Classification

COMP9417 Machine Learning and Data Mining

Term 2, 2022

Acknowledgements

Material derived from slides for the book
"Elements of Statistical Learning (2nd Ed.)" by T. Hastie,
R. Tibshirani & J. Friedman. Springer (2009)
`http://statweb.stanford.edu/~tibs/ElemStatLearn/`

Material derived from slides for the book
"Machine Learning: A Probabilistic Perspective" by P. Murphy
MIT Press (2012)
`http://www.cs.ubc.ca/~murphyk/MLbook`

Material derived from slides for the book
"Machine Learning" by P. Flach
Cambridge University Press (2012)
`http://cs.bris.ac.uk/~flach/mlbook`

Material derived from slides for the book
"Bayesian Reasoning and Machine Learning" by D. Barber
Cambridge University Press (2012)
`http://www.cs.ucl.ac.uk/staff/d.barber/brml`

Material derived from slides for the book
"Machine Learning" by T. Mitchell
McGraw-Hill (1997)
`http://www-2.cs.cmu.edu/~tom/mlbook.html`

Material derived from slides for the course
"Machine Learning" by A. Srinivasan
BITS Pilani, Goa, India (2016)

Aims

This lecture will introduce you to machine learning approaches to the problem of *classification*. Following it you should be able to reproduce theoretical results, outline algorithmic techniques and describe practical applications for the topics:

- outline the probem of learning to classify
- outline a framework for solving machine learning problems
- describe issues of generalisation and evaluation for classification
- outline the use of a linear model as a 2-class classifier
- outline the Perceptron classification algorithm
- outline the Logistic Regression classification algorithm
- compare Maximum Likelihood and Bayesian classification

Introduction

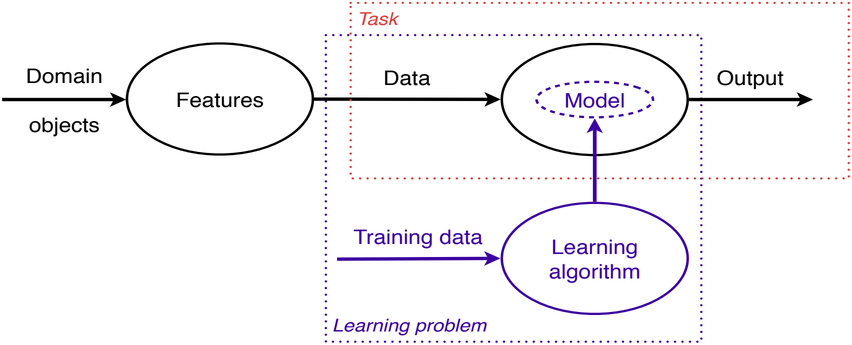
Classification (sometimes called *concept learning*) methods dominate machine learning . . .

. . . however, they often don't have convenient mathematical properties like regression, so can be more complicated to analyse. The idea is to learn a *classifier*, which is usually a function mapping from an input data point to one of a set of discrete outputs, i.e., the *classes*.

We will mostly focus on their advantages and disadvantages as learning methods first, and point to unifying ideas and approaches where applicable. In this lecture we focus on classification methods that are essentially *linear models* . . .

and in later lectures we will see other, more expressive, classifier learning methods.

How machine learning helps to solve a task



An overview of how machine learning is used to address a given task. A task (red box) requires an appropriate mapping – a model – from data described by features to outputs. Obtaining such a mapping from training data is what constitutes a learning problem (blue box).

Some terminology I

Tasks are addressed by models, whereas learning problems are solved by learning algorithms that produce models.

Some terminology II

Machine learning is concerned with using the right features to build the right models that achieve the right tasks.

Some terminology III

Models lend the machine learning field diversity, but tasks and features give it unity.

Some terminology IV

Does the algorithm require all training data to be present before the start of learning ? If yes, then it is categorised as **batch learning** algorithm.

If however, it can continue to learn a new data arrives, it is an **online learning** algorithm.

Some terminology V

If the model has a fixed number of parameters it is categorised as **parametric**.

Otherwise, if the number of parameters grows as part of training it is categorised as **non-parametric**.

