CS 232: Artificial Intelligence

Spring 2024

Prof. Carolyn Anderson
Wellesley College

Reminders

- Please install Tensorflow ASAP so I can help you with any issues!
- My help hours: Friday 3:30-4:30
- Midterm grades coming soon (will also do a general progress update email next week with count of late days used)
- HW 5 released - due Monday
- HW 6 won't be released until after Spring Break

Recap:
Logistic Regression Classifiers

How to do classification

For each feature x_i, weight w_i tells us importance of x_i

- (Plus we'll have a bias b)

We'll sum up all the weighted features and the bias

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

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

POS

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

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

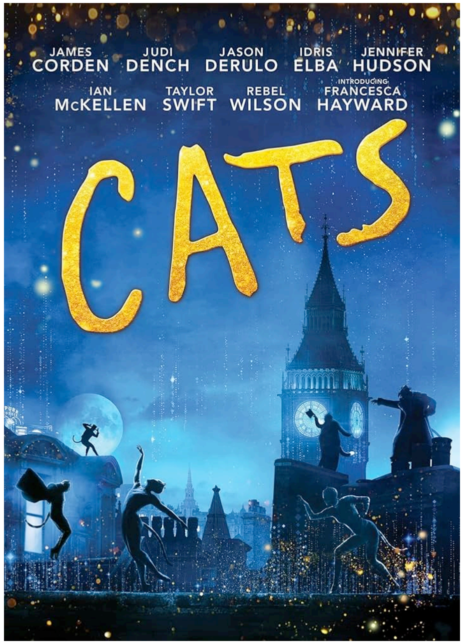
NEG

If this sum is high, we say y=1
If low, then y=0

Sentiment example: does y=1 or y=0?


Verified Jan 12, 2020

CATS was a marvelous disaster, with witty charm and emotion throughout, cheeky charisma and crying no doubt… I personally went in expecting the worst movie I had ever seen - and it was far more awful and disappointing that I expected.



Classifying sentiment for input x

Weights: [2.5,-5,0.5,2,0.7]

Bias: 0.1

count pos | presence of "no" | length of review in log scale

Features for CATS review: [6,4,1,0,3.74]

count neg | presence of "!"

$$P(y=1) = \sigma(\overbrace{w \cdot x + b}^{z})$$
$$= \sigma(-1.78)$$
$$= 0.14$$

$$z = wx + b$$
$$= (\overbrace{[2.5,-5,0.5,2,0.7]}^{w} * \overbrace{[6,4,1,0,3.74]}^{x}) + \overbrace{0.1}^{b}$$
$$= -1.78$$

$$P(y=0) = 1 - \sigma(w \cdot x + b)$$
$$= 1 - 0.14$$
$$= 0.86$$

The two phases of logistic regression

Supervised Machine Learning

Optimization

Training: we learn weights *w* and *b* using **stochastic gradient descent** and **cross-entropy loss**.

Objective Function

Test: Given a test example *x* we compute *p*(*y*|*x*) using learned weights *w* and *b*, and return whichever label (*y* = 1 or *y* = 0) is higher probability

Logistic Regression Example: Pet Picture Classification

Goal: Classify Pet Pictures

- Dataset: cat + dog pictures
- Goal: classify a picture as either a cat or a dog
- Input: grayscale images

Building a Model

We'll build our model using a machine learning library called **Tensorflow**.

Tensorflow is a Python library, but most functions are implemented in C (so they are fast!).

Tensorflow provides useful abstractions for models:

- **tensor**: n-dimensional container for data
- **layer**: apply functions to an input tensor of n dimensions to produce an output tensor of m dimensions.
- **model**: consist of layers connected together

Creating Our Model Architecture

```
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    x = layers.Flatten()(inputs)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


model = make_model(input_shape=image_size, num_classes=2)
keras.utils.plot_model(model, show_shapes=True)
```

input layer

flatten to a single dimension

select sigmoid or softmax
based on number of classes

weights + bias layer -
this is the regression bit!

Learning in Supervised Classification

Supervised classification: (in training!)

- 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.

