predicted prob. of class 2	predicted prob. of class 3
1	$\boxed{0.1}$	0.5	0.4
3	0.1	0.1	$\boxed{0.8}$
2	0.2	$\boxed{0.6}$	0.2

Idea:

- for each example, cost is the log of the prediction $\hat{p}_k$ for the actual class k

  For example 1, $\log(0.1) = -1.0$ For example 2, $\log(0.8) = -0.097$

- for total cost, use negative average

$$-\frac{1}{3}(-1.0 - 0.097 - 0.22) = 0.44$$

Cost function in softmax regression

Cross-entropy cost function:

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_{\mathrm{k}}^{(\mathrm{i})} \log(p_k^{(i)})$$

$y_k^{(i)}$ is 1 if $y^{(i)}$ is k, else 0

Example:

actual class	predicted prob. of class 1	predicted prob. of class 2	predicted prob. of class 3
1	0.1	0.5	0.4
3	0.1	0.1	0.8
2	0.2	0.6	0.2

$J(\Theta) = -\frac{1}{3} *$

(log(0.1)

+ log(0.8)

+ log(0.6))

= -1/3 (-1 - 0.10 – 0.22)

= 0.44

Summary

1. We can turn values from a linear model into probabilities using the logistic function.
2. In logistic regression (binary classification), the cost function is “log loss”.
3. You can build a k-way classifier from binary classifiers using ‘one-vs-all’ or ‘one-vs-one’
4. Softmax regression is a “native” k-way extension to logistic regression

Bonus: Log loss versus max likelihood

Training a logistic regression model is often done using the maximum likelihood method.

- Define the probability of the seeing the training data given the parameters
- Use the parameter values to maximize the probability of seeing the training data

Here's the probability of seeing the training data:

$$\prod_{i:y^{(i)}=1} \hat{p}^{(i)} \prod_{i':y^{(i')}=0} (1 - \hat{p}^{(i')})$$

If you take the log, and divide by the number of training examples, you get log loss!

Log loss versus max likelihood - detail

$$\log\left(\prod_{i:y^{(i)}=1} \hat{p}^{(i)} \prod_{i':y^{(i')}=0} \left(1-\hat{p}^{(i')}\right)\right)$$

$$= \log\left(\prod_{i:y^{(i)}=1} \hat{p}^{(i)}\right) + \log\left(\prod_{i':y^{(i')}=0} \left(1-\hat{p}^{(i')}\right)\right)$$

$$= \sum_{i:y^{(i)}=1} \log\left(\hat{p}^{(i)}\right) + \sum_{i':y^{(i')}=0} \log\left(1-\hat{p}^{(i')}\right)$$

$$= \sum_{i}^{m} y^{(i)} \log\left(\hat{p}^{(i)}\right) + \sum_{i}^{m} \left(1-y^{(i)}\right) \log\left(1-\hat{p}^{(i')}\right)$$

$$= \sum_{i}^{m} y^{(i)} \log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right) \log\left(1-\hat{p}^{(i')}\right)$$

maximum likelihood formulation

log loss is this multiplied by -1/m