Logistic Regression Classifier

Liang Liang

Logistic Regression

is for Classification

NOT Regression !

A Linear Function

$$z = w^T x + b$$

x refers to a data point/feature vector

z is a scalar function of x

$$z = z(x) = w^T x + b$$

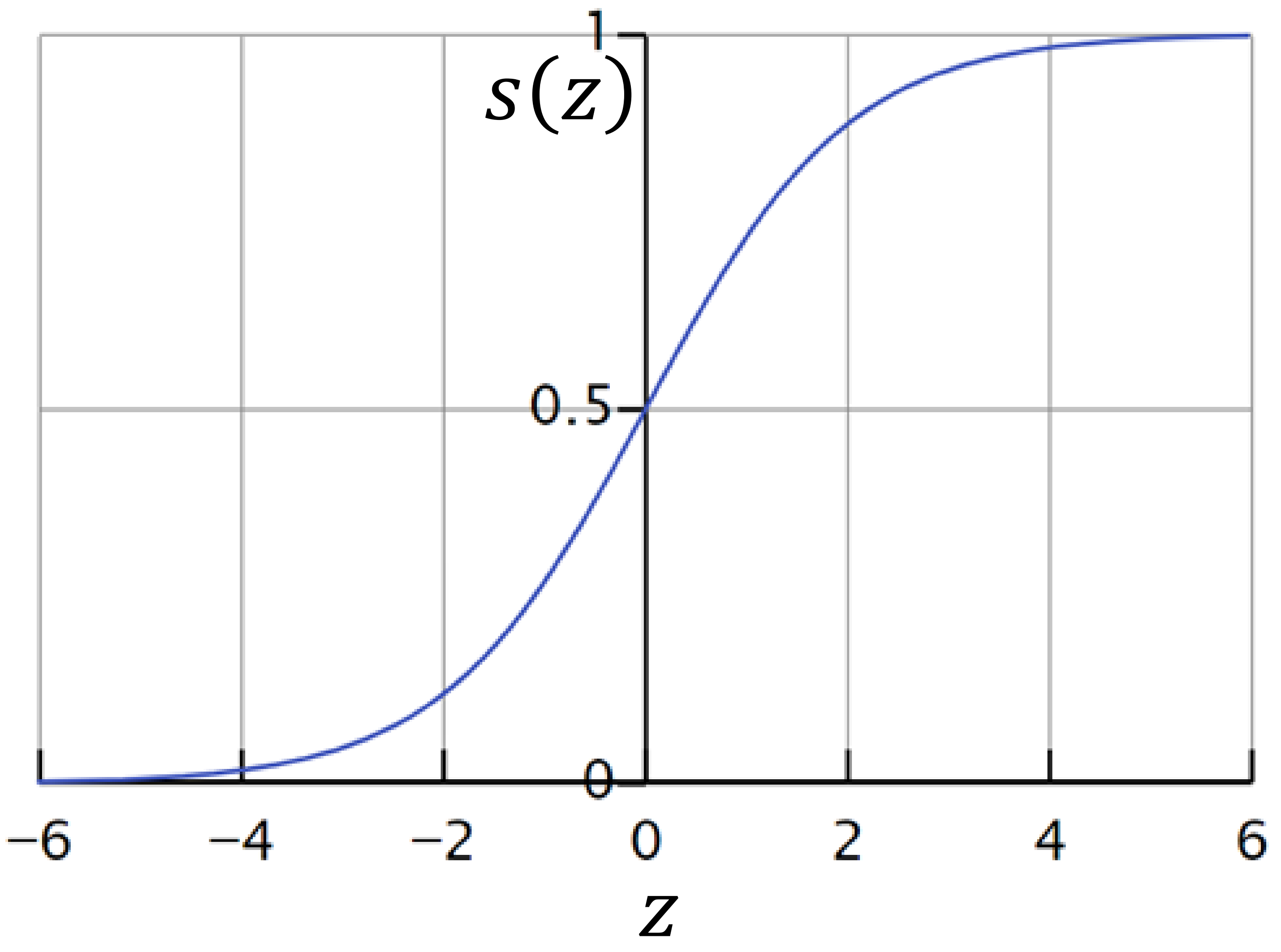
w is a parameter vector of the function

b is a scalar parameter of the function

For example: $z = [0.1, 2.0] \begin{bmatrix} x_{(1)} \\ x_{(2)} \end{bmatrix} + 1.2$

The sigmoid function

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



$$s(z = 0) = 0.5$$

$$s(z = \infty) = 1$$

$$s(z = -\infty) = 0$$

Binary Classifier: Linear + Sigmoid

a data point x

$$\hat{y} = s(w^T x + b)$$

Binary Classifier

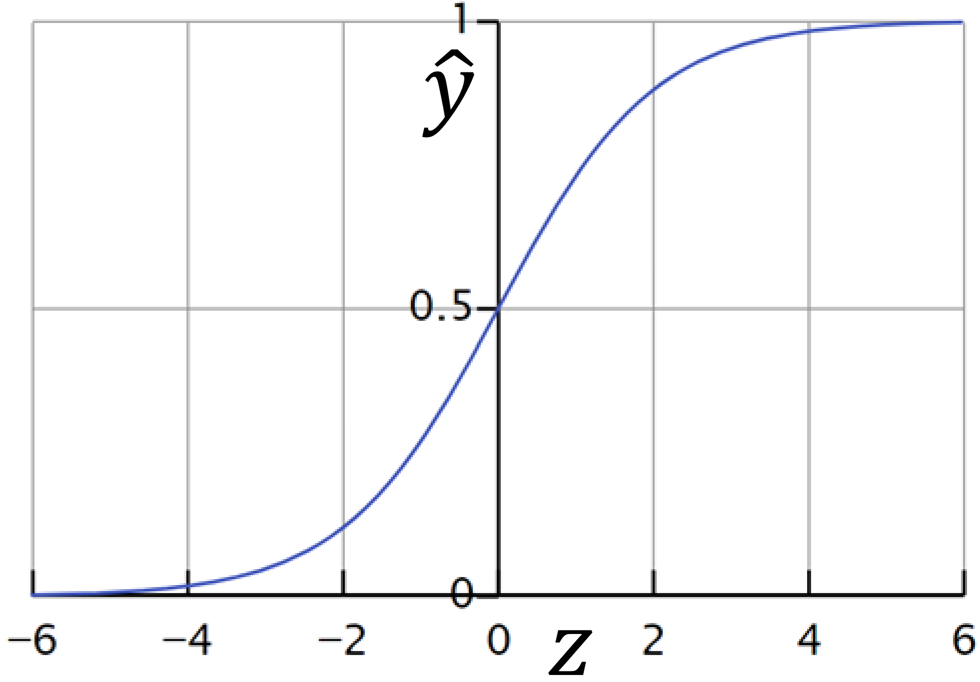
$\hat{y} > 0.5$, it is a cat

$\hat{y} < 0.5$, it is not a cat

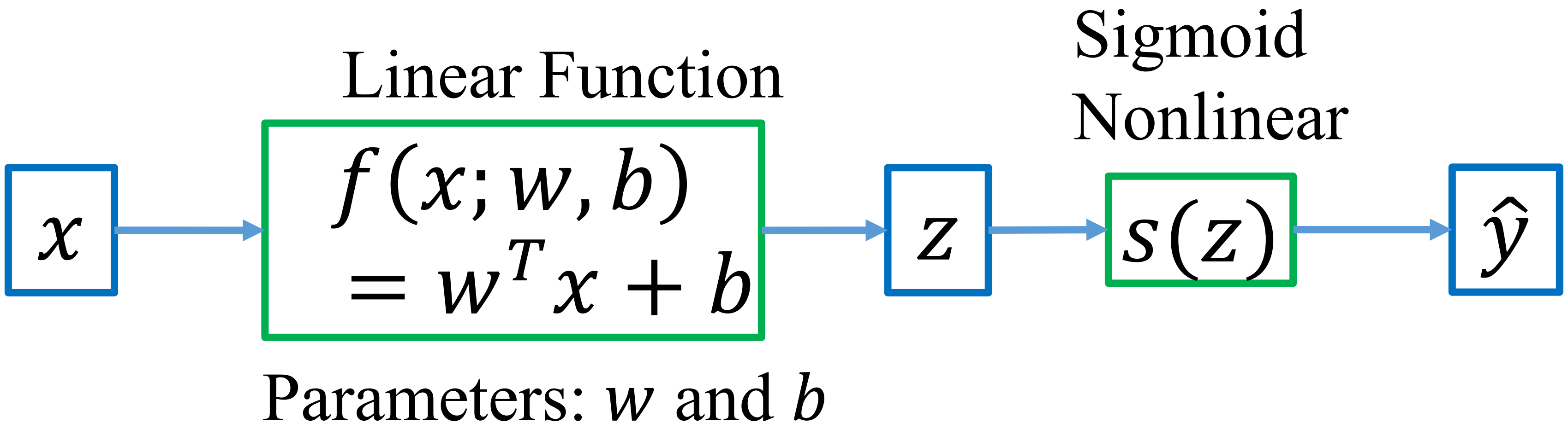
$\hat{y}$ predicted soft label

Linear function $z = w^T x + b$

Sigmoid function $\hat{y} = s(z) = \dfrac{1}{1 + e^{-z}}$



Binary Classifier: Linear + Sigmoid



- We have a set of <u>training</u> data points $\{x_1, x_2, x_3, \ldots, x_N\}$ and $x_n \in \mathcal{R}^M$
- We have <u>a set of 'ground-truth' labels</u> $\{y_1, y_2, y_3, \ldots, y_N\}$ and $y_n \in \{0, 1\}$
- The data points are from two classes: 0 vs 1 (e.g., not-cat vs cat)
- y_n is the true class label of x_n, and it is 0 or 1
- One data point belongs to only one class
- The predicted soft labels from the classifier are $\{\hat{y}_1, \hat{y}_2, \hat{y}_3, \ldots, \hat{y}_N\}$
- The classifier may make mistakes.

