

Applied Machine Learning

Logistic Regression

Siamak Ravanbakhsh

COMP 551 (winter 2020)

Learning objectives

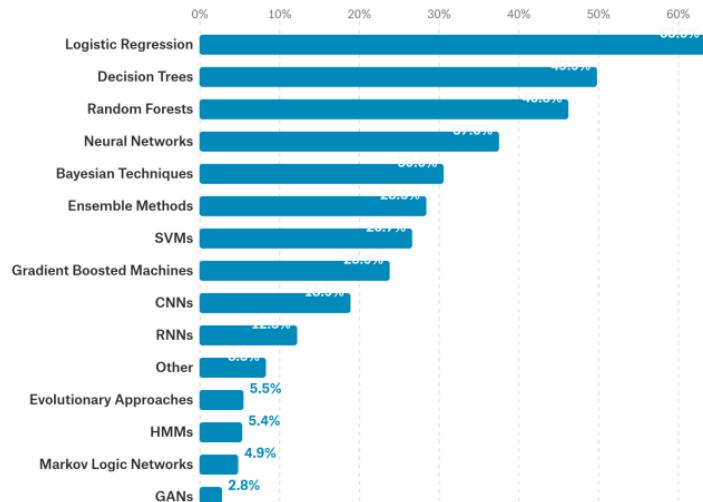
- what are linear classifiers
- logistic regression
 - model
 - loss function
- maximum likelihood view
- multi-class classification

Motivation

we have seen KNN for classification

we see more classifiers today (linear classifiers)

Logistic Regression is **the** most commonly reported data science method used at work



souce: 2017 Kaggle survey

Classification problem

dataset of inputs
and discrete targets
binary classification

$$\begin{aligned}x^{(n)} &\in \mathbb{R}^D \\y^{(n)} &\in \{0, \dots, C\} \\y^{(n)} &\in \{0, 1\}\end{aligned}$$

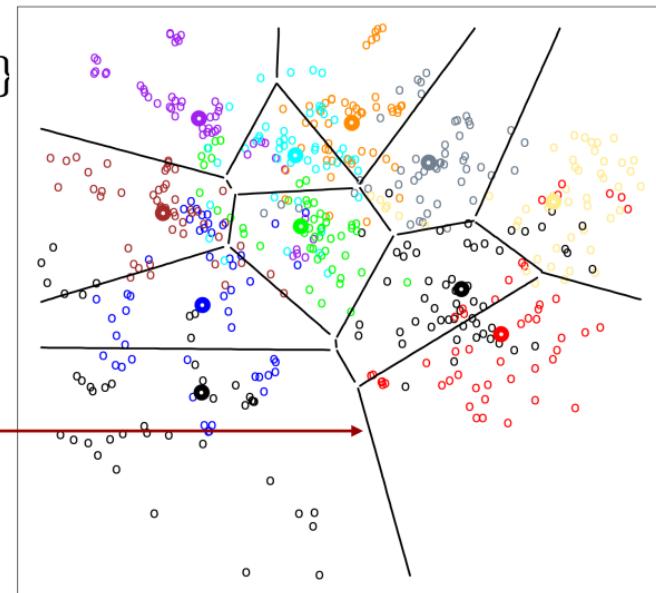
linear classification:

decision boundaries are linear

linear decision boundary $w^\top x + b$

how do we find these boundaries?

different approaches give different linear classifiers



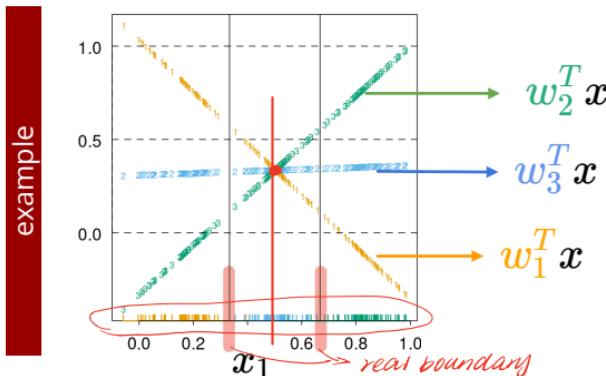
first idea

Using linear regression

fit a linear model to each class c : $w_c^* = \arg \min_{w_c} \frac{1}{2} \sum_{n=1}^N (w_c^\top x^{(n)} - \mathbb{I}(y^{(n)} = c))^2$

class label for a new instance is then $\hat{y}^{(n)} = \arg \max_c w_c^\top x^{(n)}$

decision boundary between any two classes $w_c^\top x = w_{c'}^\top x$



recall
 $x = [1, x_1]^\top$

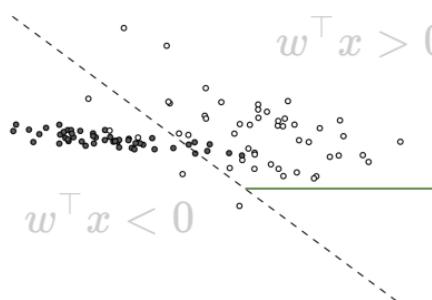
- where are the decision boundaries?
- but the instances are **linearly separable**
 - we should be able to find these boundaries
- where is the problem?