Example: assassinating spam e-mail

SpamAssassin is a widely used open-source spam filter. It calculates a score for an incoming e-mail, based on a number of built-in rules or ‘tests’ in SpamAssassin’s terminology, and adds a ‘junk’ flag and a summary report to the e-mail’s headers if the score is 5 or more.

```
-0.1 RCVD_IN_MXRATE_WL       RBL: MXRate recommends allowing
                             [123.45.6.789 listed in sub.mxrate.net]
 0.6 HTML_IMAGE_RATIO_02     BODY: HTML has a low ratio of text to image area
 1.2 TVD_FW_GRAPHIC_NAME_MID BODY: TVD_FW_GRAPHIC_NAME_MID
 0.0 HTML_MESSAGE            BODY: HTML included in message
 0.6 HTML_FONx_FACE_BAD      BODY: HTML font face is not a word
 1.4 SARE_GIF_ATTACH         FULL: Email has a inline gif
 0.1 BOUNCE_MESSAGE          MTA bounce message
 0.1 ANY_BOUNCE_MESSAGE      Message is some kind of bounce message
 1.4 AWL                     AWL: From: address is in the auto white-list
```

From left to right you see the score attached to a particular test, the test identifier, and a short description including a reference to the relevant part of the e-mail. As you see, scores for individual tests can be negative (indicating evidence suggesting the e-mail is ham rather than spam) as well as positive. The overall score of 5.3 suggests the e-mail might be spam.

Linear classification

Suppose we have only two tests and four training e-mails, one of which is spam. Both tests succeed for the spam e-mail; for one ham e-mail neither test succeeds, for another the first test succeeds and the second doesn't, and for the third ham e-mail the first test fails and the second succeeds.

It is easy to see that assigning both tests a weight of 4 correctly 'classifies' these four e-mails into spam and ham. In the mathematical notation introduced above we could describe this classifier as $4x_1 + 4x_2 > 5$ or $(4, 4) \cdot (x_1, x_2) > 5$.

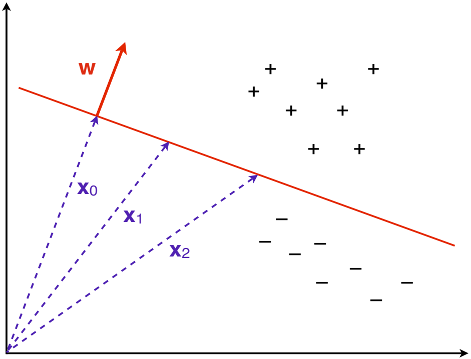
In fact, any weight between 2.5 and 5 will ensure that the threshold of 5 is only exceeded when both tests succeed. We could even consider assigning different weights to the tests – as long as each weight is less than 5 and their sum exceeds 5 – although it is hard to see how this could be justified by the training data.

Spam filtering as a classification task

The columns marked x_1 and x_2 indicate the results of two tests on four different e-mails. The fourth column indicates which of the e-mails are spam. The right-most column demonstrates that by thresholding the function $4x_1 + 4x_2$ at 5, we can separate spam from ham.

E-mail	x_1	x_2	Spam?	$4x_1 + 4x_2$
1	1	1	1	8
2	0	0	0	0
3	1	0	0	4
4	0	1	0	4

Linear classification in two dimensions



- straight line separates positives from negatives (linear “discriminant”)
- defined by $\mathbf{w} \cdot \mathbf{x}_i = t$
- $\mathbf{w}$ is perpendicular to decision boundary
- $\mathbf{w}$ points in direction of positives
- t is the decision threshold

Linear classification in two dimensions

Note: $\mathbf{x}_i$ points to a point on the decision boundary.

In particular, $\mathbf{x}_0$ points in the same direction as $\mathbf{w}$,

from which it follows that $\mathbf{w} \cdot \mathbf{x}_0 = ||\mathbf{w}||\ ||\mathbf{x}_0|| = t$ (where $||\mathbf{x}||$ denotes the length of the vector $\mathbf{x}$).