'ground-truth' labels

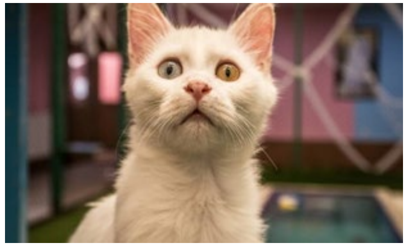
predicted soft labels

x_1

$y_1 = 1$

$\hat{y}_1 > 0.5$

x_2

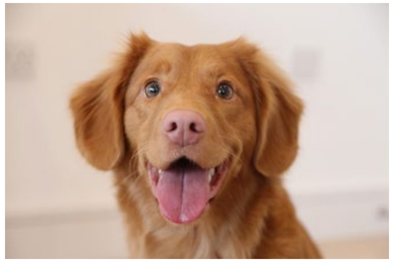


$y_2 = 1$

Classifier

$\hat{y}_2 < 0.5$

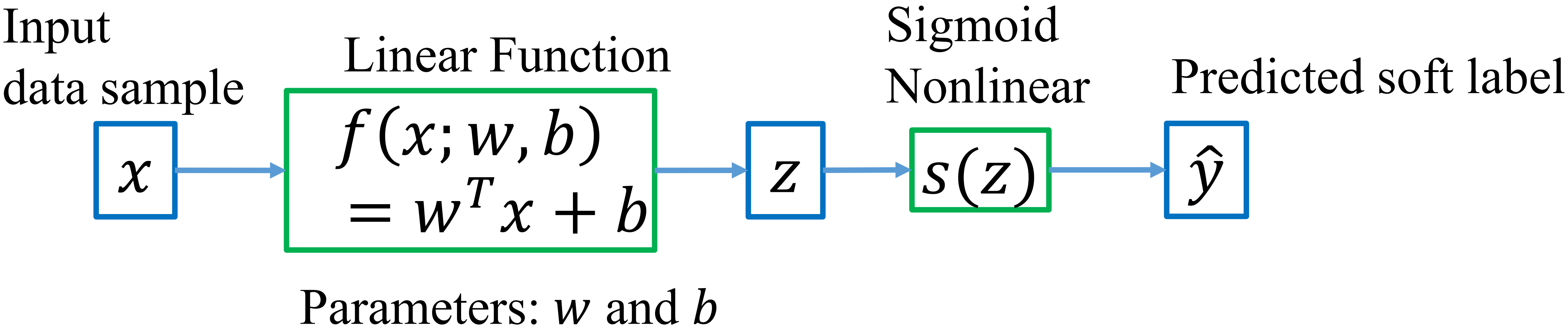
x_3



$y_3 = 0$

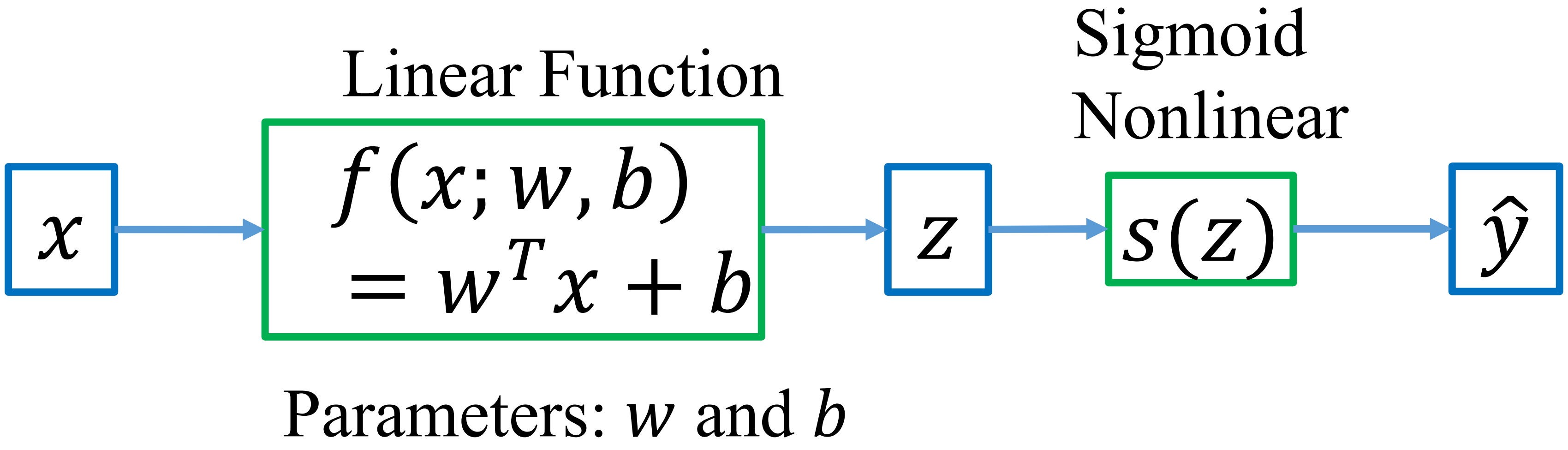
$\hat{y}_3 > 0.5$

Train the Classifier: find the optimal parameters



- Train the classifier on the training dataset:
  find the optimal parameters (w and b) such that the discrepancy between the 'ground-truth' labels $\{y_1, y_2, y_3, \ldots, y_N\}$ and the predicted labels $\{\hat{y}_1, \hat{y}_2, \hat{y}_3, \ldots, \hat{y}_N\}$ are minimized
- Need a loss function to measure the discrepancy/difference

Train the Classifier - the loss function



- Define a loss function to measure the discrepancy/difference

$$L(w, b) = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}_n - y_n)^2$$

which is called mean squared error (MSE) loss

loss functions for training the classifier

- mean squared error (MSE) loss

$$L = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

- mean absolute error (MAE) loss or L1 loss

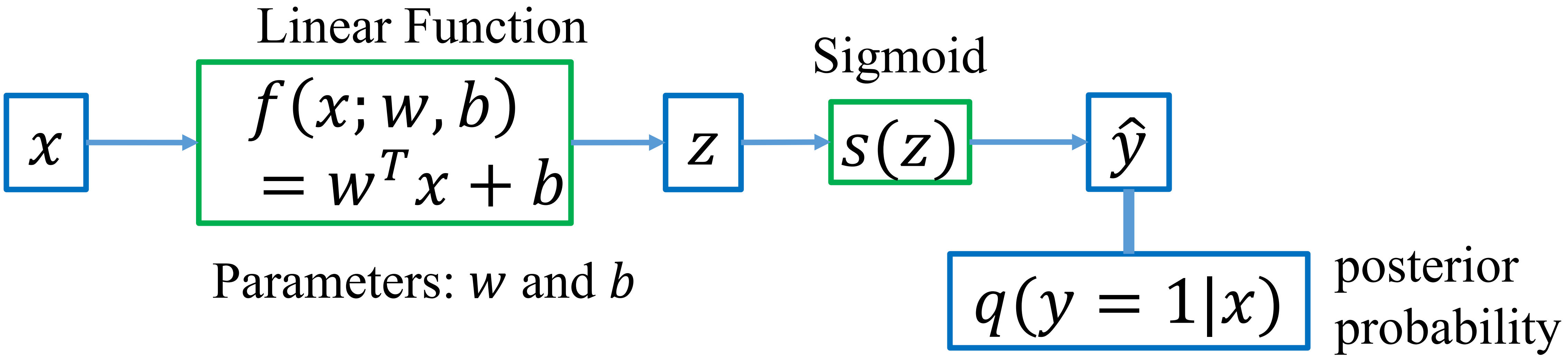
$$L = \frac{1}{N}\sum_{n=1}^{N}|\hat{y}_n - y_n|$$

- binary cross entropy (BCE) loss

$$L = -\frac{1}{N}\sum_{n=1}^{N}\left(y_n log(\hat{y}_n) + (1 - y_n)log(1 - \hat{y}_n)\right)$$

- negative log-likelihood (NLL) loss = BCE loss

Logistic Regression = Linear + Sigmoid + BCE loss



- We have a set of <u>training</u> data points $\{x_1, x_2, x_3, \dots, x_N\}$ and $x_n \in \mathcal{R}^M$
- We have <u>a set of 'ground-truth' labels</u> $\{y_1, y_2, y_3, \dots, y_N\}$ and $y_n \in \{0, 1\}$
- $q(y|x)$ is the 'probability' of x belonging to class-y
- $\hat{y} = q(y = 1|x)$, the output is the 'probability' of x belonging to class-1
- $1 - \hat{y}$ is the the 'probability' of x belonging to class-0 (two classes)

