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

source: 2017 Kaggle survey
Classification problem

dataset of inputs $x^{(n)} \in \mathbb{R}^D$
and discrete targets $y^{(n)} \in \{0, \ldots, C\}$
binary classification $y^{(n)} \in \{0, 1\}$

**Linear classification:**
decision boundaries are linear

linear decision boundary $w^T x + b$

how do we find these boundaries?
different approaches give different linear classifiers
Using linear regression

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

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

decision boundary between any two classes $w_c^T x = w_{c'}^T x$

first idea

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

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

\[ a^\top x - b^\top x = 0 \]
\[ (a - b)^\top x = 0 \]

\[ w^\top x = 0 \]
so one weight vector is enough

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

First idea: Using linear regression

Decision boundary is at \( w^\top x = 0 \)
Using linear regression

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

- correctly classified \( w^\top x^{(n)} = 100 > 0 \)
  \[ (100 - 1)^2 = 99^2 \]
- incorrectly classified \( w^\top x^{(n')} = -2 < 0 \)
  \[ (-2 - 1)^2 = 9 \]

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

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

the decision boundary is

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

still a linear decision boundary
Logistic regression: model

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

logistic function
squashing function
activation function

note the linear decision boundary
Logistic regression: the loss

**first idea** use the misclassification error

\[ L_{0/1}(\hat{y}, y) = I(y \neq \text{sign}(\hat{y} - \frac{1}{2})) \]

- not a continuous function (in \( w \))
- hard to optimize
Logistic regression: the loss

**second idea** use the L2 loss

\[
L_2(\hat{y}, y) = \frac{1}{2} (y - \hat{y})^2
\]

- 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

\[ L_{CE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \]

- it is convex in \( w \)
- probabilistic interpretation (soon!)
**Cost function**

we need to optimize the cost wrt. parameters

**first:** simplify

\[ J(w) = \sum_{n=1}^{N} -y^{(n)} \log(\sigma(w^\top x^{(n)})) - (1 - y^{(n)}) \log(1 - \sigma(w^\top x^{(n)})) \]

substitute logistic function

\[ \log \left( \frac{1}{1 + e^{-w^\top x}} \right) = - \log (1 + e^{-w^\top x}) \]

substitute logistic function

\[ \log \left( 1 - \frac{1}{1 + e^{-w^\top x}} \right) = \log \left( \frac{1}{1 + e^{w^\top x}} \right) = - \log (1 + e^{w^\top x}) \]

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

```python
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
```

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

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

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

\[ w_0 = [0, 0] \]

\[ w^* \]

\[ \sigma(w_0^* + w_1^* x) \]
Gradient

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

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

taking partial derivative \( \frac{\partial}{\partial w_d} J(w) = \sum_{n} -y^{(n)} x_d^{(n)} \frac{e^{-w^\top x^{(n)}}}{1 + e^{-w^\top x^{(n)}}} + x_d^{(n)} (1 - y^{(n)}) \frac{e^{w^\top x^{(n)}}}{1 + e^{w^\top x^{(n)}}} \\ = \sum_{n} -x_d^{(n)} y^{(n)} (1 - \hat{y}^{(n)}) + x_d^{(n)} (1 - y^{(n)}) \hat{y}^{(n)} = x_d^{(n)} (\hat{y}^{(n)} - y^{(n)}) \)

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

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

\[ \hat{y}^{(n)} \text{ is the probability of } y^{(n)} = 1 \]

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

\[ p_w(y \mid x) = \mathcal{N}(y \mid w^\top x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-w^\top x)^2}{2\sigma^2}} \]

Maximum likelihood & linear regression

Squared error loss also has max-likelihood interpretation

\[
p_w(y \mid x) = \mathcal{N}(y \mid w^\top x, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-w^\top x)^2}{2\sigma^2}}
\]

\[
L(w) = \prod_{n=1}^{N} p_w(y^{(n)} \mid x^{(n)})
\]

\[
\ell(w) = \sum_n -\frac{1}{2\sigma^2}(y^{(n)} - w^\top x^{(n)})^2 + \text{constants}
\]

Optimal params:

\[
w^* = \arg \max_w \ell(w) = \arg \min_w \frac{1}{2} \sum_n (y^{(n)} - w^\top x^{(n)})^2
\]

Linear least squares!