one feature (one dimensional)

first idea

Using linear regression

Binary classification $y \in \{0, 1\}$ so we are fitting 2 linear models $a^\top x, b^\top x$



$$a^\top x - b^\top x = 0 \text{ decision boundary is here}$$

$$(a - b)^\top x = 0$$

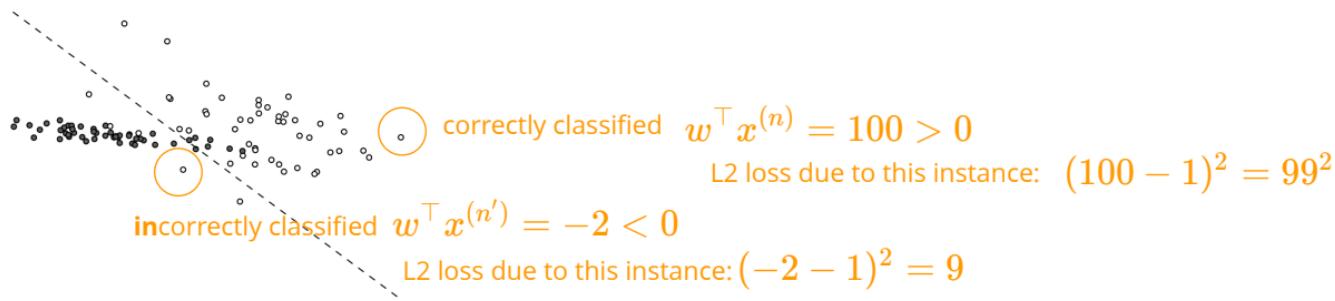
so one weight vector is enough

$$w^\top x = 0 \begin{cases} y = 1 & w^\top x > 0 \\ y = 0 & w^\top x < 0 \end{cases}$$

first idea

Using linear regression

Binary classification $y \in \{0, 1\}$ so we are fitting 2 linear models $a^\top x, b^\top x$



correct prediction can have higher loss than the incorrect one!



solution: we should try squashing all positive instance together and all the negative ones together

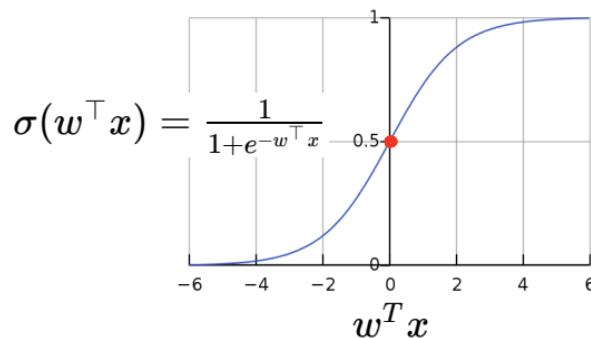
Logistic function

Idea: apply a squashing function to $w^\top x \rightarrow \sigma(w^\top x)$

desirable property of $\sigma : \mathbb{R} \rightarrow \mathbb{R}$

- | all $w^\top x > 0$ are squashed close together
- | all $w^\top x < 0$ are squashed together