Homogeneous coordinates

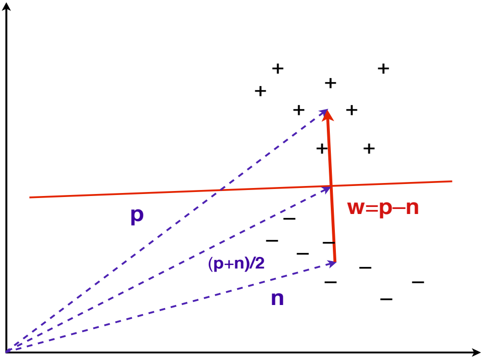
It is sometimes convenient to simplify notation further by introducing an extra constant ‘variable’ $x_0 = 1$, the weight of which is fixed to $w_0 = -t$.

The extended data point is then $\mathbf{x}^\circ = (1, x_1, \ldots, x_n)$ and the extended weight vector is $\mathbf{w}^\circ = (-t, w_1, \ldots, w_n)$, leading to the decision rule $\mathbf{w}^\circ \cdot \mathbf{x}^\circ > 0$ and the decision boundary $\mathbf{w}^\circ \cdot \mathbf{x}^\circ = 0$.

Thanks to these so-called *homogeneous coordinates* the decision boundary passes through the origin of the extended coordinate system, at the expense of needing an additional dimension.

Note: this doesn’t really affect the data, as all data points and the ‘real’ decision boundary live in the plane $x_0 = 1$.

Example: a simple linear classifier I

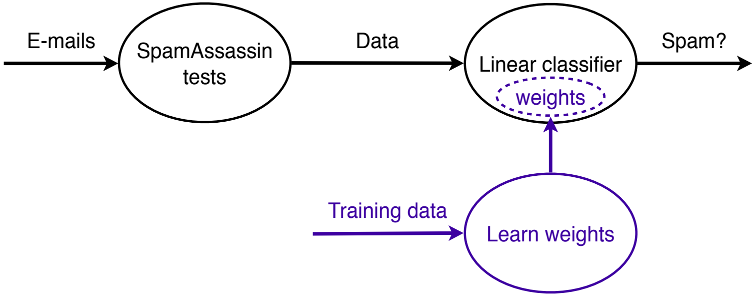


This classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass.

Example: a simple linear classifier II

The linear classifier is described by the equation $\mathbf{w} \cdot \mathbf{x} = t$, with $\mathbf{w} = \mathbf{p} - \mathbf{n}$; the decision threshold can be found by noting that $(\mathbf{p} + \mathbf{n})/2$ is on the decision boundary, and hence $t = (\mathbf{p} - \mathbf{n}) \cdot (\mathbf{p} + \mathbf{n})/2 = (||\mathbf{p}||^2 - ||\mathbf{n}||^2)/2$, where $||\mathbf{x}||$ denotes the length of vector $\mathbf{x}$.

Machine learning for spam filtering



At the top we see how SpamAssassin approaches the spam e-mail classification task: the text of each e-mail is converted into a data point by means of SpamAssassin's built-in tests, and a *linear classifier* is applied to obtain a 'spam or ham' decision. At the bottom (in blue) we see the bit that is done by machine learning.

Evaluating classification – contingency table I

For the two-class prediction case:

Actual Class	Predicted Class	
	Yes	No
Yes	True Positive (TP)	False Negative (FN)
No	False Positive (FP)	True Negative (TN)

Evaluating classification – contingency table II

Classification Accuracy on a sample of labelled pairs $(x, c(x))$ given a learned classification model that predicts, for each instance x, a class value $\hat{c}(x)$:

$$\text{acc} = \frac{1}{|\text{Test}|} \sum_{x \in \text{Test}} I[\hat{c}(x) = c(x)]$$

where Test is a test set and $I[]$ is the indicator function which is 1 iff its argument evaluates to true, and 0 otherwise.

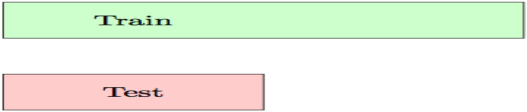
Classification Error is 1-acc.

Cross-validation I

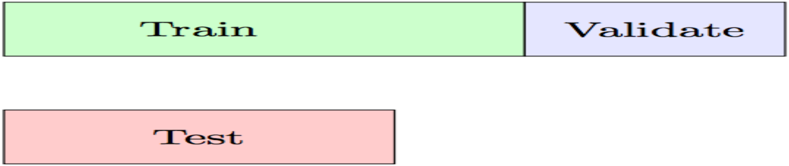
There are certain parameters that need to be estimated during learning.

