Logistic regression, cross-entropy loss, gradient descent

Gustave Cortal

Summary

Last course's reminder

Supervised machine learning

Input

a document d
a fixed set of classes $C = c_1, c_2, ..., c_J$
a training set of m hand-labeled documents $(d_1, c_1), ..., (d_m, c_m)$

Output

a learned classifier $\gamma : d \rightarrow c$

Some methods

Naïve Bayes
Logistic Regression
Support-Vector Machines
k-Nearest Neighbors

Bayes' rule applied to documents

For a document d and a class c:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

$P(d|c)$ is the *likelihood*
$P(c)$ is the *prior*
We drop the denominator $P(d)$

The classifier selects the most likely class:

$$c_{\text{max}} = \arg\max_{c \in C} P(c|d)$$

Logistic regression

Generative and discriminative classifiers

Generative classifier

The classifier learns **how the data was generated**
For a document d and a class c:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

We learn the likelihood and the prior: $P(d|c)$ and $P(c)$

$$\hat{c} = \arg\max_{c \in C} P(d|c)P(c)$$

Discriminative classifier

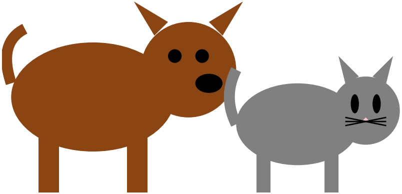
The classifier **directly learns the decision boundary between classes**
We learn the posterior $P(c|d)$ directly

$$\hat{c} = \arg\max_{c \in C} P(c|d)$$

Generative classifier

Suppose we want to predict whether an image corresponds to a cat or a dog

A generative cat model learns the cat characteristics
A generative dog model learns the dog characteristics
Characteristics can be shapes, colors, eyes, etc.

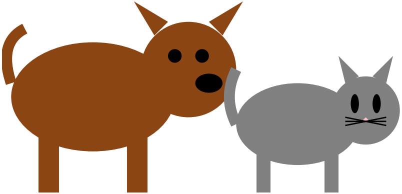


Given a new image, we run both models and see **which one assigns a greater probability of generating this image**

Discriminative classifier

Suppose we want to predict whether an image corresponds to a cat or a dog

A discriminative model learns to distinguish dogs from cats **directly**
For example, a dog has no mustache compared to a cat



Given a new image, we use the *decision boundary* of the discriminative model to determine whether it is a cat or a dog

Components of a machine learning classifier

Given m input and output pairs (x^i, y^i):

- A **feature representation** of the input (eg, *Bag-of-Words*). For each input observation x^i, a vector of features $[x_1, x_2, \ldots, x_n]$. Feature j for input x^i is x_j^i
- A **classification function** that computes $\hat{y}$, the estimated class (eg, *logistic regression*)
- An **objective function** for learning (eg, *cross-entropy loss*)
- An **optimization algorithm** for minimizing or maximizing the objective function (eg, *stochastic gradient descent*)

Weights

Input: $x = [x_1, x_2, \ldots, x_n]$
Weights: $w = [w_1, w_2, \ldots, w_n]$
Output: a predicted class $\hat{y} \in \{0, 1\}$

How to learn a classification function that takes input and weight vectors and outputs the predicted class?

Probabilistic classifier

We want a **probabilistic** classifier:

How to determine $P(y = 1|x; w)$ and $P(y = 0|x; w)$ such that:

$$P(y = 1|x; w) \in [0, 1]$$
$$P(y = 0|x; w) \in [0, 1]$$
$$P(y = 1|x; w) + P(y = 0|x; w) = 1$$

Probabilistic classifier

Let's start with a score z:

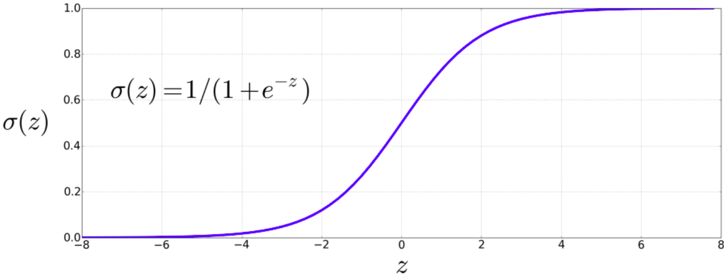
$$z = w \cdot x + b$$

w, x and b are real values vectors, therefore z is a real value

As we want a probability distribution over all possible classes, we need to turn the score into a probability

