IAML: Logistic Regression

Nigel Goddard and Victor Lavrenko
School of Informatics

Semester 1

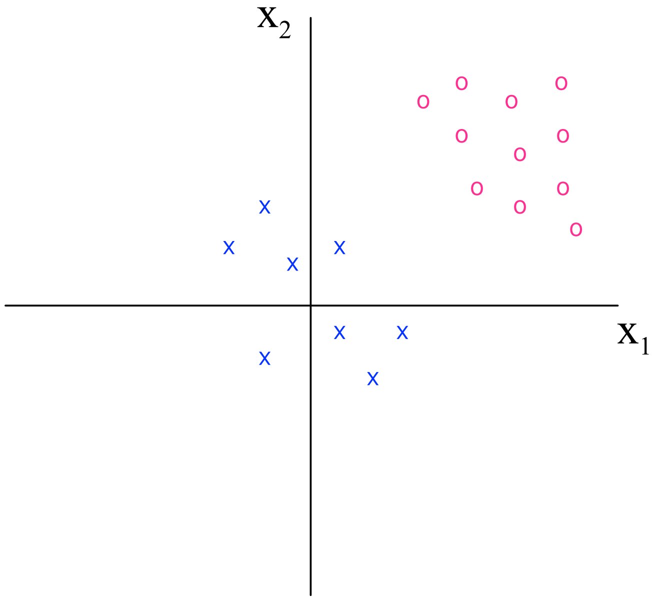
Outline

- Logistic function
- Logistic regression
- Learning logistic regression
- Optimization
- The power of non-linear basis functions
- Least-squares classification
- Generative and discriminative models
- Relationships to Generative Models
- Multiclass classification
- Reading: W & F §4.6 (but pairwise classification, perceptron learning rule, Winnow are not required)

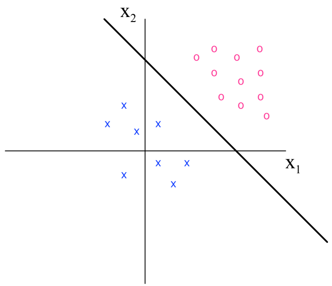
Decision Boundaries

- In this class we will discuss *linear classifiers*.
- For each class, there is a *region* of feature space in which the classifier selects one class over the other.
- The decision boundary is the boundary of this region. (i.e., where the two classes are “tied”)
- In linear classifiers the decision boundary is a line.

Example Data



Linear Classifiers



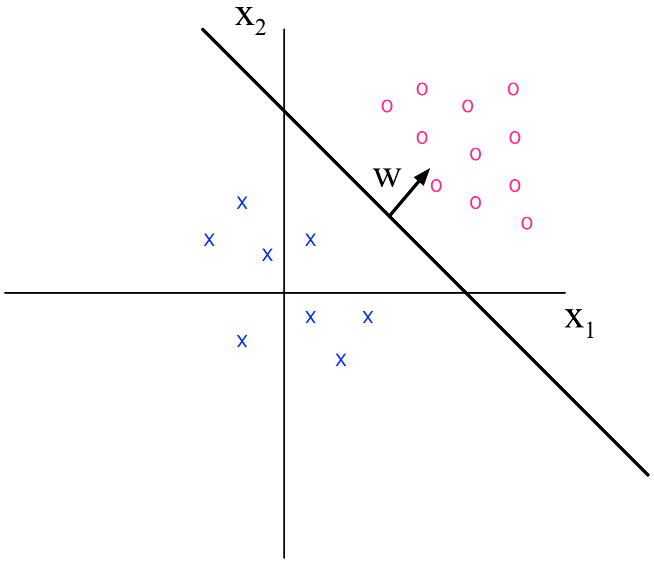
- In a two-class linear classifier, we learn a function

$$F(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \mathbf{x} + w_0$$

  that represents how aligned the instance is with $y = 1$.
- **w** are parameters of the classifier that we learn from data.
- To do prediction of an input **x**:

$$\mathbf{x} \mapsto (y = 1) \quad \text{if } F(\mathbf{x}, \mathbf{w}) > 0$$

A Geometric View



Explanation of Geometric View

- The decision boundary in this case is

$$\{\mathbf{x} | \mathbf{w}^\top \mathbf{x} + w_0 = 0\}$$

- **w** is a normal vector to this surface
- (Remember how lines can be written in terms of their normal vector.)
- Notice that in more than 2 dimensions, this boundary will be a hyperplane.