Learning components

A loss function:

- **cross-entropy loss**

An optimization algorithm:

- **stochastic gradient descent**

The distance between $\hat{y}$ and y

We want to know how far is the classifier output:

$$\hat{y} = \sigma(wx+b)$$

from the true output: $y \in \{0,1\}$

We'll call this difference: $L(\hat{y}, y)$: loss

how much $\hat{y}$
is different than y

Intuition of negative log likelihood loss = cross-entropy loss

A case of conditional maximum likelihood estimation

We choose the parameters *w*,*b* that maximize

- the log probability ???
- of the true *y* labels in the training data
- given the observations *x*

Deriving cross-entropy loss for a single observation x

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

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

Since there are only 2 discrete outcomes (0 or 1) we can express the probability $p(y|x)$ from our classifier as:

If $y=1$:

$$p(y|x) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$$
$$= \hat{y}^{1}(1-\hat{y})^{(1-1)}$$
$$= \hat{y}(1-\hat{y})^{0}$$
$$= \hat{y} \cdot 1$$
$$= 0.86$$

If $y=0$

$$p(y|x) = \hat{y}^{y}(1-\hat{y})^{(1-y)}$$
$$= \hat{y}^{0}(1-\hat{y})^{(1)}$$
$$= 1-\hat{y}$$
$$= 1-0.86$$
$$= 0.14$$

Deriving cross-entropy loss for a single observation x

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

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

$p(y|x;w) =$

Find w such that we maximize $p(y|x;w_0)$

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

$$= \underbrace{y\log\hat{y}}_{\text{if } y=0 \text{ this disappears}} + \underbrace{(1-y)\log(1-\hat{y})}_{\text{if } y=1\text{, this disappears}}$$

Whatever maximizes $\log p(y|x)$ will also maximize $p(y|x)$

Deriving cross-entropy loss for a single observation x

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

Minimize the cross-entropy loss

Goal: make $LCE(\hat{y}, y)$ very small

Minimize: $L_{CE}(\hat{y}, y) = -\log p(y|x)$

$$= -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

$$= -[y \cdot \log \sigma(wx+b) + (1-y)\log(1-\sigma(wx+b))]$$

$-$: to minimize mistakes

y: 1 if correct label is 1

$\log \sigma(wx+b)$: probability assigned to label 1 for x by weights w, b

$(1-y)$: active if actual label is 0

$\log(1-\sigma(wx+b))$: probability assigned to label 0 for x by weights w, b

Does this work for our sentiment example?

We want loss to be:

- smaller if the model estimate is close to correct
- bigger if model is confused

0 negative

Let's first suppose the true label of this is y=~~1~~ (~~positive~~)

CATS was a marvelous disaster, with witty charm and emotion throughout, cheeky charisma and crying no doubt... I personally went in expecting the worst movie I had ever seen - and it was far more awful and disappointing that I expected.

Let's see if this works for our sentiment example

True value is y=0. How well is our model doing?

$$p(+|x) = P(y=1|x) \; = \; \underbrace{0.14}_{\hat{y}}$$

$y = 0$

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

$$= -[\cancel{0 \cdot \log \sigma(w x + b)} + (1-0) \log(1 - \underbrace{\sigma(w x + b)}_{\hat{y}})]$$

$$= -\log(1 - 0.14)$$

$$= -\log(0.86)$$

$$= 0.15$$

What if the true label was 1?

$$p(+|x) = P(y=1|x) \ = 0.14$$

$$L_{\text{CE}}(\hat{y}, y) = -[\underbrace{y}_{} \log \sigma(wx+b) + \underbrace{(1-y)}_{} \log(1-\sigma(wx+b))]$$

$$= -[\log \sigma(wx+b)]$$

$$= -\log(0.14)$$

$$= 1.97$$

Let's see if this works for our sentiment example

The loss when model was right (if true y=0)

$$L_{CE}(0.14, 0) = 0.15$$

Is lower than the loss when model was wrong (if true y=1):

$$L_{CE}(0.14, 1) = 1.97$$

Sure enough, loss was bigger when model was wrong!

Learning components

A loss function:

- **cross-entropy loss**

An optimization algorithm:

- **stochastic gradient descent**

Computing with Probabilities

Numerical Underflow

So far we've been working with relatively small sample spaces. This means our probabilities have been decently large.