Sigmoid function

The *sigmoid function* takes a real value as input and outputs a value between 0 and 1



Making probabilities

$$P(y=1) = \sigma(w \cdot x + b)$$

$$= \frac{1}{1+\exp(-(w \cdot x + b))}$$

$$P(y=0) = 1 - \sigma(w \cdot x + b)$$

$$= 1 - \frac{1}{1+\exp(-(w \cdot x + b))}$$

$$= \frac{\exp(-(w \cdot x + b))}{1+\exp(-(w \cdot x + b))}$$

Decision boundary

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

The *decision boundary* gives the final classification

Example

Example (1)

Is this review: *This was an excellent movie. Excellent plot and amazing story - loved it!*, positive or negative?

Let's have a Bag-of-Words representation of the review:

$$x = [excellent, terrible, boring, amazing, loved]$$
$$x = [2, 0, 0, 1, 1]$$

After training, we might get the following weights:

$$w = [0.8, -0.9, -0.7, 0.6, 0.7]$$
$$b = -0.1$$

Example (1)

We compute the score z:

$$\begin{aligned} z &= w \cdot x + b \\ &= 2(0.8) + 0(-0.9) + 0(-0.7) + 1(0.6) + 1(0.7) - 0.1 \\ &= 1.6 + 0 + 0 + 0.6 + 0.7 - 0.1 \\ &= 2.8 \end{aligned}$$

We apply the sigmoid function σ:

$$\begin{aligned} P(\text{positive}) &= \sigma(z) \\ &= \frac{1}{1 + e^{-2.8}} \\ &= 0.94 \end{aligned}$$

Since $P(\text{positive}) = 0.94 > 0.5$, the review is positive

Example (2)

Is this review: *This movie was terrible. So boring and a waste of time!*, positive or negative?

Let's have a Bag-of-Words representation of the review:

$$x = [\mathit{excellent}, \mathit{terrible}, \mathit{boring}, \mathit{amazing}, \mathit{loved}]$$
$$x = [0, 1, 1, 0, 0]$$

After training, we might get the following weights:

$$w = [0.8, -0.9, -0.7, 0.6, 0.7]$$
$$b = -0.1$$

Example (2)

We compute the score z:

$$
\begin{aligned}
z &= w \cdot x + b \\
&= 0(0.8) + 1(-0.9) + 1(-0.7) + 0(0.6) + 0(0.7) - 0.1 \\
&= 0 - 0.9 - 0.7 + 0 + 0 - 0.1 \\
&= -1.7
\end{aligned}
$$

We apply the sigmoid function σ:

$$
\begin{aligned}
P(\text{positive}) &= \sigma(z) \\
&= \frac{1}{1 + e^{1.7}} \\
&= 0.15
\end{aligned}
$$

Since $P(\text{positive}) = 0.15 < 0.5$, the review is negative

Objective function: cross-entropy loss

Loss function and optimization algorithm

To train our logistic regression model, we need to:

- Measure how good our predictions $\hat{y}$ are compared to the true y using a *loss function* (sometimes called a cost function)
- Find the optimal weights w and bias b to minimize the loss using an *optimization algorithm* (eg, gradient descent)

Deriving cross-entropy loss

There are two discrete outcomes (0 or 1)

When $y = 1$, we want $\hat{y} = 1$
When $y = 0$, we want $1 - \hat{y} = 1$

Our goal is to maximize $\hat{y}^y (1 - \hat{y})^{(1-y)}$

Deriving cross-entropy loss

Apply log to avoid numerical instabilities
Apply negative to turn the maximization problem into a minimization one

$$\hat{y}^y (1 - \hat{y})^{(1-y)}$$
$$y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$
$$-y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Cross-entropy loss

For a single training example (x, y):

$$L(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

where:

- y is the true label (0 or 1)
- $\hat{y} = \sigma(w \cdot x + b)$ is our prediction

Cross-entropy loss

When $y = 1$:

$$L(y, \hat{y}) = -\log(\hat{y})$$

When $y = 0$:

$$L(y, \hat{y}) = -\log(1 - \hat{y})$$

The loss increases as our prediction $\hat{y}$ gets further from the true label y