We use the data, but NOT the training set, OR the test set. Instead, we use a separate *validation* or *development* set.

Cross-validation II



Cross-validation III

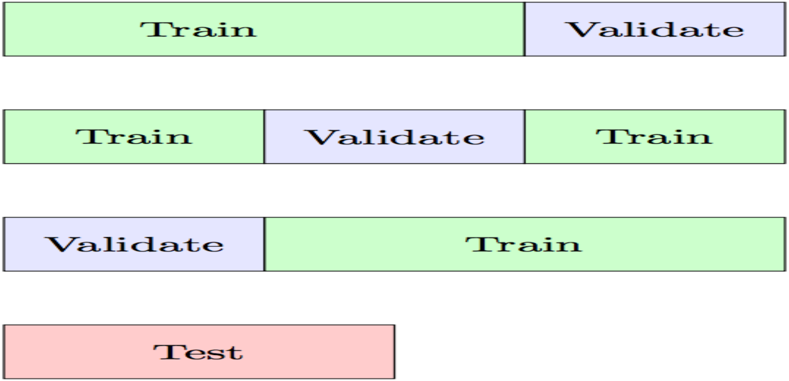


Train

Validate

Test

Cross-validation IV

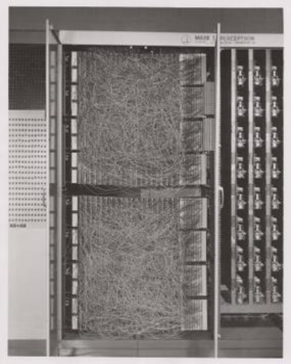


Perceptron

A linear classifier that can achieve perfect separation on linearly separable data is the *perceptron*, originally proposed as a simple *neural network* by F. Rosenblatt in the late 1950s.

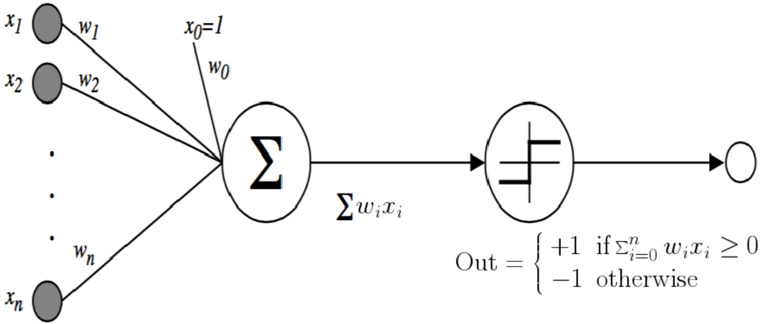
Perceptron

Originally implemented in software (based on the McCulloch-Pitts neuron from the 1940s), then in hardware as a 20x20 visual sensor array with potentiometers for adaptive weights.



Source http://en.wikipedia.org/w/index.php?curid=47541432

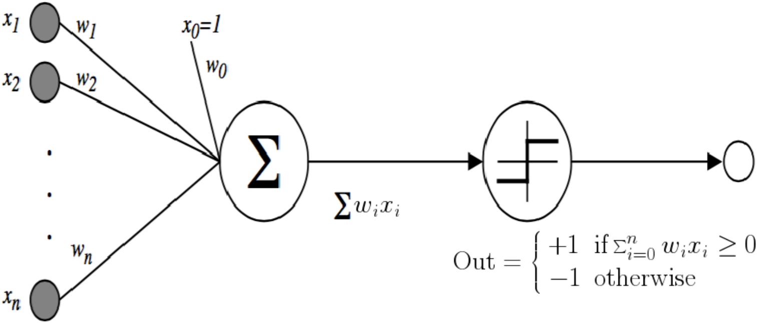
Perceptron


x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum w_i x_i$
$\text{Out} = \begin{cases} +1 & \text{if } \Sigma_{i=0}^{n} w_i x_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$

Output o is thresholded sum of products of inputs and their weights:

$$o(x_1, \ldots, x_n) = \begin{cases} +1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron

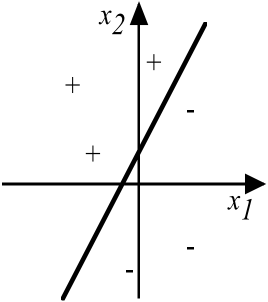


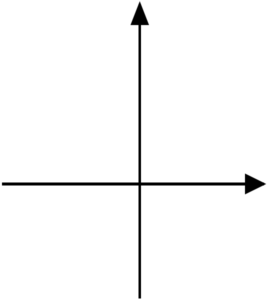
$$\text{Out} = \begin{cases} +1 & \text{if } \Sigma_{i=0}^{n} w_i x_i \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Or in vector notation:

$$o(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Decision Surface of a Perceptron





Represents some useful functions

- What weights represent $o(x_1, x_2) = AND(x_1, x_2)$?
- What weights represent $o(x_1, x_2) = XOR(x_1, x_2)$?

Decision Surface of a Perceptron

So some functions not representable

- e.g., not linearly separable
    - a labelled data set is linearly separable if there is a linear decision boundary that separates the classes
- for non-linearly separable data we'll need something else
    - e.g., networks of these ...
    - the start of "deep" networks ...

Perceptron learning

Online learner

Key idea:

"prequential":

Learning is "finding a good set of weights"

Data: (x, y)

Given: x.

Predict: $\hat{y}$

Check ?

$\hat{y} = y$

Perceptron learning is simply an iterative weight-update scheme:

$$w_i \leftarrow w_i + \Delta w_i$$

where the weight update Δw_i depends only on *misclassified* examples and is modulated by a "smoothing" parameter η typically referred to as the "learning rate".

Can prove that perceptron learning will converge:

- if training data is linearly separable
- and η sufficiently small

Perceptron learning

The perceptron iterates over the training set, updating the weight vector every time it encounters an incorrectly classified example.

- For example, let $\mathbf{x}_i$ be a misclassified positive example, then we have $y_i = +1$ and $\mathbf{w} \cdot \mathbf{x}_i < t$. We therefore want to find $\mathbf{w}'$ such that $\mathbf{w}' \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$, which moves the decision boundary towards and hopefully past $\mathbf{x}_i$.
- This can be achieved by calculating the new weight vector as $\mathbf{w}' = \mathbf{w} + \eta \mathbf{x}_i$, where $0 < \eta \leq 1$ is the *learning rate* (again, assume set to 1). We then have $\mathbf{w}' \cdot \mathbf{x}_i = \mathbf{w} \cdot \mathbf{x}_i + \eta \mathbf{x}_i \cdot \mathbf{x}_i > \mathbf{w} \cdot \mathbf{x}_i$ as required.
- Similarly, if $\mathbf{x}_j$ is a misclassified negative example, then we have $y_j = -1$ and $\mathbf{w} \cdot \mathbf{x}_j > t$. In this case we calculate the new weight vector as $\mathbf{w}' = \mathbf{w} - \eta \mathbf{x}_j$, and thus $\mathbf{w}' \cdot \mathbf{x}_j = \mathbf{w} \cdot \mathbf{x}_j - \eta \mathbf{x}_j \cdot \mathbf{x}_j < \mathbf{w} \cdot \mathbf{x}_j$.

Perceptron learning

- The two cases can be combined in a single update rule:

$$\mathbf{w}' = \mathbf{w} + \eta y_i \mathbf{x}_i$$

- Here y_i acts to change the sign of the update, corresponding to whether a positive or negative example was misclassified
- This is the basis of the *perceptron training algorithm* for linear classification
- The algorithm just iterates over the training examples applying the weight update rule until all the examples are correctly classified
- If there is a linear model that separates the positive from the negative examples, i.e., the data is linearly separable, it can be shown that the perceptron training algorithm will converge in a finite number of steps.