logistic function has these properties



the decision boundary is

$$w^\top x = 0 \Leftrightarrow \sigma(w^\top x) = \frac{1}{2}$$

still a linear decision boundary

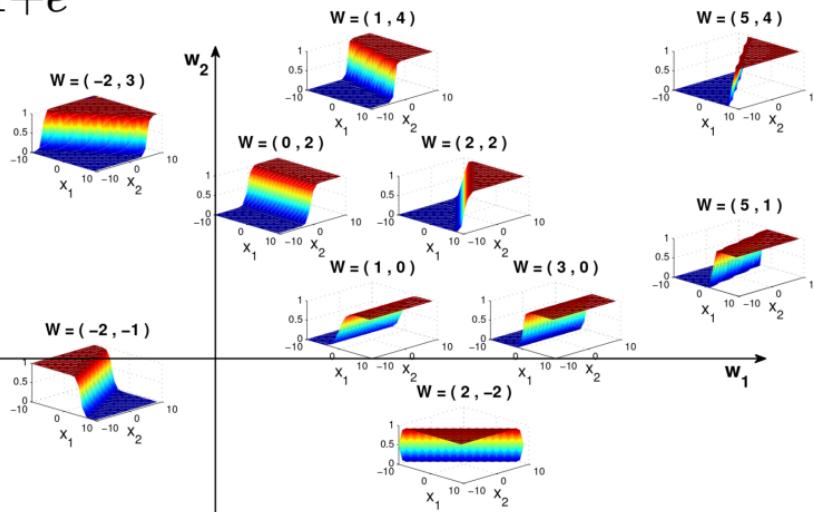
Logistic regression: model

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

ζ
logit

logistic function
squashing function
activation function

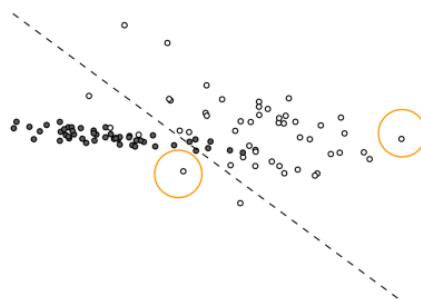
note the linear decision boundary



Logistic regression: the loss

first idea use the misclassification error

zero one loss function



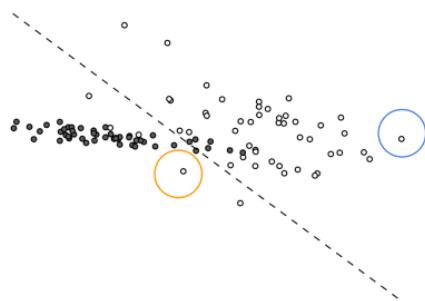
$$L_{0/1}(\hat{y}, y) = \mathbb{I}(y \neq \text{sign}(\hat{y} - \frac{1}{2})) \begin{cases} 1 & \text{wrong} \\ 0 & \text{correct} \end{cases}$$

$$\sigma(w^\top x)$$

- not a continuous function (in w)
- hard to optimize

Logistic regression: the loss

second idea use the L2 loss

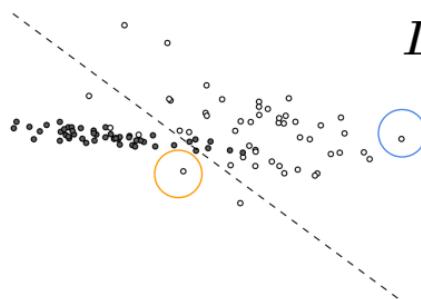


$$L_2(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$
$$\sigma(w^\top x)$$

- thanks to squashing, the previous problem is resolved
- loss is continuous
- still a problem: hard to optimize (non-convex in w)

Logistic regression: the loss

third idea use the cross-entropy loss



$$L_{CE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$
$$\sigma(w^\top x)$$

- it is convex in w
- probabilistic interpretation (soon!)



Cost function

we need to optimize the cost wrt. parameters

first: simplify

$$\begin{aligned} J(w) &= \sum_{n=1}^N -y^{(n)} \log(\sigma(w^\top x^{(n)})) - (1 - y^{(n)}) \log(1 - \sigma(w^\top x^{(n)})) \\ &\quad \downarrow \text{substitute logistic function} \qquad \qquad \qquad \downarrow \text{substitute logistic function} \\ \log\left(\frac{1}{1+e^{-w^\top x}}\right) &= -\log(1 + e^{-w^\top x}) \\ \log\left(1 - \frac{1}{1+e^{-w^\top x}}\right) &= \log\left(\frac{1}{1+e^{w^\top x}}\right) = -\log(1 + e^{w^\top x}) \end{aligned}$$

simplified cost $J(w) = \sum_{n=1}^N y^{(n)} \log(1 + e^{-w^\top x}) + (1 - y^{(n)}) \log(1 + e^{w^\top x})$

implementing the Cost function

simplified cost: $J(w) = \sum_{n=1}^N y^{(n)} \log(1 + e^{-w^\top x}) + (1 - y^{(n)}) \log(1 + e^{w^\top x})$

• • •

```
def cost(w, # D
         X, # N x D
         y # N
        ):
    z = np.dot(X,w) #N x 1
    J = np.mean(y * np.log1p(np.exp(-z)) + (1-y) * np.log1p(np.exp(z)))
    return J
```

more stable



why not `np.log(1 + np.exp(-z))` ?