Multiclass classification

**binary classification**: Bernoulli likelihood:

\[
\text{Bernoulli}(y \mid \hat{y}) = \hat{y}^y (1 - \hat{y})^{1-y} \quad \text{subject to} \quad \hat{y} \in [0, 1]
\]

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

**C classes**: categorical likelihood

\[
\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_c^{\mathbb{1}(y=c)} \quad \text{subject to} \quad \sum_c \hat{y}_c = 1
\]

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 \( p \in \Delta_c \rightarrow \sum_{c=1}^{C} p_c = 1 \)
  \[
  \hat{y}_c = \text{softmax}(z)_c = \frac{e^{z_c}}{\sum_{c'=1}^{C} e^{z_{c'}}} \quad \text{so} \quad \sum_c \hat{y}_c = 1
  \]

if input values are large, **softmax** becomes similar to **argmax**

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

so similar to logistic this is also a squashing function

```python
def softmax(z # C x ... array):
    z = z - np.max(z, 0)
    yh = np.exp(z)
    yh /= np.sum(yh, 0)
    return yh
```
**Multiclass classification**

**C classes**: categorical likelihood

\[
\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_c^{\mathbb{1}(y=c)}
\]

using softmax to enforce sum-to-one constraint

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

so we have on parameter vector for each class

to simplify equations we write \(z_c = w[c] \top x\)

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

**C classes**: categorical likelihood

\[
\text{Categorical}(y \mid \hat{y}) = \prod_{c=1}^{C} \hat{y}_{c}^{\mathbb{I}(y=c)}
\]

using softmax to enforce sum-to-one constraint

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

where \( z_c = w_{[c]}^\top x \)

substituting softmax in Categorical likelihood:

\[
L(\{w_c\}) = \prod_{n=1}^{N} \prod_{c=1}^{C} \text{softmax}([z^{(n)}_1, \ldots, z^{(n)}_C]_{c=1}^{C} \mathbb{I}(y^{(n)}=c))
\]

\[
= \prod_{n=1}^{N} \prod_{c=1}^{C} \left( \frac{e^{z^{(n)}_c}}{\sum_{c'} e^{z^{(n)}_{c'}}} \mathbb{I}(y^{(n)}=c) \right)
\]
One-hot encoding

**Likelihood**

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

**Log-likelihood**

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

**One-hot encoding** for labels

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

using this encoding from now on

**Log-likelihood**

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

```python
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
```
One-hot encoding

**side note**

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

**one-hot encoding** for input features

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

**problem**

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

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

**solution**

remove one of the one-hot encoding features

\[ x_d^{(n)} \rightarrow [\mathbb{I}(x_d^{(n)} = 1), \ldots, \mathbb{I}(x_d^{(n)} = C - 1)] \]
Implementing the cost function

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

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

where \( z_c = w_c \top x \)

naive implementation of log-sum-exp causes over/underflow

prevent this using the following trick:

\[ \log \sum_c e^{z_c} = \tilde{z} + \log \sum_c e^{z_c - \tilde{z}} \]

\( \tilde{z} \leftarrow \max_c z_c \)

def cost(X, y, W):
    Z = np.dot(X, W) # N x C
    Y = onehot(y) # N x C
    nll = - np.sum(np.sum(Z * Y, 1) - logsumexp(Z))
    return nll

def logsumexp(Z):
    Zmax = np.max(Z, axis=0)[None, :]
    else = Zmax + np.log(np.sum(np.exp(Z - Zmax), axis=0))
    return else
Optimization

given the training data \( D = \{(x^{(n)}, y^{(n)})\}_n \)
find the best model parameters \( \{w[c]\}_c \)
by minimizing the cost (maximizing the likelihood of \( D \))

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

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

\[
\nabla J(w) = \begin{bmatrix} \frac{\partial}{\partial w_1[1]} J, \ldots, \frac{\partial}{\partial w_{1[D]} J}, \ldots, \frac{\partial}{\partial w_{C[D]} J} \end{bmatrix}^\top \\
\text{length } C \times D
\]
Gradient

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

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

using chain rule

\[ \frac{\partial}{\partial w_{[c],d}} J = \sum_{n=1}^{N} \left( \frac{\partial J}{\partial z_c^{(n)}} \frac{\partial z_c^{(n)}}{\partial w_{[c],d}} \right) = \sum_{n} (\hat{y}_{c}^{(n)} - y_{c}^{(n)}) x_{d}^{(n)} \]

\[ \text{this looks familiar!} \]

\[ -y_{c}^{(n)} + \frac{e^{z_{c}^{(n)}}}{\sum_{c'} e^{z^{(n)}_{c'}}} \]

so the derivative of log-sum-exp is softmax
Summary

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

    \[
    \text{Gaussian likelihood} \iff \text{L2 loss} \\
    \text{Bernoulli likelihood} \iff \text{cross-entropy loss}
    \]

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