Logistic Regression = Linear + Sigmoid + BCE loss

- We have a set of training data points $\{x_1, x_2, x_3, \dots, x_N\}$ and $x_n \in \mathcal{R}^M$
- We have a set of 'ground-truth' labels $\{y_1, y_2, y_3, \dots, y_N\}$ and $y_n \in \{0, 1\}$
- The predicted labels from the classifier are $\{\hat{y}_1, \hat{y}_2, \hat{y}_3, \dots, \hat{y}_N\}$
- $\hat{y}_n = q(y = 1|x_n)$ is the (estimated) 'probability' of x_n belonging to class-1
- $1 - \hat{y}_n$ is the (estimated) 'probability' of x_n belonging to class-0
- Define the likelihood function:

$$L(w, b) = \prod_{n=1}^{N} (\hat{y}_n)^{y_n} (1 - \hat{y}_n)^{1-y_n}$$

- Logistic regression: maximize the likelihood function

$$L(w,b) = \prod_{n=1}^{N} (\hat{y}_n)^{y_n} (1-\hat{y}_n)^{1-y_n}$$

where $\hat{y}_n$ is the (estimated) 'probability' of x_n belonging to class-1

$1-\hat{y}_n$ is the (estimated) 'probability' of x_n belonging to class-0

Example: $\{x_1, x_2, x_3\}$ and $\{y_1 = 1, y_2 = 0, y_3 = 0\}$

Then: $L(w,b) = \hat{y}_1 (1-\hat{y}_2)(1-\hat{y}_3)$

the 'probability' of x_1 belonging to class-1

the 'probability' of x_3 belonging to class-0

the 'probability' of x_2 belonging to class-0

$L(w,b)$ is the 'probability' of observing the sequence $\{x_1, x_2, x_3\}$, assuming i.i.d.

Logistic Regression = Linear + Sigmoid + BCE loss

- $\hat{y}_n = q(y = 1|x_n)$ is the (estimated) 'probability' of x_n belonging to class-1
- $1 - \hat{y}_n$ is the (estimated) 'probability' of x_n belonging to class-0
- The likelihood function:

$$L(w, b) = \prod_{n=1}^{N} (\hat{y}_n)^{y_n} (1 - \hat{y}_n)^{1-y_n}$$

- negative log-likelihood (NLL) loss function:

$$\begin{aligned} loss &= -\frac{1}{N} log L(w, b) \\ &= -\frac{1}{N} \sum_{n=1}^{N} \left(y_n log(\hat{y}_n) + (1 - y_n) log(1 - \hat{y}_n) \right) \\ &= \frac{1}{N} \sum_{n=1}^{N} cross\_entropy(y_n, \hat{y}_n) \end{aligned}$$

BCE loss = NLL loss

Binary classification, so it is called binary cross entropy loss

$$cross\_entropy(y_n, \hat{y}_n) = -\left(y_n log(\hat{y}_n) + (1 - y_n) log(1 - \hat{y}_n)\right)$$

Ground truth label

Predicted “soft” label, the (estimated) 'probability' of x_n belonging to class-1

n : the index of data point x_n

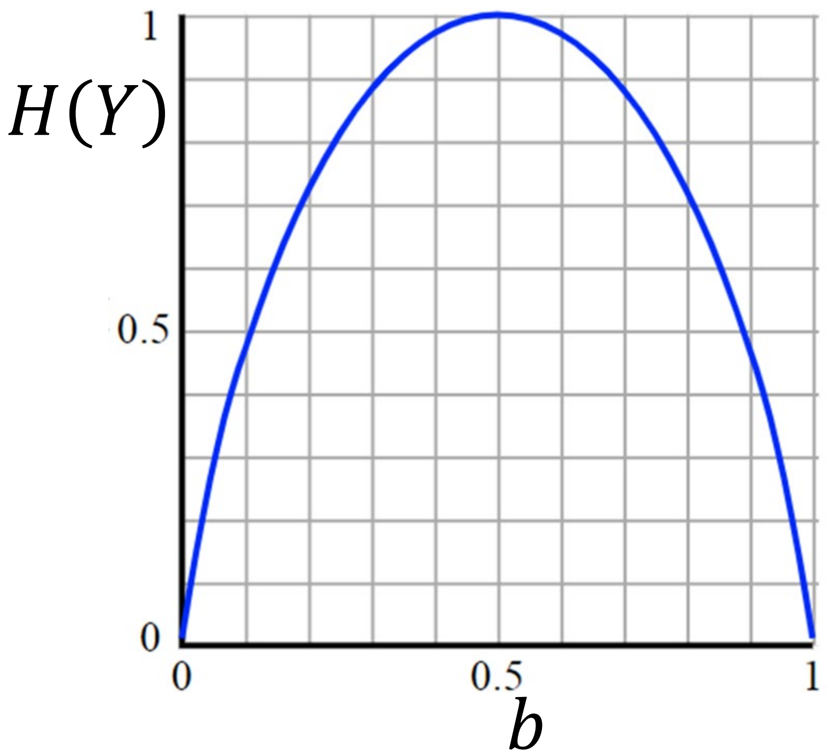
Entropy of a discrete random variable

- a discrete random variable Y

  the value of Y could be $a_1, a_2, \ldots., a_K$
- it has a probability mass function (PMF): $p(y = a_k) = p_k$
- the entropy is defined to be

$$H(Y) = -\sum_{k=1}^{K} p_k log(p_k) \geq 0$$

$$H(Y) \equiv H(p)$$



- Example: $P(Y = 1) = b$ and $P(Y = 0) = 1 - b$

  $H(Y) = -blog(b) - (1 - b)log(1 - b)$

Relative Entropy of two PMFs

- We have two PMFs $p(y)$ and $q(y)$ where y takes values in the same set
- The difference between the two probability mass functions can be quantified by the so-called relative-entropy:

$$D(p||q) = \sum_{y} p(y) log\left(\frac{p(y)}{q(y)}\right)$$

  - It is also called Kullback-Leibler (KL) distance/divergence
  - $D(p||q) \geq 0$
  - $D(p||q) = 0$ if and only if the two distributions are the same
- For this classification application,

  $p(y)$ is the true PMF of the random variable Y

  $q(y)$ is an estimation/approximation of the true PMF

Cross Entropy of two PMFs

- $D(p||q) = \sum_y p(y) log\left(\frac{p(y)}{q(y)}\right)$

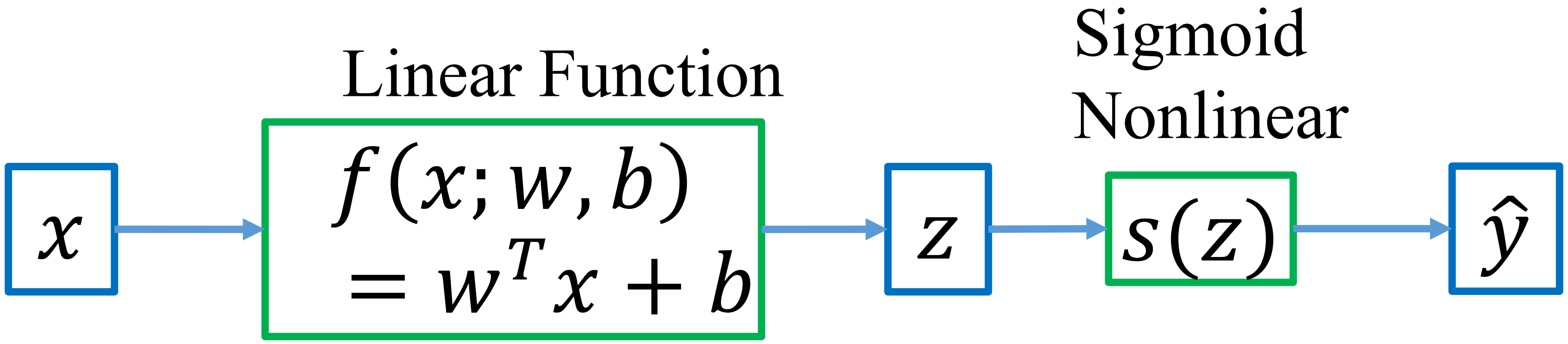
  $= \sum_y p(y) log\big(p(y)\big) - \sum_y p(y) log(q(y))$

  $= -H(p) \; - \sum_y p(y) log(q(y))$
- Define cross entropy $H(p, q) = -\sum_y p(y) log(q(y))$
- cross entropy is a distance measure if $H(p)$ is a constant
- Example: binary classification,$\{x_1, x_2\}$ and $\{y_1 = 1, y_2 = 0\}$
  - $p(y|x)$ is the true PMF of the data labels:

    $p(y_1 = 1|x_1) = 1$ and $p(y_1 = 0|x_1) = 0$, thus $H_{X_1}(p)$ is a constant

    $p(y_2 = 0|x_2) = 1$ and $p(y_2 = 1|x_2) = 0$, thus $H_{X_2}(p)$ is a constant
  - $q(y|x)$ is the output of the classifier, the estimated PMF of the labels

Training the classifier: Linear + Sigmoid + Loss



Parameters: w and b

- select a loss function L

- obtain the optimal w :

  $\frac{\partial L}{\partial w} = 0$, then we get w …., but, there is no closed-form solution