Total loss

For all m training examples:

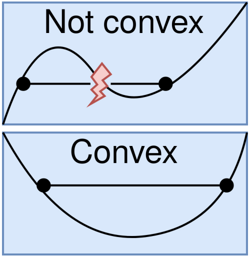
$$L(y, \hat{y}) = -\frac{1}{m} \sum_{i=1}^{m} [y^i \log(\hat{y}^i) + (1 - y^i) \log(1 - \hat{y}^i)]$$

Our goal is to minimize this loss using an optimization algorithm:

$$w^*, b^* = \arg\min_{w,b} \; L(y, \hat{y})$$

Properties of cross-entropy loss

- Always non-negative
- Equals 0 only when predictions exactly match true labels
- For logistic regression, the loss is convex (garanteed to find the global minimum)
- For neural networks, the loss is non-convex (**not** garanteed to find the global minimum)



Optimization algorithm: gradient descent

What are gradients?

A *gradient* is a vector of partial derivatives that points in the direction of steepest increase

For a function $L(x_1, x_2)$:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \frac{\partial L}{\partial x_2} \end{bmatrix}$$

$\frac{\partial L}{\partial x_1}$ indicates how much a small change in x_1 influence the loss L

The negative gradient $-\nabla L$ points in the direction of steepest decrease

Minimizing the loss

To find the optimal weights and bias:

- Compute the gradients $\nabla_\theta L$
- Use gradient descent to update parameters:

$$\theta = \theta - \alpha \nabla_\theta L$$

- Repeat until convergence

where $\theta = (w, b)$ and α is the *learning rate*

The learning rate is a *hyperparameter*

A small learning rate leads to a slow convergence
A high learning rate leads to divergent behaviors

Gradient calculations

Remember:

$$L = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$
$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$
$$z = w \cdot x + b$$

Using the chain rule:

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_j}$$

$$= (\frac{-y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}) \cdot \hat{y}(1 - \hat{y}) \cdot x_j$$

$$= (\hat{y} - y)x_j$$

Types of gradient descent

- Batch gradient descent: uses all training examples for each update, more stable but slower
- Stochastic gradient descent: uses one random example for each update, faster but more noisy
- Mini-batch gradient descent: uses a small random batch of examples, best of both worlds

Regularization

Why regularization?

- Models with many features can *overfit* the training data
- Overfitting: model performs well on training data but poorly on new data
- Signs of overfitting: large weights values, complex decision boundaries, perfect training accuracy but poor test accuracy
- Solution: penalizing large weights using a regularization term in the loss function

Types of regularization

Two common types, lasso ($L1$) and ridge ($L2$) regressions:

Regularization hyperparameter λ controls the strength of regularization

$$L_{L2} = L_{original} + \lambda \sum_{j=1}^{n} w_j^2$$

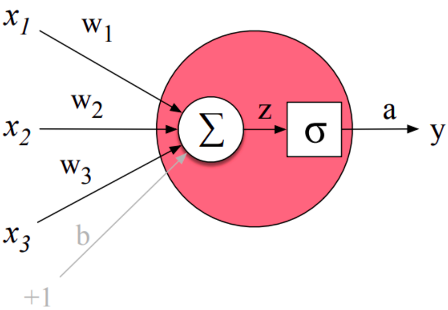
$L2$ regularization drives weights to be small but non-zero

$$L_{L1} = L_{original} + \lambda \sum_{j=1}^{n} |w_j|$$

$L1$ regularization can drive weights to zero, leading to sparse solutions

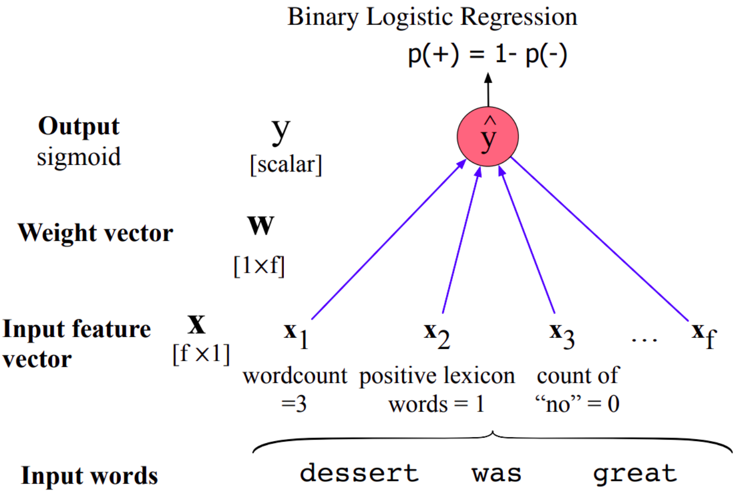
To be continued...