Two Class Discrimination

- For now consider a two class case: $y \in \{0, 1\}$.
- From now on we'll write $\mathbf{x} = (1, x_1, x_2, \ldots x_d)$ and $\mathbf{w} = (w_0, w_1, \ldots x_d)$.
- We will want a linear, probabilistic model. We could try $P(y = 1|\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$. But this is stupid.
- Instead what we will do is

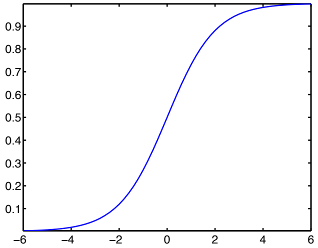
$$P(y = 1|\mathbf{x}) = f(\mathbf{w}^\top \mathbf{x})$$

- f must be between 0 and 1. It will squash the real line into $[0, 1]$
- Furthermore the fact that probabilities sum to one means

$$P(y = 0|\mathbf{x}) = 1 - f(\mathbf{w}^\top \mathbf{x})$$

The logistic function

- We need a function that returns probabilities (i.e. stays between 0 and 1).
- The logistic function provides this
- $f(z) = \sigma(z) \equiv 1/(1 + \exp(-z))$.
- As z goes from $-\infty$ to ∞, so f goes from 0 to 1, a "squashing function"
- It has a "sigmoid" shape (i.e. S-like shape)



Linear weights

- Linear weights + logistic squashing function == logistic regression.
- We model the class probabilities as

$$p(y = 1|\mathbf{x}) = \sigma(\sum_{j=0}^{D} w_j x_j) = \sigma(\mathbf{w}^T \mathbf{x})$$

- $\sigma(z) = 0.5$ when $z = 0$. Hence the decision boundary is given by $\mathbf{w}^T \mathbf{x} = 0$.
- Decision boundary is a $M - 1$ hyperplane for a M dimensional problem.

Logistic regression

- For this slide write $\tilde{\mathbf{w}} = (w_1, w_2, \dots w_d)$ (i.e., exclude the bias w_0)
- The bias parameter w_0 shifts the position of the hyperplane, but does not alter the angle
- The direction of the vector $\tilde{\mathbf{w}}$ affects the angle of the hyperplane. The hyperplane is perpendicular to $\tilde{\mathbf{w}}$
- The magnitude of the vector $\tilde{\mathbf{w}}$ effects how certain the classifications are
- For small $\tilde{\mathbf{w}}$ most of the probabilities within the region of the decision boundary will be near to 0.5.
- For large $\tilde{\mathbf{w}}$ probabilities in the same region will be close to 1 or 0.

Learning Logistic Regression

- Want to set the parameters **w** using training data.
- As before:
  - Write out the model and hence the likelihood
  - Find the derivatives of the log likelihood w.r.t the parameters.
  - Adjust the parameters to maximize the log likelihood.

- Assume data is independent and identically distributed.
- Call the data set $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots (\mathbf{x}_n, y_n)\}$
- The likelihood is

$$\begin{aligned} p(D|\mathbf{w}) &= \prod_{i=1}^{n} p(y = y_i|\mathbf{x}_i, \mathbf{w}) \\ &= \prod_{i=1}^{n} p(y = 1|\mathbf{x}_i, \mathbf{w})^{y_i} \left(1 - p(y = 1|\mathbf{x}_i, \mathbf{w})\right)^{1-y_i} \end{aligned}$$

- Hence the log likelihood $L(\mathbf{w}) = \log p(D|\mathbf{w})$ is given by

$$L(\mathbf{w}) = \sum_{i=1}^{n} y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))$$

- It turns out that the likelihood has a unique optimum (given sufficient training examples). It is *convex*.
- How to maximize? Take gradient

$$\frac{\partial L}{\partial w_j} = \sum_{i=1}^{n} (y_i - \sigma(\mathbf{w}^T \mathbf{x}_i)) x_{ij}$$

- (Aside: something similar holds for linear regression

$$\frac{\partial E}{\partial w_j} = \sum_{i=1}^{n} (\mathbf{w}^T \phi(\mathbf{x}_i) - y_i) x_{ij}$$

  where E is squared error.)
- Unfortunately, you cannot maximize $L(\mathbf{w})$ explicitly as for linear regression. You need to use a numerical method (see next lecture).