Training the Classifier by simple gradient descent

- Step-1: initialize parameters w and b using random numbers
- Step-2: compute $\frac{\partial L}{\partial w}$ and $\frac{\partial L}{\partial b}$
  - $\frac{\partial L}{\partial w}$ is a vector/direction: if w moves along this direction, then the loss function L will increase (this is the meaning of derivative)
  - We want to minimize L.
  - So, w should go in the opposite direction $-\frac{\partial L}{\partial w}$
- Step-3: update w and b

$$\mathrm{w} \leftarrow w - \eta \frac{\partial L}{\partial w}, \text{ and } b \leftarrow b - \eta \frac{\partial L}{\partial b}$$

where the scalar η is called learning rate, $\eta > 0$

- Repeat step-2 and step-3 until the algorithm converges

other gradient based optimization methods

- Limited-memory BFGS
- SAG: ~ Stochastic Average Gradient
- newton-cg: Newton-conjugate gradient

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

solver : *str, {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default: 'liblinear'.*

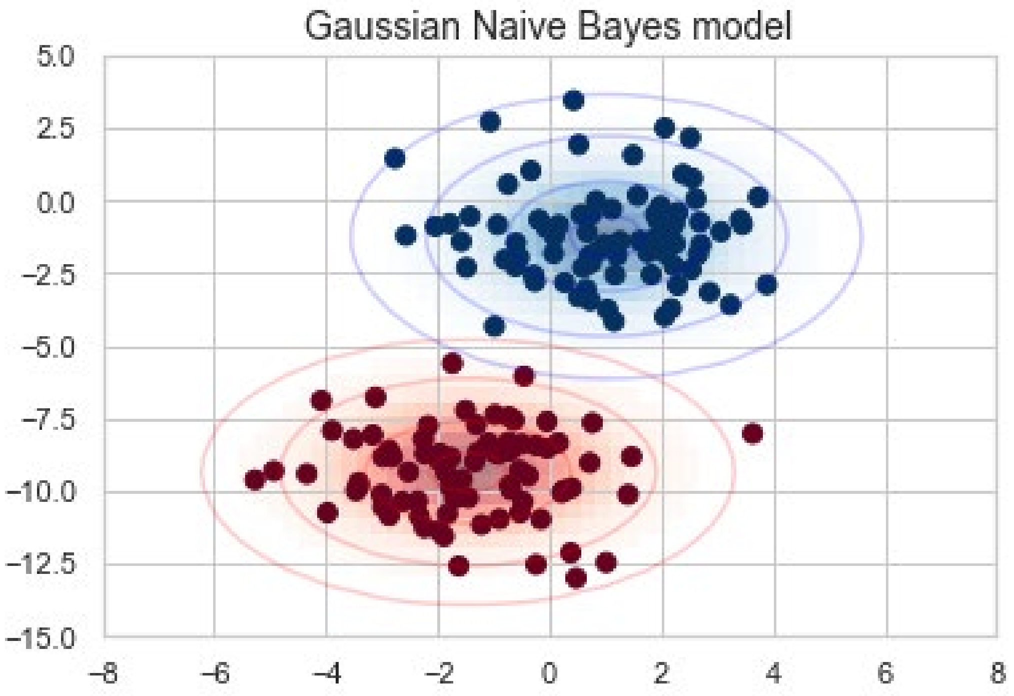
Algorithm to use in the optimization problem.

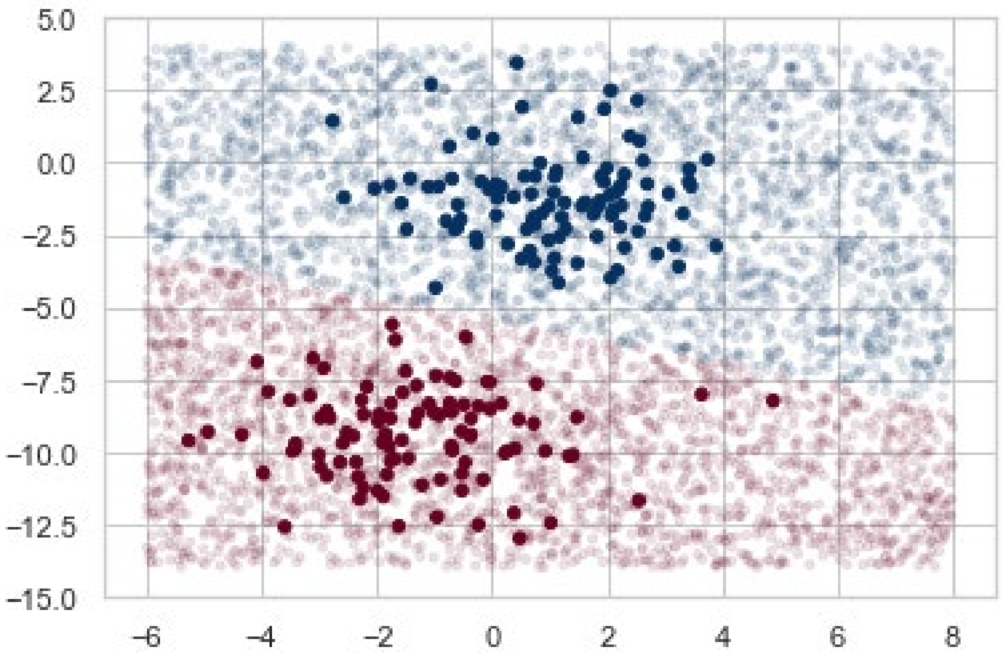
- For small datasets, 'liblinear' is a good choice, whereas 'sag' and 'saga' are faster for large ones.
- For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs' handle multinomial loss; 'liblinear' is limited to one-versus-rest schemes.
- 'newton-cg', 'lbfgs' and 'sag' only handle L2 penalty, whereas 'liblinear' and 'saga' handle L1 penalty.

see demo LR_Classifier_1D_2D.ipynb

Generate 2D data points
using two Gaussian PDFs

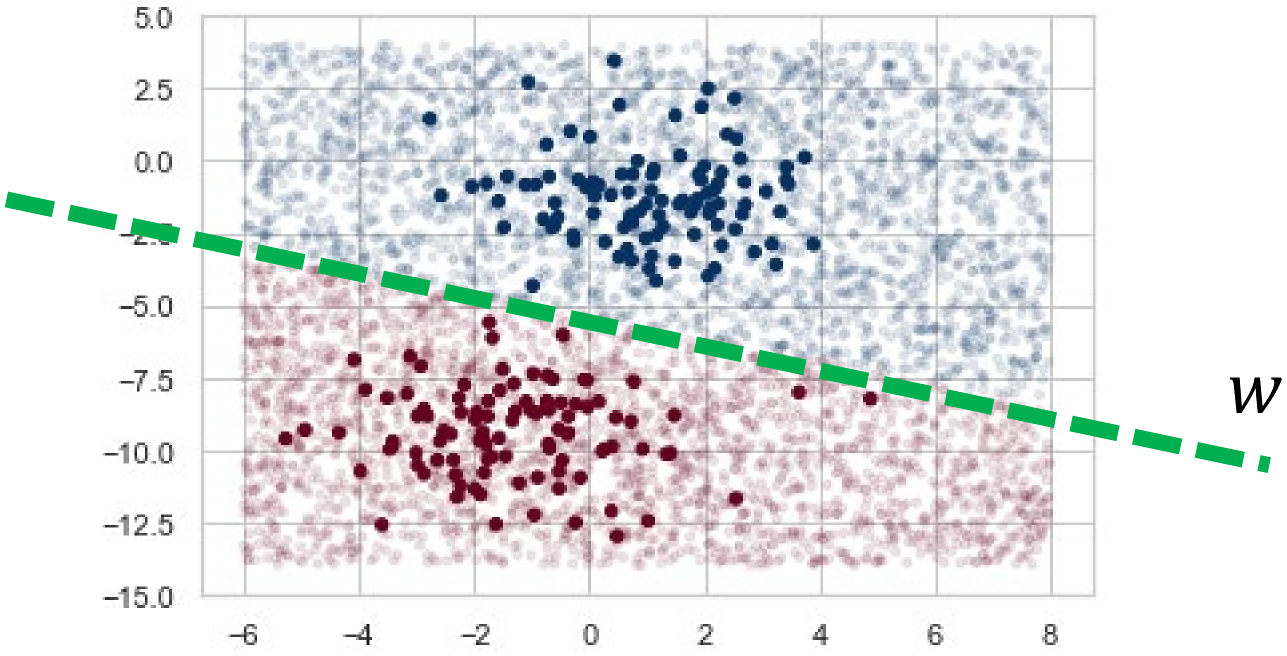
a LR Classifier





The decision boundary is a straight line/ hyperplane
Why ?

