Logistic Regression

COMP 527

Danushka Bollegala

UNIVERSITY OF LIVERPOOL

Binary Classification

- Given an instance **x** we must classify it to either positive (1) or negative (0) class
  - We can use {1,-1} instead of {1,0} but we will use the latter formulation as it simplifies the notation in subsequent derivations
- Binary classification can be seen as learning a function *f* such that *f*(x) returns either 1 or 0, indicating the predicted class

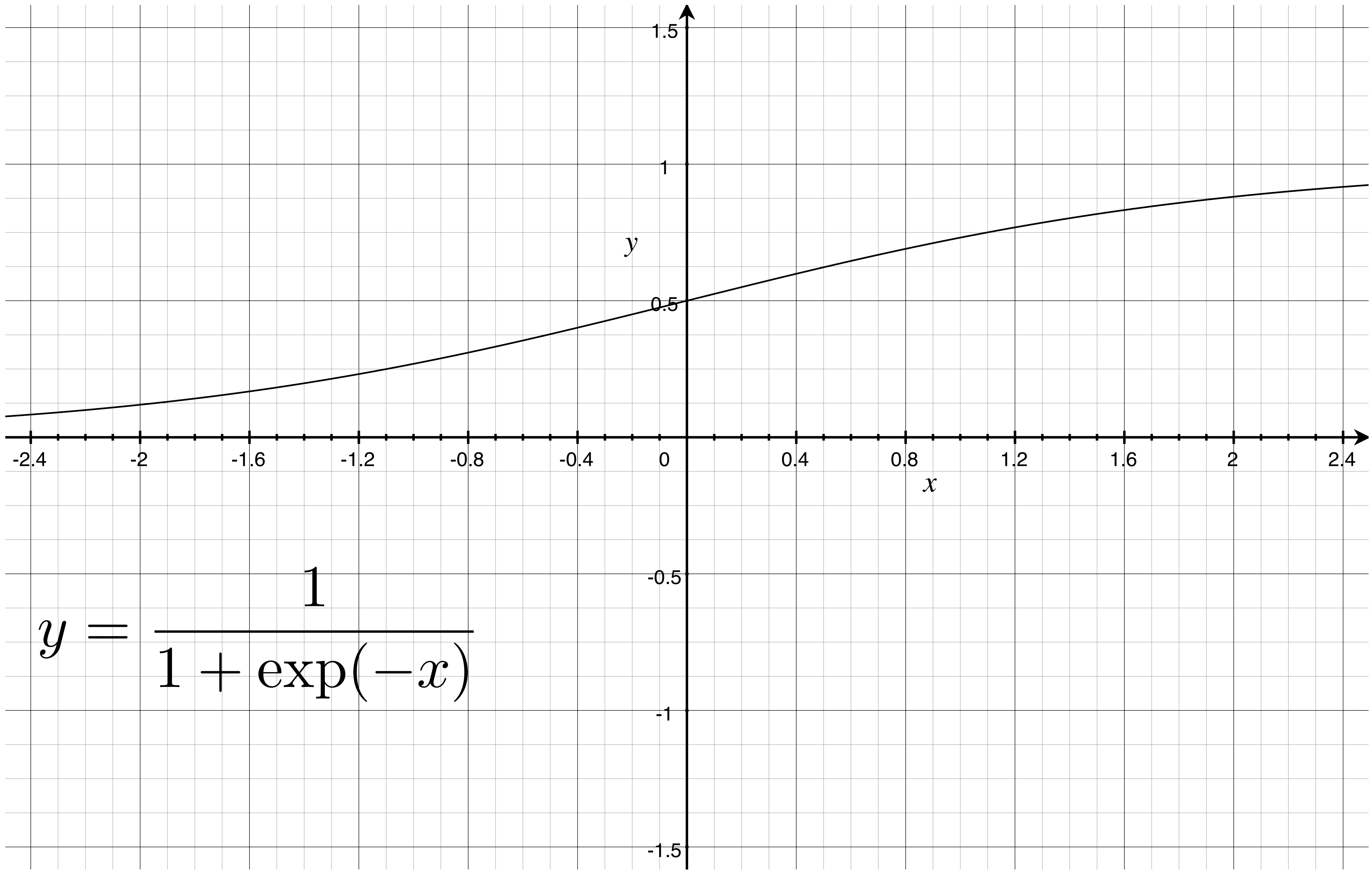
Some terms in Machine Learning

- Training dataset with N instances
  - $\{(x_1,t_1), ..., (x_N,t_N)\}$
- Target label (class)
  - t: The class labels in the training dataset
  - Annotated by humans (supervised learning)
- Predicted label
  - Labels predicted by our model $f(x)$
- P(A|B): conditional probability of observing an event A, given an event B
- P(A): marginal probability of event A
  - We have *marginalised out* all the variables on which A depends upon (cf. margin of a probability table)
- Prior probability P(B)
- Posterior probability P(B|A)

Logistic Regression

- is not a *regression* model
- is a *classification* model
- is the basis of many advanced machine learning methods
    - neural networks, deep learning, conditional random fields, ...
- Try to fit a logistic sigmoid function to predict the class labels

Logistic Sigmoid Function



$$y = \frac{1}{1 + \exp(-x)}$$

Why do we use logistic sigmoid?