Logistic regression as a neural unit

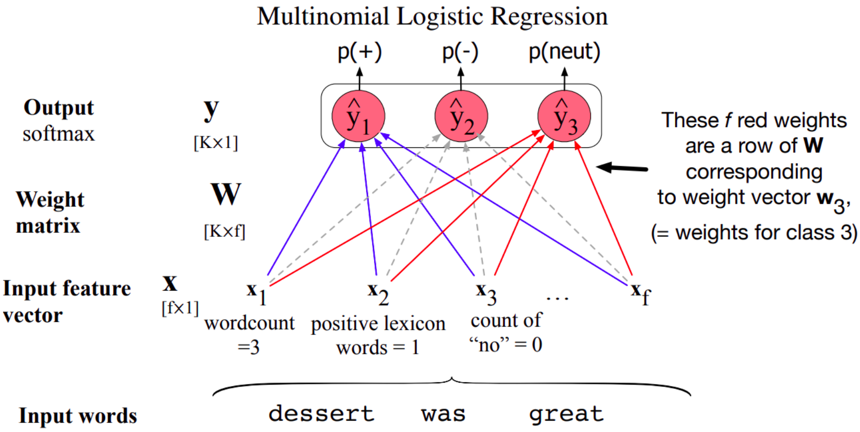


$$y = \sigma(w \cdot x + b) = \frac{1}{1 + \exp(-(w \cdot x + b))}$$

Binary logistic regression



Multinomial logistic regression



Softmax function

The softmax function generalizes the sigmoid to multiple classes:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

For K classes: $\text{z} = [z_1, z_2, ..., z_K]$ becomes probabilities $[p_1, p_2, ..., p_K]$

Multinomial logistic regression

For K classes:

- Each class k has its own weight vector $\mathrm{w_k}$
- Compute K scores: $z_k = \mathrm{w_k} \cdot \mathrm{x} + b_k$
- Apply softmax to get probabilities:

$$P(Y = k|\mathrm{x}) = \frac{e^{\mathrm{w_k} \cdot \mathrm{x} + b_k}}{\sum_{j=1}^{K} e^{\mathrm{w_j} \cdot \mathrm{x} + b_j}}$$

Prediction:

$$\hat{y} = \arg\max_k P(Y = k|\mathrm{x})$$

Cross-entropy loss:

$$L = -\sum_{k=1}^{K} y_k \log(p_k)$$

where y_k is 1 if k is the true class, 0 otherwise

Relationship between cross-entropy and KL divergence

Cross-entropy and *Kullback–Leibler* (KL) divergence are closely related measures used to compare two probability distributions—usually a predicted distribution p and a true distribution q

Cross-entropy $H(q, p)$:

$$H(q, p) = -\sum_x q(x) \log p(x)$$

KL divergence $D_{\mathrm{KL}}(q\|p)$:

$$D_{\mathrm{KL}}(q\|p) = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

$H(q, p) = H(q) + D_{\mathrm{KL}}(q\|p)$

$$H(q) = -\sum_x q(x) \log q(x)$$

Relationship between cross-entropy and KL divergence

Cross-entropy can be decomposed into:

$H(q)$: the entropy of the true distribution (intrinsic uncertainty)

$D_{\mathrm{KL}}(q \| p)$: how much extra uncertainty is introduced by using p instead of q

Minimizing cross-entropy $\implies$ Minimizing $D_{\mathrm{KL}}(q \| p)$, since $H(q)$ is constant

When $p = q$, $D_{\mathrm{KL}}(q \| p) = 0$, and $H(q, p) = H(q)$

Exercices

- Continue implementing naive Bayes classifier from scratch
- Project: discuss possible datasets (final day), write datasheet, perform exploratory data analysis, apply n-grams, naive Bayes and logistic regression on the project dataset