Geometric Intuition of Gradient

- Let's say there's only one training point $D = \{(\mathbf{x}_1, y_1)\}$. Then
$$\frac{\partial L}{\partial w_j} = (y_1 - \sigma(\mathbf{w}^\top \mathbf{x}_1))x_{1j}$$
- Also assume $y_1 = 1$. (It will be symmetric for $y_1 = 0$.)
- Note that $(y_1 - \sigma(\mathbf{w}^\top \mathbf{x}_1))$ is always positive because $\sigma(z) < 1$ for all z.
- There are three cases:
  - If $\mathbf{x}_1$ is classified as right answer with high confidence, e.g., $\sigma(\mathbf{w}^\top \mathbf{x}_1) = 0.99$
  - If $\mathbf{x}_1$ is classified wrong, e.g., $(\sigma(\mathbf{w}^\top \mathbf{x}_1) = 0.2)$
  - If $\mathbf{x}_1$ is classified correctly, but just barely, e.g., $\sigma(\mathbf{w}^\top \mathbf{x}_1) = 0.6$.

Geometric Intuition of Gradient

- One training point, $y_1 = 1$.

$$\frac{\partial L}{\partial w_j} = (y_1 - \sigma(\mathbf{w}^\top \mathbf{x}_1))x_{1j}$$

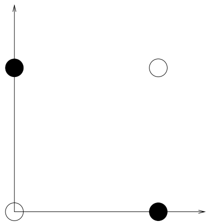
- Remember: gradient is direction of steepest *increase*. We want to maximize, so let's nudge the parameters in the direction $\frac{\partial L}{\partial w_j}$
- If $\sigma(\mathbf{w}^\top \mathbf{x}_1)$ is correct, e.g., 0.99
  - Then $(y_1 - \sigma(\mathbf{w}^\top \mathbf{x}_1))$ is nearly 0, so we don't change w_j.
- If $\sigma(\mathbf{w}^\top \mathbf{x}_1)$ is wrong, e.g., 0.2
  - This means $\mathbf{w}^\top \mathbf{x}_1$ is negative. It should be positive.
  - The gradient has the same sign as x_{1j}
  - If we nudge w_j, then w_j will tend to increase if $x_{1j} > 0$ or decrease if $x_{1j} < 0$.
  - Either way $\mathbf{w}^\top \mathbf{x}_1$ goes up!
- If $\sigma(\mathbf{w}^\top \mathbf{x}_1)$ is just *barely* correct, e.g., 0.6
  - Same thing happens as if we were wrong, just more slowly.

Fitting this into the general structure for learning algorithms:

- Define the **task**: classification, discriminative
- Decide on the **model structure**: logistic regression model
- Decide on the **score function**: log likelihood
- Decide on **optimization/search method** to optimize the score function: numerical optimization routine. Note we have several choices here (stochastic gradient descent, conjugate gradient, BFGS).

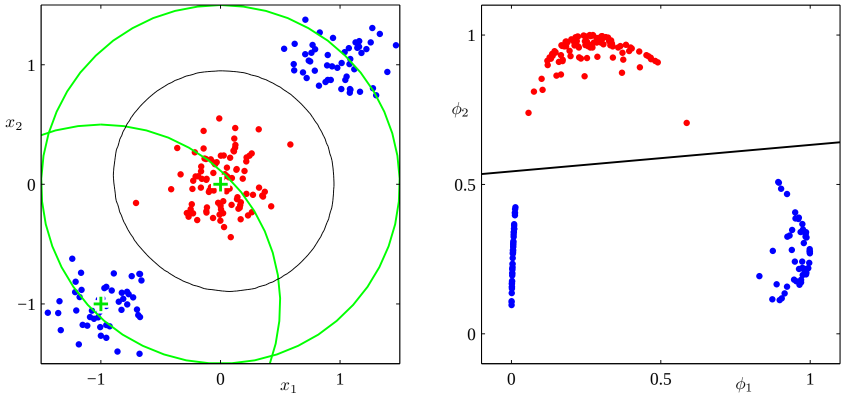
XOR and Linear Separability

- A problem is linearly separable if we can find weights so that
  - $\tilde{\mathbf{w}}^T\mathbf{x} + w_0 > 0$ for all positive cases (where $y = 1$), and
  - $\tilde{\mathbf{w}}^T\mathbf{x} + w_0 \leq 0$ for all negative cases (where $y = 0$)
- XOR, a failure for the perceptron



