

Logistic Regression

COMP 527

Danushka Bollegala



UNIVERSITY OF
LIVERPOOL

Binary Classification

- Given an instance \mathbf{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 $f(\mathbf{x})$ returns either 1 or 0, indicating the predicted class

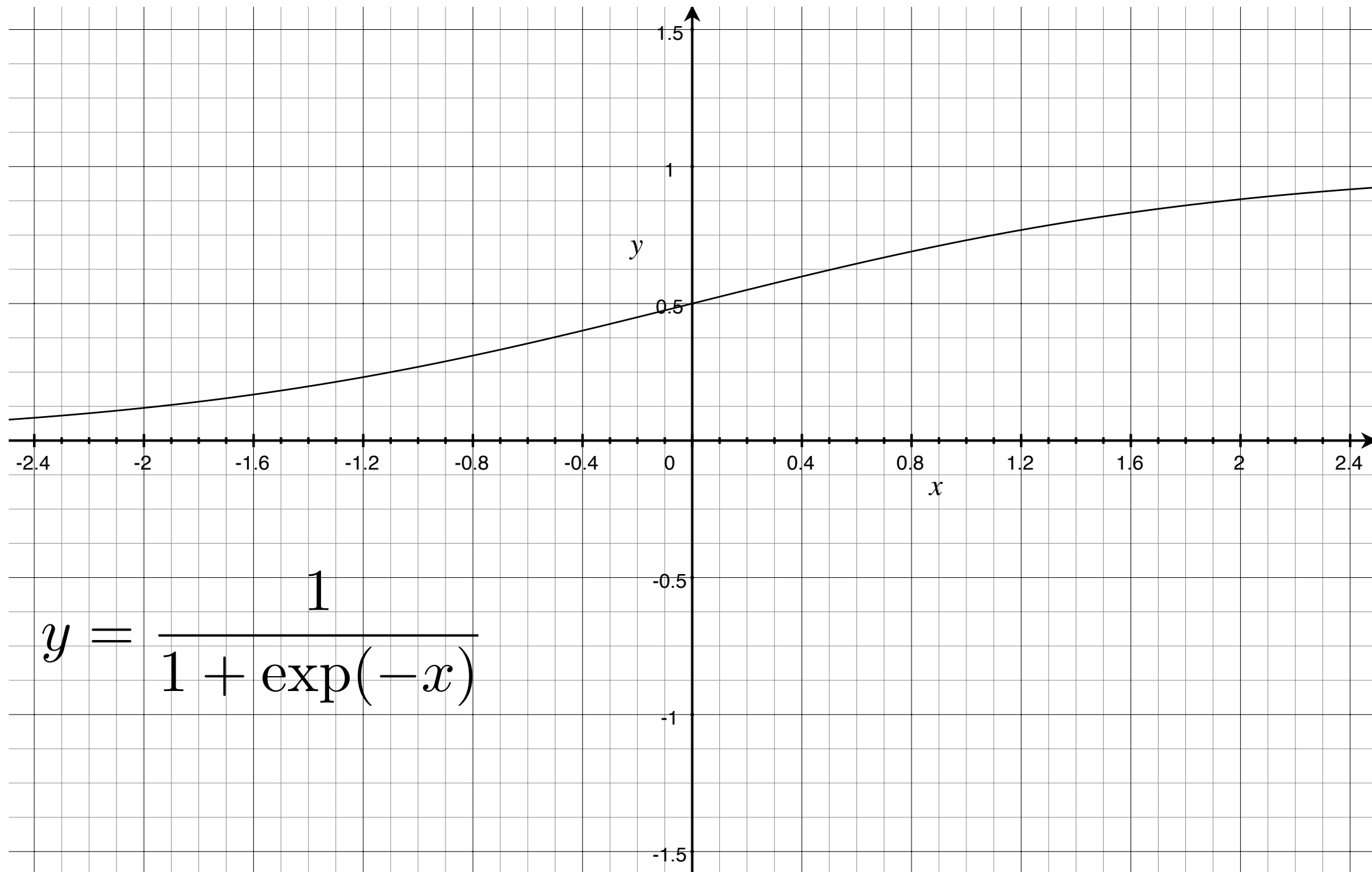
Some terms in Machine Learning

- Training dataset with N instances
 - $\{(x_1, t_1), \dots, (x_N, t_N)\}$
- 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



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

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

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

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

$$y_n = \sigma(\mathbf{w}^\top \mathbf{x}_n) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)}$$

$$\mathbf{t} = (t_1, \dots, t_n)^\top$$

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

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

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^N \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \quad \text{Q1}$$

By differentiating $E(\mathbf{w})$ w.r.t. \mathbf{w} we get $\nabla E(\mathbf{w})$ as follows:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^N (y_n - t_n) \mathbf{x}_n \quad \text{Q2}$$

Q1: Derivation of Cross Entropy Error Function

Q2: Derivation of the gradient

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 $\{(x_1, t_1), \dots, (x_N, t_N)\}$, learning rate, η , and iterations T
- Initialise weight vector $\mathbf{w} = \mathbf{0}$
- For j in $1, \dots, T$
 - For n in $1, \dots, N$
 - 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 \mathbf{w}

Prediction Function *pred*

- Given the weight vector \mathbf{w} , returns the class label for an instance \mathbf{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 \mathbf{w} by $L(D, \mathbf{w})$ and we would like to impose L2 regularisation on \mathbf{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)

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

- The gradient of the overall objective simply becomes the addition of the loss-gradient and the scaled weight vector \mathbf{w} .

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

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 $-2\eta\lambda\mathbf{w}$ term as shown in the following examples
- L2 regularised Perceptron update (for a misclassified instance we do)

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

- L2 regularised logistic regression

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

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/>