• • •

for small ϵ , $\log(1 + \epsilon)$ suffers from floating point inaccuracies

```
In [3]: np.log(1+1e-100)
Out[3]: 0.0
In [4]: np.log1p(1e-100)
Out[4]: 1e-100
```



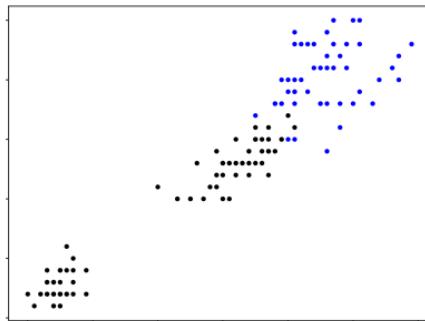
$$\log(1 + \epsilon) = \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - \dots$$

Example: binary classification

classification on **Iris flowers dataset**:

(*a classic dataset originally used by Fisher*)

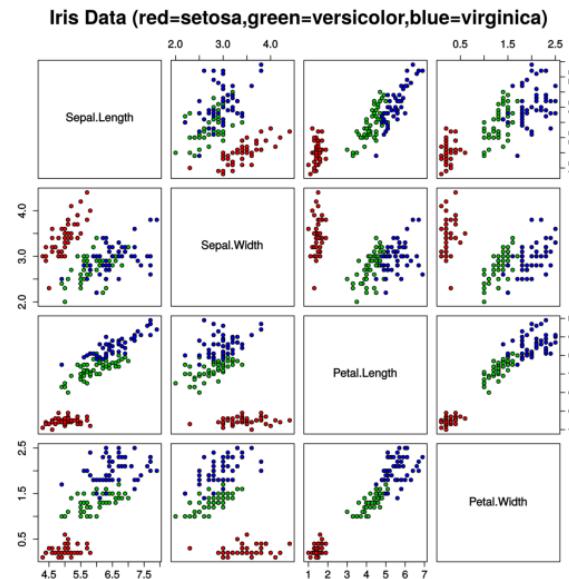
$N_c = 50$ samples with $D=4$ features, for each of $C=3$ species of Iris flower



our setting

2 classes
(blue vs others)

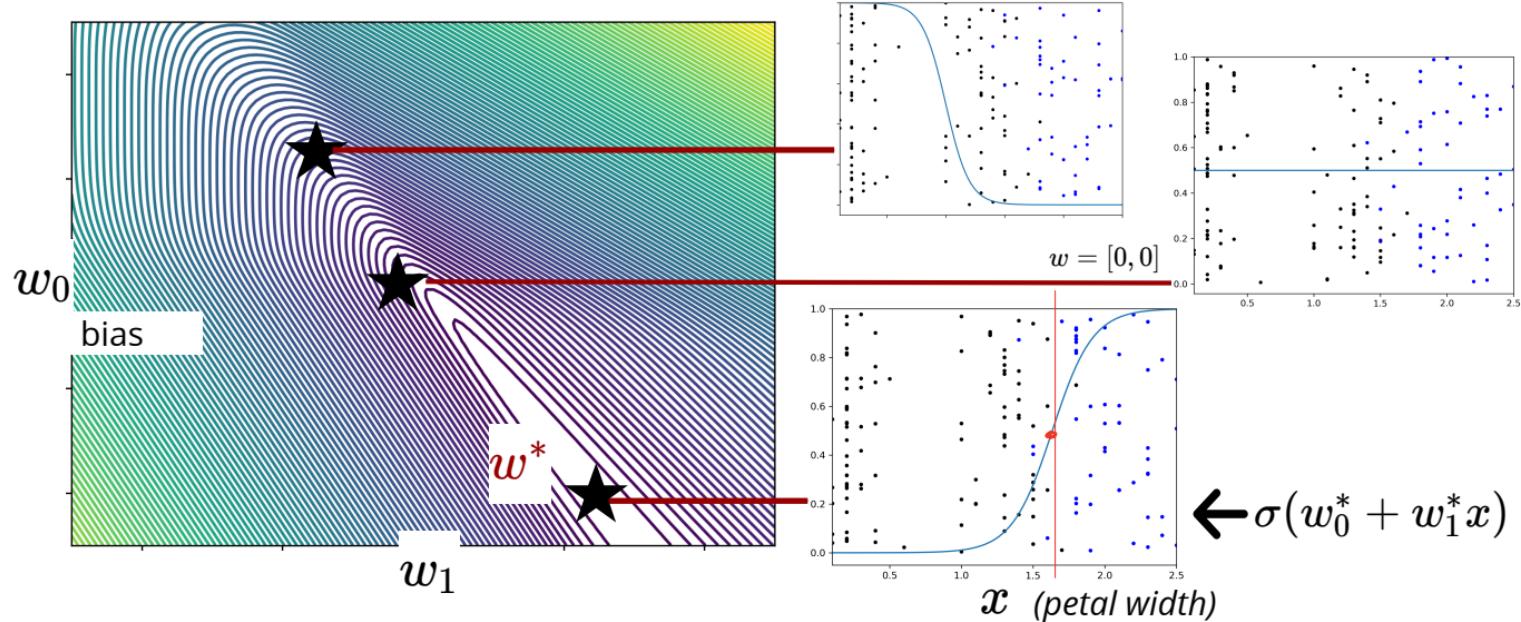
1 features
(petal width + bias)