- XOR can be solved by a perceptron using a nonlinear transformation $\phi(\mathbf{x})$ of the input; exercise - can you find one?

The power of non-linear basis functions



Using two Gaussian basis functions $\phi_1(\mathbf{x})$ and $\phi_2(\mathbf{x})$

Figure credit: Chris Bishop, PRML

As for linear regression, we can transform the input space if we want $\mathbf{x} \rightarrow \phi(\mathbf{x})$

Generative and Discriminative Models

- Notice that we have done something very different here than with naive Bayes.
- Naive Bayes: Modelled how a class "generated" the feature vector $p(\mathbf{x}|y)$. Then could classify using

$$p(y|\mathbf{x}) \propto p(\mathbf{x}|y)p(y)$$

  . This called is a *generative* approach.
- Logistic regression: Model $p(y|\mathbf{x})$ directly. This is a *discriminative* approach.
- Discriminative advantage: Why spend effort modelling $p(\mathbf{x})$? Seems a waste, we're always given it as input.
- Generative advantage: Can be good with missing data (remember how naive Bayes handles missing data). Also good for detecting outliers. Or, sometimes you really do want to generate the input.

Generative Classifiers can be Linear Too

Two scenarios where naive Bayes gives you a linear classifier.

1. *Gaussian data with equal covariance.* If $p(\mathbf{x}|y=1) \sim N(\boldsymbol{\mu}_1, \Sigma)$ and $p(\mathbf{x}|y=0) \sim N(\boldsymbol{\mu}_2, \Sigma)$ then

$$p(y=1|\mathbf{x}) = \sigma(\tilde{\mathbf{w}}^T \mathbf{x} + w_0)$$

   for some $(w_0, \tilde{\mathbf{w}})$ that depends on $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, Σ and the class priors
2. *Binary data.* Let each component x_j be a Bernoulli variable i.e. $x_j \in \{0, 1\}$. Then a Naïve Bayes classifier has the form

$$p(y=1|\mathbf{x}) = \sigma(\tilde{\mathbf{w}}^T \mathbf{x} + w_0)$$

3. Exercise for keeners: prove these two results

Multiclass classification

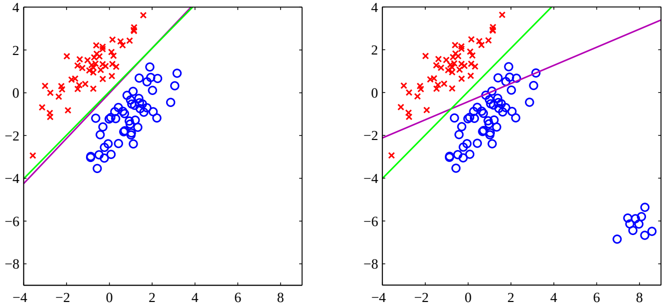
- Create a different weight vector $\mathbf{w}_k$ for each class
- Then use the "softmax" function

$$p(y = k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x})}{\sum_{j=1}^{C} \exp(\mathbf{w}_j^T \mathbf{x})}$$

- Note that $0 \leq p(y = k|\mathbf{x}) \leq 1$ and $\sum_{j=1}^{C} p(y = j|\mathbf{x}) = 1$
- This is the natural generalization of logistic regression to more than 2 classes.

Least-squares classification

- Logistic regression is more complicated algorithmically than linear regression
- Why not just use linear regression with 0/1 targets?



Green: logistic regression; magenta, least-squares regression

Figure credit: Chris Bishop, PRML

Summary

- The logistic function, logistic regression
- Hyperplane decision boundary
- The perceptron, linear separability
- We still need to know how to *compute* the maximum of the log likelihood. Coming soon!