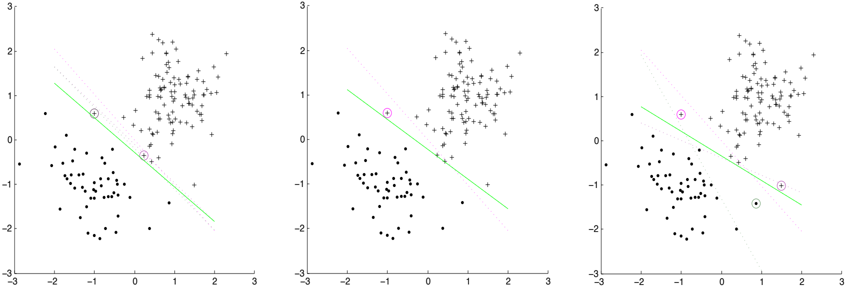
Perceptron training algorithm

```
Algorithm Perceptron($D, \eta$) // perceptron training for linear classification
Input: labelled training data $D$ in homogeneous coordinates; learning rate $\eta$.
Output: weight vector $\mathbf{w}$ defining classifier $\hat{y} = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x})$.
$\mathbf{w} \leftarrow \mathbf{0}$ // Other initialisations of the weight vector are possible
$converged \leftarrow$ false
while $converged =$ false do
    $converged \leftarrow$ true
    for $i = 1$ to $|D|$ do
        if $y_i \mathbf{w} \cdot \mathbf{x}_i \leq 0$ then          // i.e., $\hat{y_i} \neq y_i$
            $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$
            $converged \leftarrow$ false  // We changed $\mathbf{w}$ so haven't converged yet
        end
    end
end
```

Modify: make η depend on epoch.

$$\text{Error rate} = \frac{FP + FN}{\#\text{ of examples}}$$

Perceptron training – varying learning rate



(left) A perceptron trained with a small learning rate ($\eta = 0.2$). The circled examples are the ones that trigger the weight update. (middle) Increasing the learning rate to $\eta = 0.5$ leads in this case to a rapid convergence. (right) Increasing the learning rate further to $\eta = 1$ may lead to too aggressive weight updating, which harms convergence.

The starting point in all three cases was the simple linear classifier.

Linear Regression for classification

Question: can we use Linear Regression for classification ?

Answer: not really, unless we use an approach like the following, e.g.,

Training: train a separate linear regression model for each class

- set $y = 1$ if example in class
- set $y = 0$ otherwise

Classes $\in \{r, g, b\}$

3 models: $\hat{y}_r$; $\hat{y}_g$; $\hat{y}_b$

Prediction: for each example

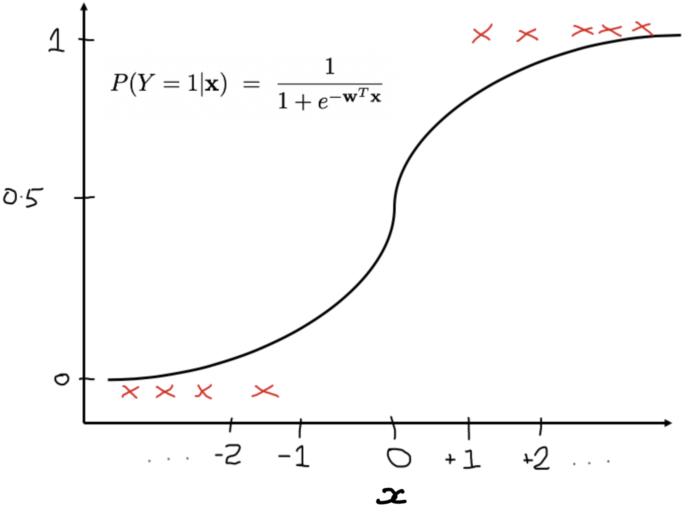
- run example x_i through all regression models
- predict the class with the largest output value for y

Called *multi-response* linear regression.[1]

Note: does not obey linear regression assumptions, but may work in practice.

[1] See: Witten et al. (2017).

Logistic regression



$$P(Y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

Maximum Likelihood Principle

Likelihood : $$L(\theta ; X) \triangleq p(X ; \theta)$$

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{n} p(x_i ; \theta)$$

Assumes : x_i (data) are independent

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log L(\theta ; X)$$

Model is a probabilistic / random "coin-flipping" mechanism that generates data

Example: ML estimation of parameter θ of a Bernoulli distribution (coin flip).