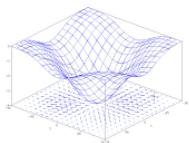
Example: binary classification

we have two weights associated with bias + petal width

$J(w)$ as a function of these weights



Gradient



how did we find the optimal weights?
(in contrast to linear regression, no closed form solution)

$$\text{cost: } J(w) = \sum_{n=1}^N y^{(n)} \log (1 + e^{-w^\top x^{(n)}}) + (1 - y^{(n)}) \log (1 + e^{w^\top x^{(n)}})$$

$$\begin{aligned} \text{taking partial derivative } \frac{\partial}{\partial w_d} J(w) &= \sum_n -y^{(n)} x_d^{(n)} \frac{e^{-w^\top x^{(n)}}}{1+e^{-w^\top x^{(n)}}} + x_d^{(n)} (1 - y^{(n)}) \frac{e^{w^\top x^{(n)}}}{1+e^{w^\top x^{(n)}}} \\ &= \sum_n -x_d^{(n)} y^{(n)} (1 - \hat{y}^{(n)}) + x_d^{(n)} (1 - y^{(n)}) \hat{y}^{(n)} = x_d^{(n)} (\hat{y}^{(n)} - y^{(n)}) \end{aligned}$$

gradient $\nabla J(w) = \sum_n x^{(n)} (\hat{y}^{(n)} - y^{(n)}) \sigma(w^\top x^{(n)})$

Cross-entropy is matched to logistic

compare to gradient for linear regression $\nabla J(w) = \sum_n x^{(n)} (\hat{y}^{(n)} - y^{(n)}) w^\top x^{(n)}$

L2 loss is matched to (no activation)

Probabilistic view of logistic regression

probabilistic interpretation of logistic regression $\hat{y} = p_w(y = 1 \mid x) = \frac{1}{1+e^{-w^\top x}} = \sigma(w^\top x)$

logit function is the inverse of logistic $\log \frac{\hat{y}}{1-\hat{y}} = w^\top x$

the log-ratio of class probabilities is linear

likelihood probability of data as a function of model parameters

$$L(w) = p_w(y^{(n)} \mid x^{(n)}) = \text{Bernoulli}(y^{(n)}; \sigma(w^\top x^{(n)})) = \hat{y}^{(n)}^{y^{(n)}} (1 - \hat{y}^{(n)})^{1-y^{(n)}}$$

is a function of w

not a probability distribution function

$\hat{y}^{(n)}$ is the probability of $y^{(n)} = 1$

$1 - \hat{y}^{(n)}$ is the probability of $y^{(n)} = 0$.

$$\text{likelihood of the dataset } L(w) = \prod_{n=1}^N p_w(y^{(n)} \mid x^{(n)}) = \prod_{n=1}^N \hat{y}^{(n)}^{y^{(n)}} (1 - \hat{y}^{(n)})^{1-y^{(n)}}$$

assume instances are iid

Maximum likelihood & logistic regression

likelihood $L(w) = \prod_{n=1}^N p_w(y^{(n)} | x^{(n)}) = \prod_{n=1}^N \hat{y}^{(n)}^{y^{(n)}} (1 - \hat{y}^{(n)})^{1-y^{(n)}}$

maximum likelihood use the model that maximizes the likelihood of observations

$$w^* = \arg \max_w L(w)$$

likelihood value blows up for large N, work with log-likelihood instead (same maximum)

log likelihood $\max_w \sum_{n=1}^N \log p_w(y^{(n)} | x^{(n)})$

$$= \max_w \sum_{n=1}^N y^{(n)} \log(\hat{y}^{(n)}) + (1 - y^{(n)}) \log(1 - \hat{y}^{(n)})$$
$$= \min_w J(w) \quad \text{the cross entropy cost function!}$$

so using cross-entropy loss in logistic regression is maximizing conditional likelihood