As we go on in this class, our sample spaces are going to get much larger. We want to be able to reason about the probabilities of things like:

- All words in English
- All pixels in a photo
- All possible game states for Pacman

Numerical Underflow

Problem: when our probabilities get really really small, programming languages start making mistakes.

There is a **bound on precision** in numerical computing.

This is because of the limitations on space allocation for (floating point) numbers.

Solution: make the numbers bigger

- Intuition: we care about how big probabilities are relative to the other probabilities in our distribution, not the actual value.

<u>Probabilities:</u>
p(heart) = 0.1
p(rainbow) = 0.2
p(letter) = 0.7

Interpretation: a letter is **7 times more likely** than a heart!

Solution: make the numbers bigger

- Intuition: we care about how big probabilities are relative to the other probabilities in our distribution, not the actual value.

<u>Probabilities:</u>

p(heart) = ~~0.1~~ 100

p(rainbow) = ~~0.2~~ 200

p(letter) = ~~0.7~~ 700

What if we just multiply all our probs by 100?

This preserves the ratio.

Solution: make the numbers bigger

- What if we just multiply all our probs by 100? This preserves the ratio.

<u>Probabilities:</u>
p(heart) = ~~0.1~~ 100
p(rainbow) = ~~0.2~~ 200
p(letter) = ~~0.7~~ 700

However, if we want to recover the probabilities later, we'll need to **renormalize** them. This means **remembering that we multiplied by 100**.

Solution: log-transform the numbers

- Instead, we use a log transformation. This changes the range from [0,1] to [-∞, 0].

math.log( )   math.exp( )

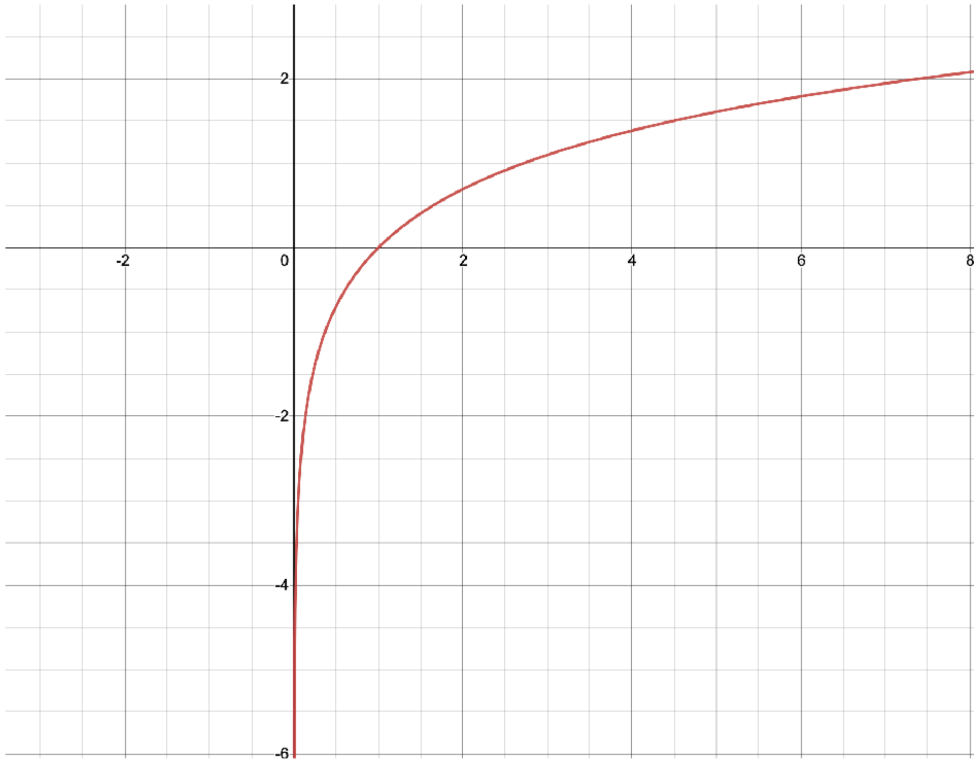
Log base doesn't matter much but we usually use natural log (base e):

Probabilities:

p(heart) = ~~0.1~~ -2.3

p(rainbow) = ~~0.2~~ -1.6

p(letter) = ~~0.7~~ -0.36



www.desmos.com/calculator/aczt76asao