Let X be a Bernoulli r.v., $X \in \{0, 1\}$

$$p(X = x) = \begin{cases} \theta & \text{if } x = 1 \\ 1 - \theta & \text{if } x = 0 \end{cases}$$

$$= \theta^{x} (1 - \theta)^{1-x}$$

and: data $\{x_1, \ldots, x_n\}$, each x_i is either 1 or 0

$$L(\theta ; x_1, \dots, x_n) = \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1-x_i}$$

$$= \prod_{i=1}^{n} \theta^{x_i}(1-\theta)^{1}(1-\theta)^{-x_i}$$

$$= \theta^{h}(1-\theta)^{n}(1-\theta)^{-h}$$ where $h = \sum x_i$

For MLE:

$$\hat{\theta} = \arg\max_{\theta} \theta^{h}(1-\theta)^{n-h} , \quad 0 \le \theta \le 1 .$$

Take derivative, set to zero.

$$\frac{\partial}{\partial \theta}\left[\theta^{h}(1-\theta)^{n-h}\right]$$

$$= \boxed{h\theta^{h-1}(1-\theta)^{n-h} + \theta^{h}(n-h)(1-\theta)^{n-h-1}(-1)}$$

Derivative is zero if:

(i) $\theta = 0$ (ii) $\theta = 1$

(iii) $h(1-\theta) = (n-h)\theta$

$\Rightarrow h - h\theta = n\theta - h\theta$

$\Rightarrow h = n\theta$

$\Rightarrow \underline{\underline{\theta = h/n}}$

$$\boxed{\hat{\theta} = \frac{\text{\# of 1's in data}}{\text{\# of } x\text{'s in data}}}$$

MLE: $\hat{\theta}$ = "proportion of $X=1$"

i.e., number of heads in a sequence of coin flips.

Logistic regression

Conditional Likelihood

Data $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

In the case of a two-class classification problem, if we model the probability $P(Y = 1)$ of an instance $\mathbf{x}$ being a positive example like this:

$Y \in \{0,1\}$

$$p = P(Y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

$L(\theta; X) \hat{=} p(X; \theta)$

$p(Y=1 | x; \theta)$

then this probability vs. the alternative $(1 - P(Y = 1))$ can be written like this: "log odds"

$$\log \frac{p}{1-p}$$

$$\ln \frac{P(Y = 1|\mathbf{x})}{1 - P(Y = 1|\mathbf{x})} = \boxed{\mathbf{w}^T\mathbf{x}}$$

$p \in [0, 1]$

odds $\in [0, +\infty)$

log odds $\in (-\infty, +\infty)$

The quantity on the l.h.s. is called the *logit* and we are defining a linear model for the logit.

Logistic regression

Unlike linear regression, no analytical maximum likelihood (ML) solution to find weights $\mathbf{w}$.

An iterative gradient ascent method can be used to maximize log likelihood.

The (conditional) log likelihood is:

$$\sum_{i=1}^{N} y^{(i)} \log P(1|\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - P(1|\mathbf{x}^{(i)}))$$

Over a set of N examples, choose $\mathbf{w}$ to maximise this expression, noting that $y^{(i)}$ is always either 0 or 1.

Generalises to multiple class versions (Y can have more than two values).

Bayes Theorem

Bayes Theorem for estimating probability of model $\overset{\text{i.e., parameters } \theta}{\overset{\uparrow}{m}}$ from data D:

$$\overset{\text{Posterior prob.}}{P(m|D)} = \frac{\overset{\text{Likelihood}}{P(D|m)}\overset{\text{Prior (model)}}{P(m)}}{\underset{\text{Prior (data)}}{P(D)}}$$

where

$P(m)$ = prior probability of model m
$P(D)$ = prior probability of training data D
$P(m|D)$ = probability of m given D
$P(D|m)$ = probability of D given m

Finite set $\mathcal{M}$ of models

Size of $|\mathcal{M}| = S$.