Maximum likelihood & linear regression

squared error loss also has max-likelihood interpretation

cond. probability $p_w(y | x) = \mathcal{N}(y | w^\top x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-w^\top x)^2}{2\sigma^2}}$

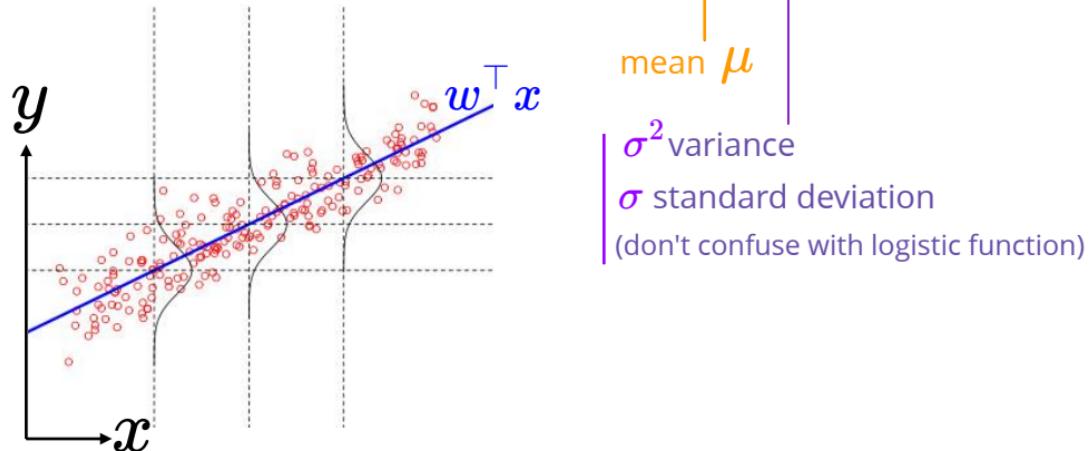


image: <http://blog.nguyenvq.com/blog/2009/05/12/linear-regression-plot-with-normal-curves-for-error-sideways/>

Maximum likelihood & linear regression

squared error loss also has max-likelihood interpretation

cond. probability $p_w(y | x) = \mathcal{N}(y | w^\top x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-w^\top x)^2}{2\sigma^2}}$

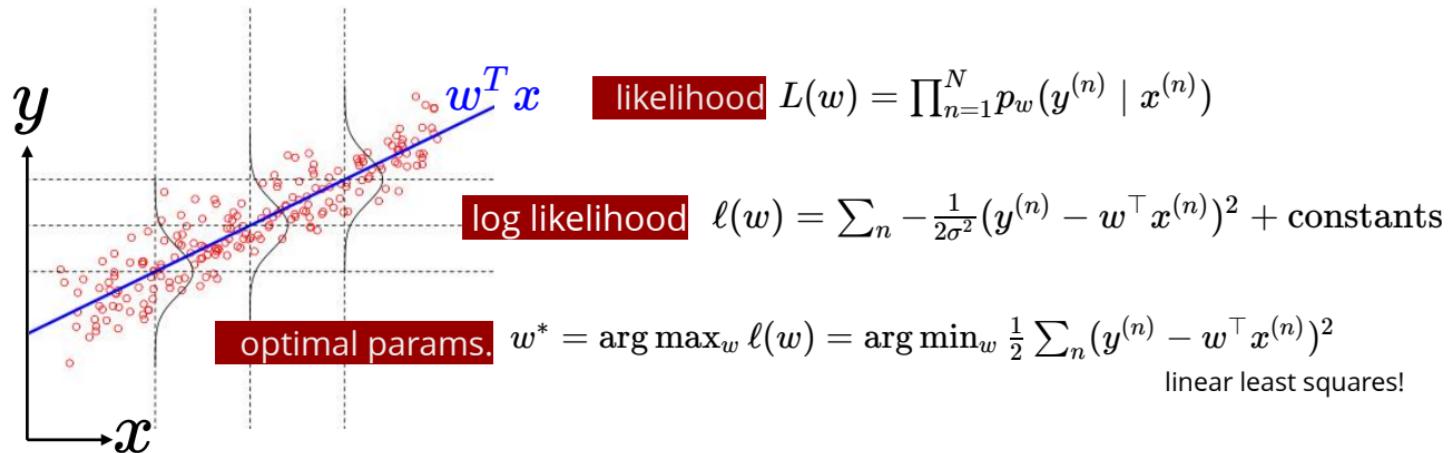


image: <http://blog.nguyenvq.com/blog/2009/05/12/linear-regression-plot-with-normal-curves-for-error-sideways/>