LR_Classifier_1D_2D.ipynb

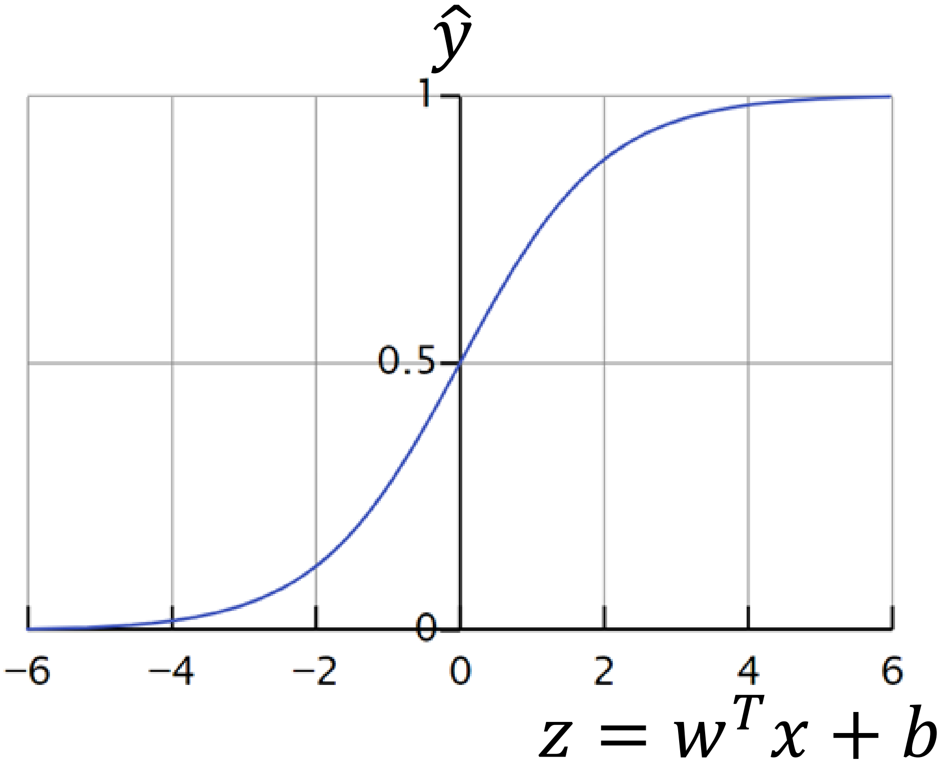


$$w^T x + b = 0$$

A logistic regression classifier is a linear classifier: decision boundary is a straight line/ hyperplane

$$z = w^T x + b$$

$$\hat{y} = \frac{1}{1 + e^{-z}}$$



Which loss function is the best ?

- mean squared error (MSE) loss

$$L = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2$$

- mean absolute error (MAE) loss or L1 loss

$$L = \frac{1}{N}\sum_{n=1}^{N}|\hat{y}_n - y_n|$$

- binary cross entropy (BCE) loss (used by logistic regression)

$$L = -\frac{1}{N}\sum_{n=1}^{N}\left(y_n log(\hat{y}_n) + (1 - y_n) log(1 - \hat{y}_n)\right)$$

- negative log-likelihood (NLL) loss = BCE loss

Convex Sets

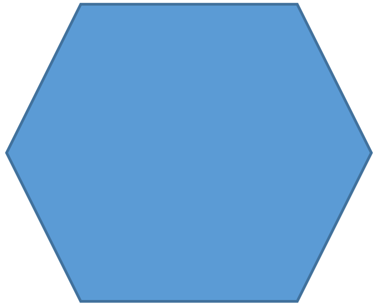
- Definition: a set A is convex if

$$\forall\, x, y \in A, \text{ then } \lambda x + (1-\lambda) y \in A$$

$$\lambda \text{ is any scalar in } 0 \le \lambda \le 1$$

which means the line segment between any two points is also in the set.

- Examples:



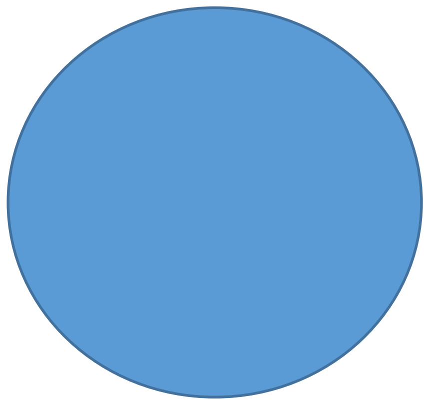
convex

non-convex

Convex Set examples

- an Euclidean ball:

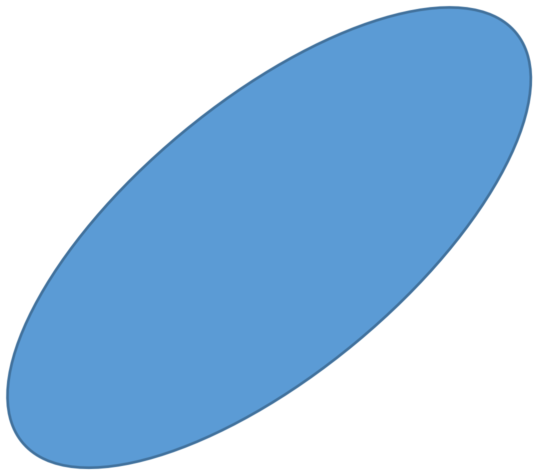
$$A = \{y \ \mid \|y - c\|_2 \leq r\}$$



- an ellipsoid:

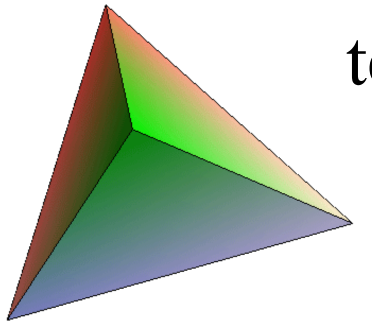
$$A = \{y \ \mid (y - \mu)^T \Sigma^{-1} (y - \mu) \leq 1\}$$

The eigenvector and eigenvalues of Σ determine the direction and shape



Convex Set examples

- some polyhedron



tetrahedron

$$A = \{y \mid a_i^T y \le b_i, i = 1,2, \ldots\}$$

it is an intersection of a finite set of hyperplanes

i.e., a finite set of linear equalities and inequalities

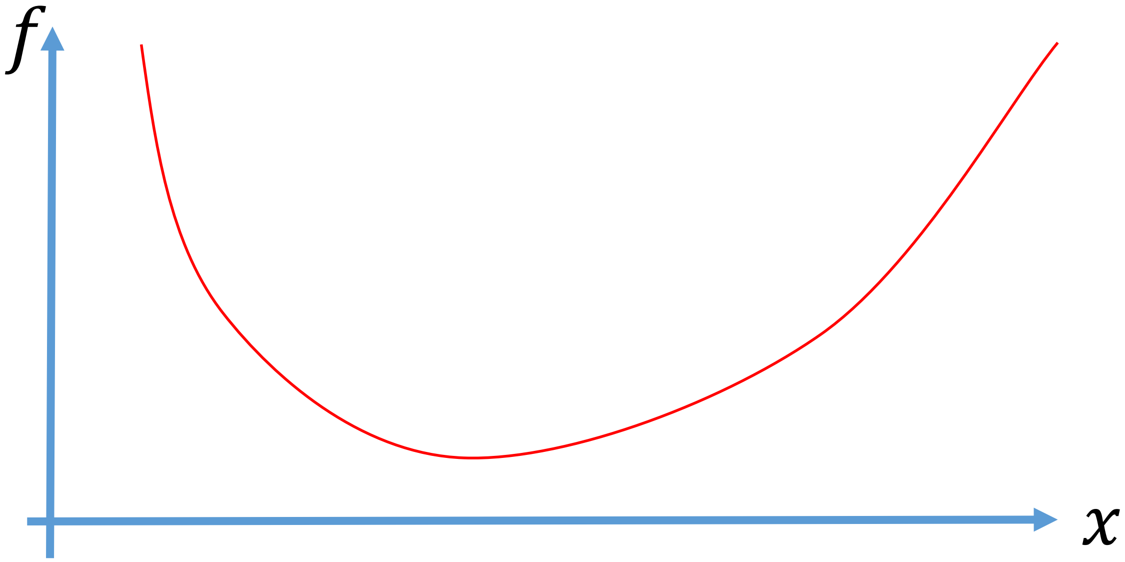
Convex Function (Domain is a convex set)

- Definition: a function $f\colon \mathcal{R}^N \to \mathcal{R}$ is convex if :

$$f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda) f(y)$$

λ is any scalar in $0 \le \lambda \le 1$, any x and y in domain of f

- Geometric meaning:



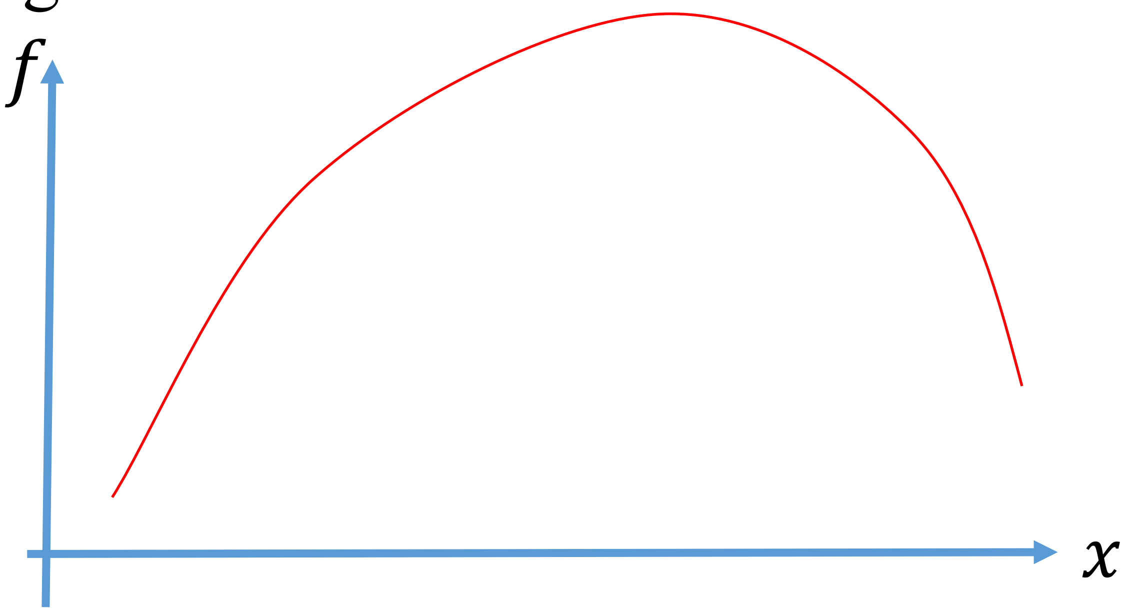
Concave Function

- Definition: a function $f\colon \mathcal{R}^N \to \mathcal{R}$ is concave if :

$$f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda)f(y)$$

λ is any scalar in $0 \leq \lambda \leq 1$, any x and y in domain of f

- Geometric meaning:



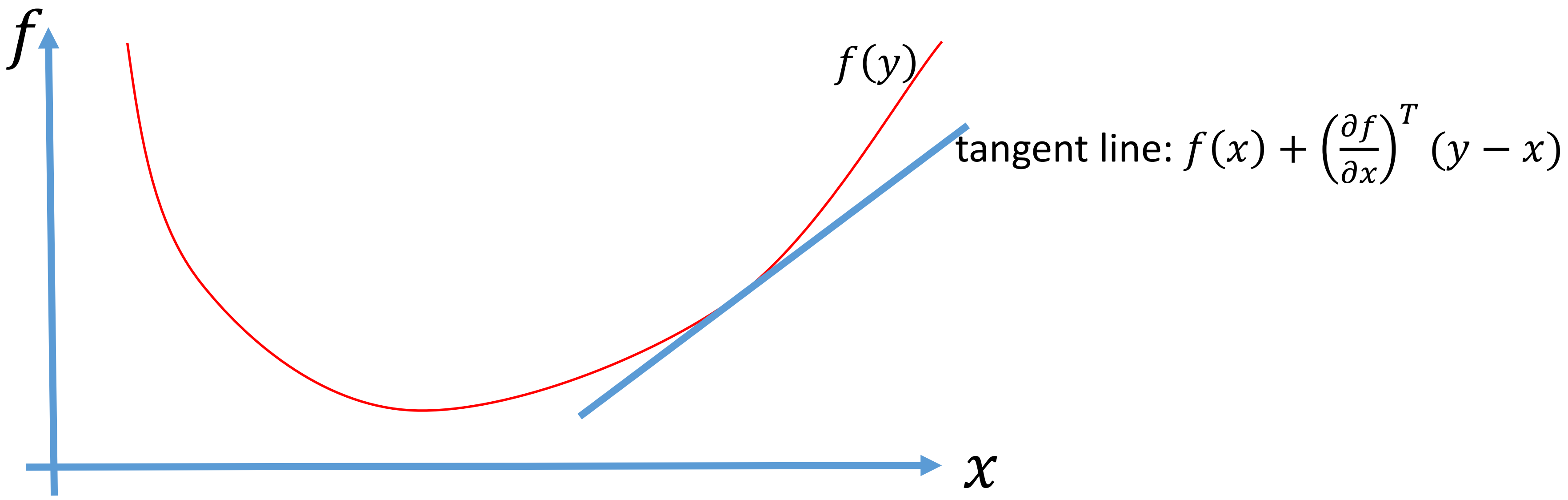
Convex Function

- a function $f\colon \mathcal{R}^N \to \mathcal{R}$ is convex iif :

$$f(y) \geq f(x) + (\nabla f(x))^T (y - x)$$

any x and y in domain of f

- Geometric meaning:



Convex Function

- a function $f\colon \mathcal{R}^N \to \mathcal{R}$ is convex **iif** : the Hessian matrix

  $\nabla^2 f(x)$ is positive semi-definite, for any x in domain of f

positive semi-definite: all eigenvalues are non negative

$\nabla^2 f(x)$ is call Hessian matrix, it is symmetric

an example:

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} \\ \frac{\partial^2 f}{\partial x_3 \partial x_1} & \frac{\partial^2 f}{\partial x_3 \partial x_2} & \frac{\partial^2 f}{\partial x_3^2} \end{bmatrix}, \text{ where } x = [x_1, x_2, x_3]^T$$

- the function/curve has positive (upward) curvature at every point.

Convex Optimization

- Definition: an optimization problem is specified by

$$\text{minimize } f(x)$$
$$\text{subject to } g_i(x) \leq 0, \text{ i=1,2,…,m}$$
$$h_j(x) = 0, \text{ j=1,2,…, p}$$

- A convex optimization problem has the following requirements:
  - the objective $f(x)$ is convex
  - every inequality constraint function $g_i(x)$ is convex
  - every equality constraint function $h_j(x)$ is affine: $h_j(x) = a_j^T x + b_j$
- Theorem: for convex optimization problem, any local optimum is also a global optimum, which means the global optimum can be found by gradient-based optimization algorithms.

Convex Function Examples

- Exponential functions: $f(x) = e^{ax}$ for every $a \in \mathcal{R}$
- Power functions: $f(x) = x^a$ when $a \geq 1$ or $a \leq 0$
- Power of absolute value: $f(x) = |x|^p$ for $p \geq 1$
- Negative log: $f(x) = -\log(x)$ for $x > 0$
- Negative entropy : $\mathrm{xlog}(x)$ is convex
- Vector Norms
- Log-sum-exp: $f(x) = -\log(e^{x_1} + e^{x_2} + \cdots)$
- Max: $f(x) = \max\{x_1, \ldots, x_N\}$
- log-determinant: $f(X) = \log(detX)$ for all positive definite matrices
- Summation of convex functions is convex
- Composition of two functions:

  $f(x) = h(g(x))$ is convex if (1) or (2) is true:

  (1) $g(x)$ is convex, and $h(x)$ is convex and non-decreasing

  (2) $g(x)$ is concave, and $h(x)$ is convex and non-increasing

binary cross entropy loss function is Convex in w and b

- BCE loss:

$$L(w, b) = -\frac{1}{N}\sum_{n=1}^{N}\left(y_n log(\hat{y}_n) + (1 - y_n) log(1 - \hat{y}_n)\right)$$

where $\hat{y}_n = \frac{1}{1+e^{-z_n}}$ and $z_n = w^T x_n + b$, $y_n = 0 \; or \; 1$

BCE loss is a convex function of w and b

The convexity is not dependent on the training dataset

$-log(\hat{y}_n)$ is convex in w for any n

$-log(1 - \hat{y}_n)$ is convex in w for any n

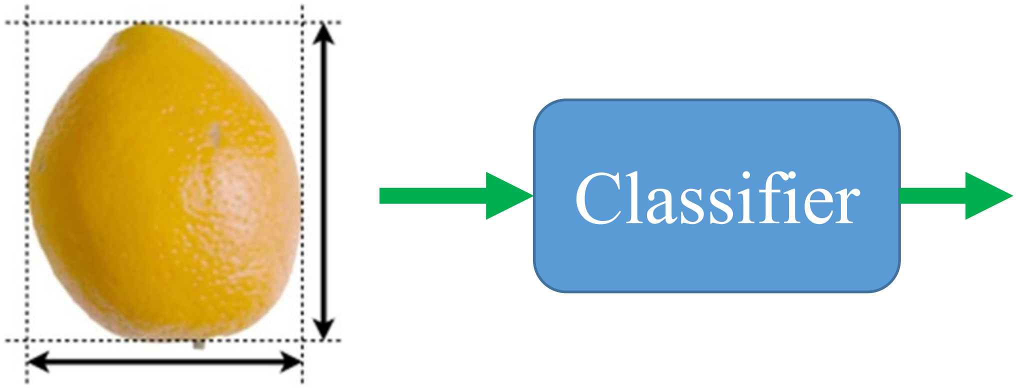
(proof: take the second order derivative and show it is nonnegative)

Multiclass Classification

- We have a set of <u>training</u> data points $\{x_1, x_2, x_3, \ldots, x_N\}$ and $x_n \in \mathcal{R}^M$
- We have <u>a set of 'ground-truth' labels</u> $\{y_1, y_2, y_3, \ldots, y_N\}$

  where $y_n \in \{1, 2, 3, \ldots, K\}$
- The <u>predicted labels</u> from the classifier are $\{\hat{y}_1, \hat{y}_2, \hat{y}_3, \ldots, \hat{y}_N\}$

  $\hat{y}_n$ from the classifier is a soft label vector



$$\hat{y} = \begin{bmatrix} \mathbf{0.80} \\ 0.10 \\ 0.08 \\ 0.02 \end{bmatrix}$$

convert it to a categorical label

label/fruit
1:apple
2:mandarin
3:orange
4:lemon

Multiclass Classification

- We have a set of 'ground-truth' labels $\{y_1, y_2, y_3, \ldots, y_N\}$
  where $y_n \in \{1, 2, 3, \ldots, K\}$