- Reason 1:
  - We must squash the prediction score $\mathbf{w}^T\mathbf{x}$, which is in the range $(-\infty,+\infty)$ to the range [0,1] when performing binary classification
- Reason 2: (Bayes' Rule)
  - Posterior $\propto$ Conditional x Prior

$$
\begin{aligned}
P(t=1|x) &= \frac{P(x|t=1)P(t=1)}{P(x)} \\
&= \frac{P(x|t=1)P(t=1)}{P(t=1)P(x|t=1)+P(t=0)P(x|t=0)} \\
&= \frac{1}{1+\frac{1}{\frac{P(x|t=1)P(t=1)}{P(t=0)P(x|t=0)}}}
\end{aligned}
$$

$$
\exp(a) = \frac{P(x|t=1)P(t=1)}{P(t=0)P(x|t=0)}
$$

$$
P(t=1|x) = \frac{1}{1+\exp(-a)} = \sigma(a)
$$

Likelihood

- We have a probabilistic model (logistic sigmoid function $\sigma(\mathbf{w}^T\mathbf{x})$) that tells us the probability of a particular training instance $\mathbf{x}$ being positive ($t=1$) or negative ($t=0$)
- We can use this model to predict the probability of the entire training dataset
  - *likelihood* of the training dataset
- However, this dataset is already *observed* (we have it with us)
- If we want to *explain* this training dataset, then our model must maximise the likelihood for this training dataset (more than any other labelling of the dataset)
- Maximum Likelihood Estimate/Principle (MLE)

Maximum Likelihood Estimate

$$
\begin{aligned}
y_n &= \sigma(\boldsymbol{w}^\top \boldsymbol{x}_n) = \frac{1}{1 + \exp(-\boldsymbol{w}^\top \boldsymbol{x_n})} \\
\boldsymbol{t} &= (t_1, \ldots, t_n)^\top \\
p(\boldsymbol{t}|\boldsymbol{w}) &= \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{(1-t_n)}
\end{aligned}
$$

By taking the negative of the logarithm of the above product we define the cross-entropy error function

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

Q1

By differentiating E(w) w.r.t. w we get ∇E(w) as follows:

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - t_n) x_n$$

Q2

Q1: Derivation of Cross Entropy Error Function

$$E(w) = -\ln p(\mathbf{t}|w) = -\ln \prod_{n=1}^{N} y_n^{t_n} (1-y_n)^{(1-t_n)}$$

$$= -\sum_{n=1}^{N} \ln y_n^{t_n} (1-y_n)^{(1-t_n)}$$

$$= -\sum_{n=1}^{N} \left\{ \ln y_n^{t_n} + \ln (1-y_n)^{(1-t_n)} \right\}$$

$$= -\sum_{n=1}^{N} \left\{ t_n \ln y_n + (1-t_n) \ln (1-y_n) \right\}. \quad // . \quad (Q.E.D)$$

Q2: Derivation of the gradient

$$\nabla \equiv \left(\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \cdots, \frac{\partial}{\partial w_D}\right)^T, \quad \frac{\partial}{\partial x}\ln x = \frac{1}{x}.$$

$$\therefore \nabla E(w) = -\sum_{n=1}^{N}\left\{t_n \frac{1}{y_n}\cdot\frac{\partial y_n}{\partial w} + (1-t_n)\frac{1}{1-y_n}\left(-\frac{\partial y_n}{\partial w}\right)\right\}$$

$$= -\sum_{n=1}^{N}\left\{\frac{t_n}{y_n} - \frac{1-t_n}{1-y_n}\right\}\left(\frac{\partial y_n}{\partial w}\right)$$

$$= -\sum_{n=1}^{N}\left\{\frac{(t_n - y_n)}{y_n(1-y_n)}\frac{\partial y_n}{\partial w}\right\} \quad \text{— (1)}$$

$$y_n = \frac{1}{1+\exp(-w^T x_n)}$$

$$\frac{\partial y_n}{\partial w} = \frac{\partial}{\partial w}\left[1+\exp(-w^T x_n)\right]^{-1}$$

$$= \frac{-1}{(1+\exp(-w^T x_n))^2}\cdot \exp(-w^T x_n)\cdot(-x_n).$$

$$= \underbrace{\frac{1}{1+\exp(-w^T x_n)}}_{y_n} \cdot \underbrace{\frac{\exp(-w^T x_n)}{1+\exp(-w^T x_n)}}_{(1-y_n)} \cdot x_n.$$

$$= y_n(1-y_n)x_n. \quad \text{— (2)}$$

$\therefore$ substituting (2) in (1) we get

$$\nabla E(w) = -\sum_{n=1}^{N}\frac{(t_n - y_n)}{\cancel{y_n(1-y_n)}}\cdot \cancel{y_n(1-y_n)}x_n$$

$$= \sum_{n=1}^{N}(y_n - t_n)x_n. \quad // \quad (Q.E.D).$$

Updating the weight vector

- Generic update rule

$$\boldsymbol{w}^{(r+1)} = \boldsymbol{w}^{(r)} - \eta \nabla E(\boldsymbol{w})$$