Multiclass classification

binary classification: Bernoulli likelihood:

$$\text{Bernoulli}(y \mid \hat{y}) = \hat{y}^y (1 - \hat{y})^{1-y} \xrightarrow{\substack{\text{subject to} \\ \bullet \text{ scalar}}} \hat{y} \in [0, 1]$$

using logistic function to ensure this $\hat{y} = \sigma(z) = \sigma(w^T x)$

C classes: categorical likelihood

$$\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^C \hat{y}_c^{\mathbb{I}(y=c)} \xrightarrow{\substack{\text{subject to} \\ \bullet \text{ vector}}} \sum_c \hat{y}_c = 1$$

achieved using softmax function
how to enforce it?

Softmax

generalization of logistic to > 2 classes:

- **logistic:** $\sigma : \mathbb{R} \rightarrow (0, 1)$ produces a single probability
 - probability of the second class is $(1 - \sigma(z))$

- **softmax:** $\mathbb{R}^C \rightarrow \Delta_C$ probability simplex $p \in \Delta_c \rightarrow \sum_{c=1}^C p_c = 1$

$$\hat{y}_c = \text{softmax}(z)_c = \frac{e^{z_c}}{\sum_{c'=1}^C e^{z_{c'}}} \text{ so } \sum_c \hat{y} = 1$$

if input values are large, softmax becomes similar to argmax

example $\text{softmax}([10, 100, -1]) \approx [0, 1, 0]$

numerical stability

so similar to logistic this is also a squashing function

```
...  
1 def softmax(  
2     z # C x ... array  
3 ):  
4     z = z - np.max(z, 0)  
5     yh = np.exp(z)  
6     yh /= np.sum(yh, 0)  
7     return yh
```

Multiclass classification

C classes: categorical likelihood

$$\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^C \hat{y}_c^{\mathbb{I}(y=c)} \quad \text{using softmax to enforce sum-to-one constraint}$$

$$\hat{y}_c = \text{softmax}([\mathbf{w}_{[1]}^\top \mathbf{x}, \dots, \mathbf{w}_{[C]}^\top \mathbf{x}])_c = \frac{e^{\mathbf{w}_{[c]}^\top \mathbf{x}}}{\sum_{c'} e^{\mathbf{w}_{[c']}^\top \mathbf{x}}}$$

so we have one parameter vector for each class

to simplify equations we write $\mathbf{z}_c = \mathbf{w}_{[c]}^\top \mathbf{x}$

$$\hat{y}_c = \text{softmax}([z_1, \dots, z_C])_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}}$$

Likelihood

C classes: categorical likelihood

Categorical($y \mid \hat{y}$) = $\prod_{c=1}^C \hat{y}_c^{\mathbb{I}(y=c)}$ using softmax to enforce sum-to-one constraint

$$\hat{y}_c = \text{softmax}([z_1, \dots, z_C])_c = \frac{e^{z_c}}{\sum_{c'} e^{z_{c'}}} \text{ where } z_c = w_{[c]}^\top x$$

substituting softmax in Categorical likelihood:

$$\begin{aligned} \text{likelihood} \quad L(\{w_c\}) &= \prod_{n=1}^N \prod_{c=1}^C \text{softmax}([z_1^{(n)}, \dots, z_C^{(n)}])_c^{\mathbb{I}(y^{(n)}=c)} \\ &= \prod_{n=1}^N \prod_{c=1}^C \left(\frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z_{c'}^{(n)}}} \right)^{\mathbb{I}(y^{(n)}=c)} \end{aligned}$$

One-hot encoding

likelihood

$$L(\{w_c\}) = \prod_{n=1}^N \prod_{c=1}^C \left(\frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z_{c'}^{(n)}}} \right)^{\mathbb{I}(y^{(n)}=c)}$$

log-likelihood

$$\ell(\{w_c\}) = \sum_{n=1}^N \sum_{c=1}^C \mathbb{I}(y^{(n)} = c) z_c^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}}$$

one-hot encoding for labels

$$y^{(n)} \rightarrow [\mathbb{I}(y^{(n)} = 1), \dots, \mathbb{I}(y^{(n)} = C)]$$



```
1 def one_hot(
2     y, #vector of size N class-labels [1,...,C]
3     ):
4     N, C = y.shape[0], np.max(y)
5     y_hot = np.zeros(N, C)
6     y_hot[np.arange(N), y-1] = 1
7     return y_hot
```

using this encoding from now on

log-likelihood

$$\ell(\{w_c\}) = \sum_{n=1}^N y^{(n)\top} z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}}$$