- Convert every y_n to the format of one-hot encoding

label/fruit
1:apple
2:mandarin
3:orange
4:lemon

$$y_n = 4 \quad \Rightarrow \quad y_n = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

note: in sk-learn, labels are 0, 1, 2, 3 $y_n \in \{0, 1, 2, \ldots, K-1\}$

Multiclass Classification: the Softmax function

- The Softmax function of scalars $z_1, \dots, z_K$

$$s(z_1, \dots, z_K) = \begin{bmatrix} \frac{e^{z_1}}{\sum_{k=1}^{K} e^{z_k}} \\ \frac{e^{z_2}}{\sum_{k=1}^{K} e^{z_k}} \\ \vdots \\ \frac{e^{z_K}}{\sum_{k=1}^{K} e^{z_k}} \end{bmatrix}$$

$$\exp(z) \equiv e^z$$

Sum of the elements is 1

It is a vector-valued function

Multiclass Classification: the Softmax function

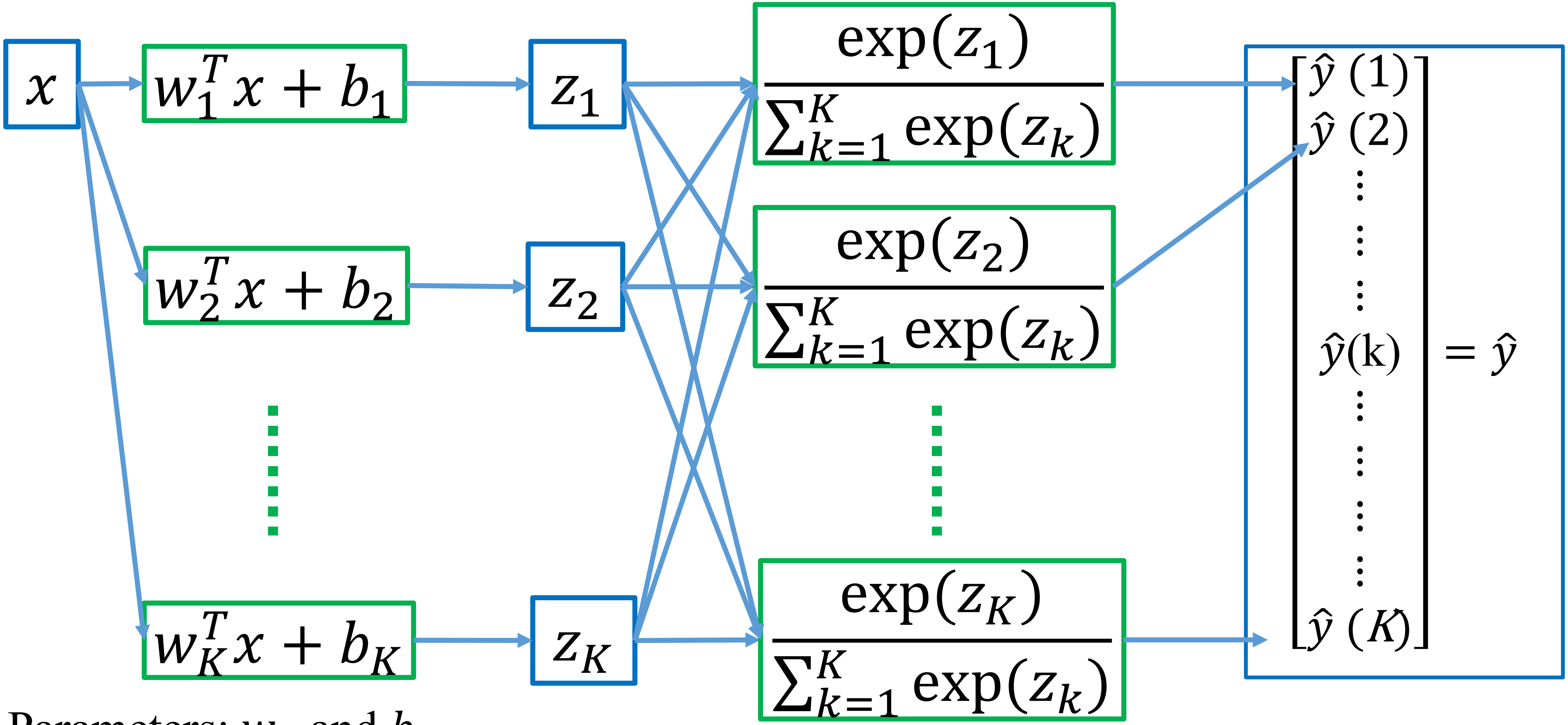
- The Softmax function of scalars z_1, z_2 when $K = 2$

$$s(z_1, z_2) = \begin{bmatrix} \dfrac{e^{z_1}}{e^{z_1} + e^{z_2}} \\[2ex] \dfrac{e^{z_2}}{e^{z_1} + e^{z_2}} \end{bmatrix}$$

$$\exp(z) \equiv e^{z}$$

It is a vector-valued function

The Multi-output Classifier



Parameters: w_k and b_k

$0 \leq \hat{y}(\mathrm{k}) \leq 1$ is the 'probability' of x belonging to class-k, $\sum_{k=1}^{K} \hat{y}(\mathrm{k}) = 1$

Training a Multiclass Logistic Regression Classifier

- cross-entropy (CE) loss function

$$L = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} y_{(n,k)} log\left(\hat{y}_{(n,k)}\right)$$

$y_{(n,k)}$ is the k-th element of the vector y_n (ground-truth)

$\hat{y}_{(n,k)}$ is the k-th element of the vector $\hat{y}_n$ (prediction)

- training: gradient descent

$$w_k \leftarrow w_k - \eta \frac{\partial L}{\partial w_k}, \text{ and } b_k \leftarrow b_k - \eta \frac{\partial L}{\partial b_k}$$

Softmax-based Logistic Regression Classifier = Sigmoid-based Logistic Regression Classifier When the number of classes is 2

$$s(z_1, z_2) = \begin{bmatrix} \dfrac{e^{z_1}}{e^{z_1} + e^{z_2}} \\ \\ \dfrac{e^{z_2}}{e^{z_1} + e^{z_2}} \end{bmatrix} = \begin{bmatrix} \dfrac{e^{z_1} \times e^{-z_1}}{(e^{z_1} + e^{z_2}) \times e^{-z_1}} \\ \\ \dfrac{e^{z_2} \times e^{-z_2}}{(e^{z_1} + e^{z_2}) \times e^{-z_2}} \end{bmatrix} = \begin{bmatrix} \dfrac{1}{1 + e^{z_2 - z_1}} \\ \\ \dfrac{1}{1 + e^{z_1 - z_2}} \end{bmatrix}$$

$$\frac{1}{1 + e^{z_1 - z_2}} + \frac{1}{1 + e^{z_2 - z_1}} = 1$$

Then define $z = z_2 - z_1$

(multinominal) Multiclass logistic regression in sklearn

sklearn.linear_model.LogisticRegression

class `sklearn.linear_model.` **`LogisticRegression`**(*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None*) [source]

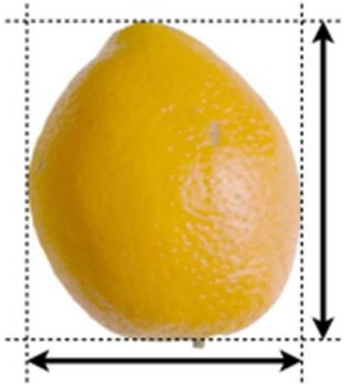
Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross- entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag' and 'newton-cg' solvers.)

Try it on the Fruit Dataset

A bucket of fruits

The fruit dataset was created by Dr. Iain Murray at the University of Edinburgh. He bought a few dozen oranges, lemons and apples, and recorded their features in a table.