Uniform prior = $\frac{1}{S}$ $\forall$ models $\in \mathcal{M}$

Choosing a model

$$P(m|D) = \frac{P(D|m)P(m)}{P(D)}$$

Generally, we want the most probable model given the training data

Maximum a posteriori hypothesis m_{MAP}:

$$\begin{aligned} m_{MAP} &= \arg\max_{m \in \mathcal{M}} P(m|D) \\ &= \arg\max_{m \in \mathcal{M}} \frac{P(D|m)P(m)}{P(D)} \\ &= \arg\max_{m \in \mathcal{M}} P(D|m)P(m) \end{aligned}$$

Choosing a model

If assume $P(m_i) = P(m_j)$ then can further simplify, and choose the *Maximum Likelihood* (ML) hypothesis

$$h_{ML} = \arg \max_{m_i \in \mathcal{M}} P(D|m_i)$$

This assumption means we believe all models are *a priori* equally likely.

A Bayesian framework for Classification

1. Define a prior on models
2. Define the likelihood, i.e., the probability of the data given the model
3. Learning is finding the required parameters by fitting models to data
4. Predict (classify) using, e.g., the MAP model

A Maximum Likelihood framework for Classification

1. Define the likelihood, i.e., the probability of the data given the model
2. Learning is finding the required parameters by fitting models to data
3. Predict (classify) using, e.g., the ML model

Bayes Error

What is the best performance attainable by a (two-class) classifier ?

Define the probability of error for classifying some instance x by

$$\begin{aligned} P(\text{error}|x) &= P(\text{class}_1|x) \quad \text{if we predict class}_2 \\ &= P(\text{class}_2|x) \quad \text{if we predict class}_1 \end{aligned}$$

Only know P(error) for synthetic data.

This gives

$$\Sigma_x \quad P(\text{error}) = P(\text{error}|x)\ P(x)$$

So we can justify the use of the decision rule

$$\begin{aligned} \text{if} \quad P(\text{class}_1|x) > P(\text{class}_2|x) \quad &\text{then} \quad \text{predict class}_1 \\ &\text{else} \quad \text{predict class}_2 \end{aligned}$$

On average, this decision rule minimises probability of classification error.

Most Probable Classification of New Instances

Most probable *model* given the data D is m_{MAP}.

Given new instance x, what is its most probable *classification* ?

- $m_{MAP}(x)$ not necessarily the most probable classification!

Most Probable Classification of New Instances

Consider:

- Three possible models:

  $$P(m_1|D) = .4,\ P(m_2|D) = .3,\ P(m_3|D) = .3$$

- Given new instance x,

  $$m_1(x) = +,\ m_2(x) = -,\ m_3(x) = -$$

- What's most probable classification of x?

Bayes Optimal Classifier

Bayes optimal classification:

$$\arg\max_{y_j \in \mathcal{Y}} \sum_{m_i \in \mathcal{M}} P(y_j|m_i)P(m_i|D)$$

Example:

$$\begin{aligned}
P(m_1|D) = .4, \quad P(-|m_1) = 0, \quad P(+|m_1) = 1\\
P(m_2|D) = .3, \quad P(-|m_2) = 1, \quad P(+|m_2) = 0\\
P(m_3|D) = .3, \quad P(-|m_3) = 1, \quad P(+|m_3) = 0
\end{aligned}$$

Bayes Optimal Classifier

therefore

$$\begin{aligned}\sum_{m_i \in \mathcal{M}} P(+|m_i)P(m_i|D) &= .4 \\ \sum_{m_i \in \mathcal{M}} P(-|m_i)P(m_i|D) &= .6\end{aligned}$$

and

$$\arg\max_{y_j \in \mathcal{Y}} \sum_{m_i \in \mathcal{M}} P(y_j|m_i)P(m_i|D) = -$$

Key point: no other classification method using the same model space and same prior knowledge can outperform this method on average

Bayes Optimal Classifier

Problem: Bayes optimal classification is typically intractable and is therefore *not* used in practice !

- requires training *all* models
- therefore only a theoretical device
- but can be useful to consider methods as approximations to this

Summary

- Classification and classifier learning
- Classifier as a linear model
- Geometric interpretation of a linear classifier
- Perceptron learning
- Logistic regression
- Maximum Likelihood Principle *vs.* Bayes for classifiers
- So we have established a basis for learning classifiers
- Later we will see how to extend by building on these ideas

Witten, I., Frank, E., Hall, M., and Pal, C. (2017). *Data Mining (Fourth Edition)*. Morgan Kaufmann.