One-hot encoding

side note

we can also use this encoding for **categorical inputs** features

one-hot encoding for input features

$$x_d^{(n)} \rightarrow [\mathbb{I}(x_d^{(n)} = 1), \dots, \mathbb{I}(x_d^{(n)} = C)]$$

problem

these features are **not** linearly independent, why?

might become an issue for *linear regression*. why?

remove one of the one-hot encoding features

$$x_d^{(n)} \rightarrow [\mathbb{I}(x_d^{(n)} = 1), \dots, \mathbb{I}(x_d^{(n)} = C - 1)]$$

solution

Implementing the cost function

softmax cross entropy cost function is the negative of the log-likelihood
similar to the binary case

$$J(\{w_c\}) = - \left(\sum_{n=1}^N y^{(n)^\top} z^{(n)} - \log \sum_{c'} e^{z_{c'}^{(n)}} \right) \text{ where } z_c = w_{[c]}^\top x$$



naive implementation of log-sum-exp causes over/underflow
prevent this using the following trick:

$$\log \sum_c e^{z_c} = \bar{z} + \log \sum_c e^{z_c - \bar{z}}$$

$$\bar{z} \leftarrow \max_c z_c$$


```
1 def cost(X, # NxD design matrix
2         y, # N labels in {1,...,C}
3         W # C x D: one weight vector per class
4         ):
5
6     Z = np.dot(X, W.T) # N x C
7     Y = onehot(y) # N x C
8     nll = - np.sum( np.sum(Z * Y, 1) - logsumexp(Z) )
9     return nll
```



```
1 def logsumexp(
2     Z# C x N
3     ):
4     Zmax = np.max(Z, axis=0)[None,:]
5     lse = Zmax + np.log(np.sum(np.exp(Z - Zmax), axis=0))
6     return lse #N
```

Optimization

given the training data $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_n$

find the best model parameters $\{w_{[c]}\}_c$

by minimizing the cost (maximizing the likelihood of \mathcal{D})

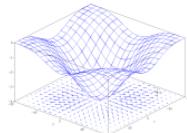
$$J(\{w_c\}) = - \sum_{n=1}^N y^{(n)^\top} z^{(n)} + \log \sum_{c'} e^{z_{c'}^{(n)}} \quad \text{where } z_c = w_{[c]}^\top x$$

need to use gradient descent (for now calculate the gradient)

$$\nabla J(w) = [\frac{\partial}{\partial w_{[1],1}} J, \dots, \frac{\partial}{\partial w_{[1],D}} J, \dots, \frac{\partial}{\partial w_{[C],D}} J]^\top$$

length $C \times D$

Gradient



need to use gradient descent (for now calculate the gradient)

$$J(\{w_c\}) = - \sum_{n=1}^N y^{(n)^\top} z^{(n)} + \log \sum_{c'} e^{z_{c'}^{(n)}} \quad \text{where} \quad z_c = w_{[c]}^\top x$$

using chain rule

$$\frac{\partial}{\partial w_{[c],d}} J = \sum_{n=1}^N \frac{\partial J}{\partial z_c^{(n)}} \frac{\partial z_c^{(n)}}{\partial w_{[c],d}} = \sum_n (\hat{y}_c^{(n)} - y_c^{(n)}) x_d^{(n)}$$

this looks familiar!

$$\begin{array}{c} \downarrow \\ \downarrow \\ x_d^{(n)} \end{array}$$
$$-y_c^{(n)} + \frac{e^{z_c^{(n)}}}{\sum_{c'} e^{z_{c'}^{(n)}}}$$

\downarrow so the derivative of log-sum-exp is softmax

$$\hat{y}^{(n)}$$

Summary

- logistic regression: logistic activation function + cross-entropy loss
 - cost function
 - probabilistic interpretation
 - using maximum likelihood to derive the cost function

$$\begin{array}{ccc} \text{Gaussian likelihood} & \leftrightarrow & \text{L2 loss} \\ \text{Bernoulli likelihood} & & \text{cross-entropy loss} \end{array}$$

- multi-class classification: softmax + cross-entropy
 - cost function
 - one-hot encoding
 - gradient calculation (will use later!)