The fruit dataset (a large table) {1:apple, 2:mandarin, 3:orange, 4:lemon},

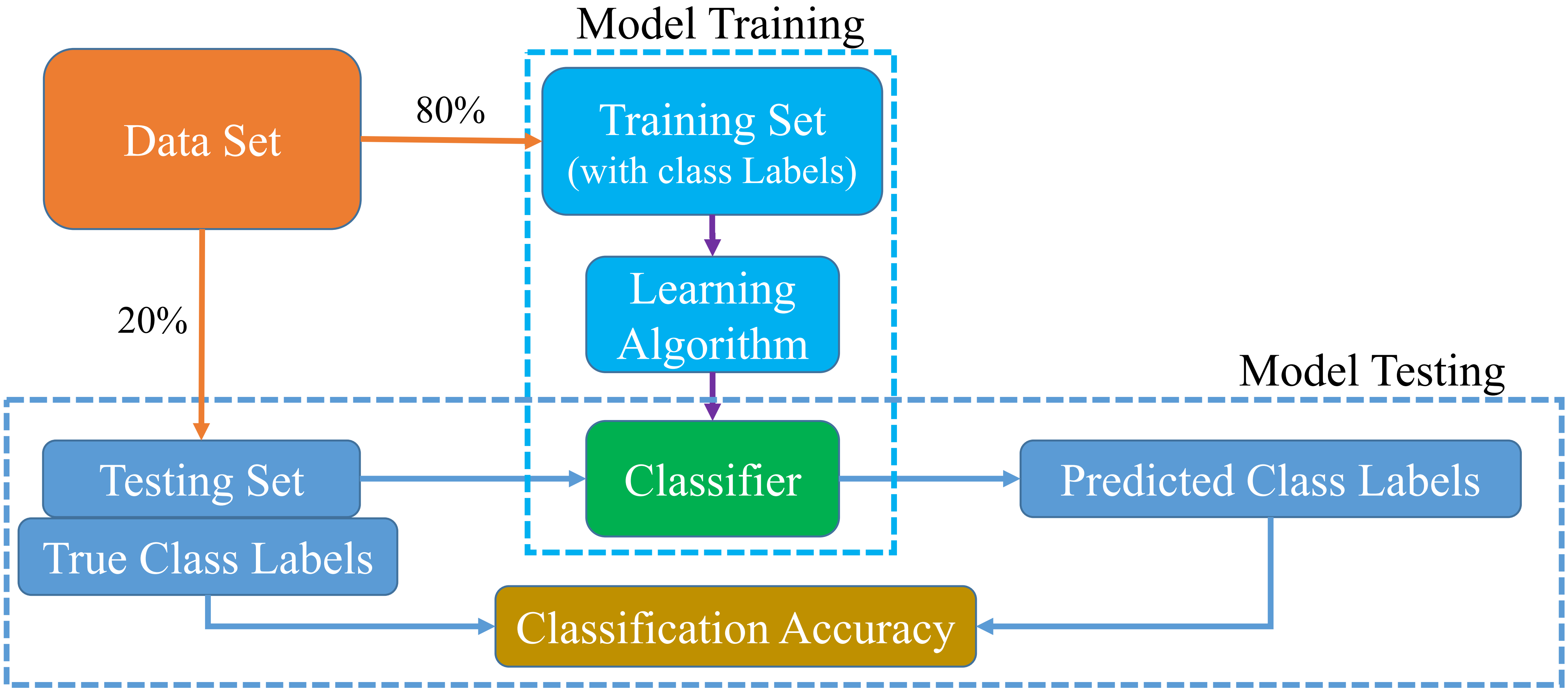
Each row contains the information of a fruit sample/instance

fruit label	fruit_name	subtype	mass (g)	width (cm)	height (cm)	color_score
1	apple	granny_smith	192	8.4	7.3	0.55
4	lemon	spanish_belsan	194	7.2	10.3	0.70

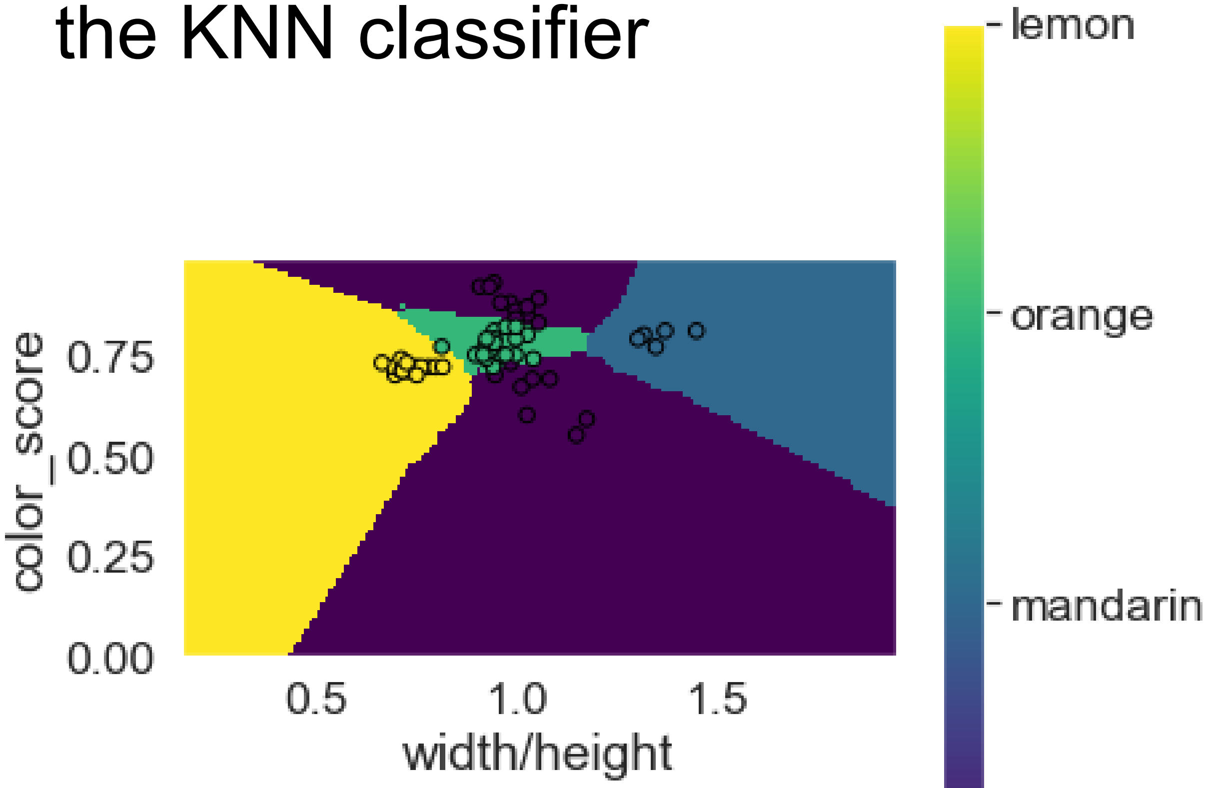
http://usapple.org/the-industry/apple-varieties/

The flowchart of a classification study

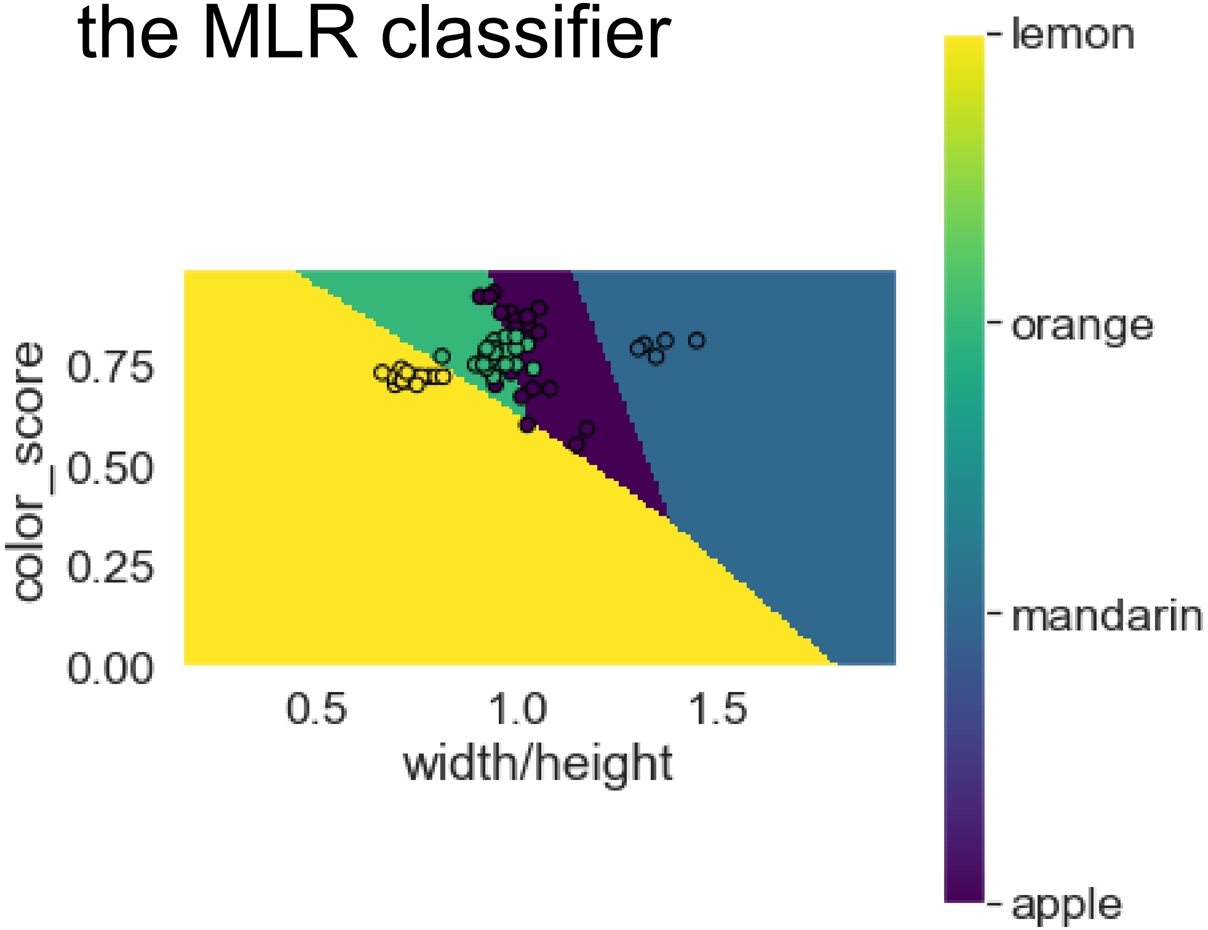
- Classification is a subcategory of supervised learning where the goal is to predict the class labels of new samples.



Decision Boundary Plot of the KNN classifier



Decision Boundary Plot of the MLR classifier



see demo in MLR_Classifier.ipynb