- Update rule with cross-entropy error function

$$\boldsymbol{w}^{(r+1)} = \boldsymbol{w}^{(r)} - \eta (y_n - t_n) \boldsymbol{x}_n$$

Logistic Regression Algorithm

- Given a set of training instances $\{(x_1,t_1), \ldots, (x_N,t_N)\}$, learning rate, η, and iterations T
- Initialise weight vector $\mathbf{w} = \mathbf{0}$
- For j in 1,...,T
    - For n in 1,...,N
        - if $\text{pred}(\mathbf{x}_i) \neq t_i$ #misclassification
            - $\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \eta(y_n - t_n)\mathbf{x}_n$
- Return the final weight vector **w**

Prediction Function *pred*

- Given the weight vector **w**, returns the class label for an instance **x**
  - if $\mathbf{w}^T\mathbf{x} > 0$:
    - predicted label = +1 # positive class
  - else:
    - predicted label = 0 # negative class

Online vs. Batch

- Online vs. Batch Logistic Regression
  - The algorithm we discussed in the previous slides is an *online algorithm* because it considers only one instance at a time and updates the weight vector
    - Referred to as the Stochastic Gradient Descent (SGD) update
  - In the batch version, we will compute the cross-entropy error over the *entire* training dataset and then update the weight vector
    - Popular optimisation algorithm for the batch learning of logistic regression is the Limited Memory BFGS (L-BFGS) algorithm
- Batch version is slow compared to the SGD version. But shows slightly improved accuracies in many cases
- SGD version can require multiple iterations over the dataset before it converges (if ever)
- SGD is a technique that is frequently used with large scale machine learning tasks (even when the objective function is non-convex)

Regularisation

- Regularisation
    - Reducing overfitting in a model by constraining it (reducing the complexity/no. of parameters)
    - For classifiers that use a weight vector, regularisation can be done by minimising the norm (length) of the weight vector.
    - Several popular regularisation methods exist
        - L2 regularisation (ridge regression or Tikhonov regularisation)
        - L1 regularisation (Lasso regression)
        - L1+L2 regularisation (mixed regularisation)

L2 regularisation

- Let us denote the Loss of classifying a dataset D using a model represented by a weight vector **w** by L(D,**w**) and we would like to impose L2 regularisation on **w**.
- The overall objective to minimise can then be written as follows (here λ is called the regularisation coefficient and is set via cross-validation)

$$J(D, \boldsymbol{w}) = L(D, \boldsymbol{w}) + \lambda \, ||\boldsymbol{w}||_2^2$$

- The gradient of the overall objective simply becomes the addition of the loss-gradient and the scaled weight vector **w**.

$$\frac{\partial J(D, \boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{\partial L(D, \boldsymbol{w})}{\partial \boldsymbol{w}} + 2\lambda \boldsymbol{w}$$

Examples

- Note that SGD update for minimising a loss multiplies the loss gradient by a negative learning rate (η). Therefore, the L2 regularised update rules will have a -2ηλ**w** term as shown in the following examples

- L2 regularised Perceptron update (for a misclassified instance we do)

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + t\boldsymbol{x} - 2\lambda \boldsymbol{w}^{(k)}$$

- L2 regularised logistic regression

$$\begin{aligned} \boldsymbol{w}^{(k+1)} &= \boldsymbol{w}^{(k)} - \eta((y-t)\boldsymbol{x} + 2\lambda \boldsymbol{w}^{(k)}) \\ &= (1 - 2\lambda\eta)\boldsymbol{w}^{(k)} - \eta(y-t)\boldsymbol{x} \end{aligned}$$

How to set λ

- Split your training dataset into training and validation parts (eg. 80%-20%)

- Try different values for λ (typically in the logarithmic scale). Train a different classification model for each λ and select the value that gives the best performance (eg. accuracy) on the validation data.

  - $\lambda = 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 0, 10^{1}, 10^{2}, 10^{3}, 10^{4}, 10^{5}$

References

- Bishop (Pattern Recognition and Machine Learning) Section 4.3.2
- Software
    - scikit-learn (Python)
        - http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
    - Classias (C)
        - http://www.chokkan.org/software/classias/