



Classification

David Barber

Classification

- This is a form of supervised learning.
- We have a dataset of input-output examples $(\mathbf{x}^n, c^n), n = 1, \dots, N$
- For example we might have an image, represented by a vector \mathbf{x} and a corresponding discrete labels:


$$\underbrace{\hspace{1.5cm}}_{\mathbf{x}^1}, \quad c^1 = 4$$


$$\underbrace{\hspace{1.5cm}}_{\mathbf{x}^2}, \quad c^2 = 7$$

\vdots

- We want to learn a mapping from inputs \mathbf{x} to class label c such that when we get a new input \mathbf{x} , we will output the correct class label.

Generalisation

Basic Assumptions

- The data we have (all inputs and corresponding labels) is generated from the same unvarying mechanism.
 - We will typically split our available data into three distinct parts: train data, validation data and test data.
 - We want to find a model that will have good generalisation performance.
-

Validation



Different models can be trained using the train data. The optimal model is determined by the empirical performance on the validation data. An independent measure of the generalisation performance of this optimal model is obtained by using a separate test set.

Generative Models

Do As Your Neighbour Does

- Each input vector \mathbf{x} has a corresponding class label, $c^n \in \{1, \dots, C\}$. Given a dataset of N train examples, $\mathcal{D} = \{\mathbf{x}^n, c^n\}, n = 1, \dots, N$, and a novel \mathbf{x} , we aim to return the correct class $c(\mathbf{x})$.
- For a classifier that works on unseen data, there must be some ‘smoothness’ in the underlying data generating process.
- If two points \mathbf{x}^1 and \mathbf{x}^2 that are very close can map to completely different class labels, there is no regularity in the process – the labels are essentially random.
- All classifiers assume some form of smoothness – the labels typically will not change much as we move a small distance in the input space.

A first classifier

- Nearest neighbour methods are a useful starting point since they readily encode basic smoothness intuitions and are easy to program.
- For novel \mathbf{x} , find the nearest input in the training set and use the class of this nearest input.

Nearest Neighbour: Squared Euclidean Distance

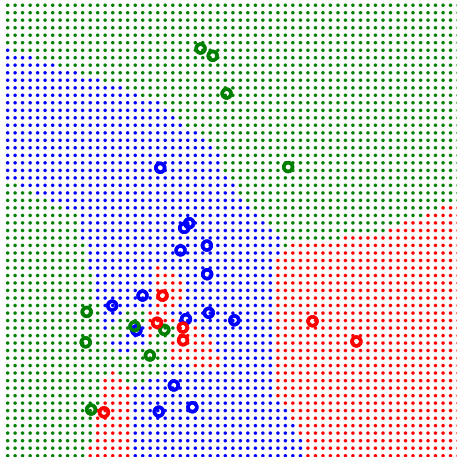
- For vectors \mathbf{x} and \mathbf{x}' representing two different datapoints, we measure 'nearness' using a dissimilarity function $d(\mathbf{x}, \mathbf{x}')$.
- A common dissimilarity is the squared Euclidean distance

$$d(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^T (\mathbf{x} - \mathbf{x}') = \sum_{i=1}^D (x_i - x'_i)^2$$

which can be more conveniently written $\|\mathbf{x} - \mathbf{x}'\|^2$.

- Based on the squared Euclidean distance, the decision boundary is determined by the perpendicular bisectors of the closest training points with different training labels.
- This partitions the input space into regions classified equally and is called a Voronoi tessellation.

Nearest Neighbour: Voronoi tessellation



Here there are three classes, with training points given by the circles, along with their class. The dots indicate the class of the nearest training vector. The decision boundary bisects two datapoints belonging to different classes.

Nearest Neighbour: taking care with distance

- If the length scales of the components of the vector \mathbf{x} vary greatly, the largest length scale will dominate the squared distance, with potentially useful class-specific information in other components of \mathbf{x} lost. ●
- For example, we might decide to represent a quantity in millimeters or in meters. This could heavily affect any distance calculations and completely change the classifier. ●
- The Mahalanobis distance

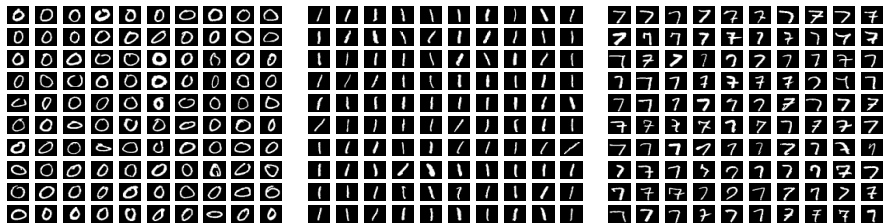
$$d(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^T \Sigma^{-1} (\mathbf{x} - \mathbf{x}')$$

where Σ is the covariance matrix of the inputs (from all classes) can overcome some of these problems since it effectively rescales the input vector components. ●

Nearest Neighbour: Comments

- The whole dataset needs to be stored to make a classification since the novel point must be compared to all of the train points. This can be partially addressed by removing datapoints which have little or no effect on the decision boundary.
- Particularly for low dimensional data, finding the nearest neighbour can be speeded up by various techniques that exploit the geometry of the space (see later).
- Each distance calculation can be expensive if the datapoints are high dimensional. Principal Components Analysis, is one way to address this and replaces \mathbf{x} with a low dimensional projection \mathbf{p} . The Euclidean distance of two datapoints $(\mathbf{x}^a - \mathbf{x}^b)^2$ is then approximately given by $(\mathbf{p}^a - \mathbf{p}^b)^2$. This is both faster to compute and can also improve classification accuracy since only the large scale characteristics of the data are retained in the PCA projections.
- It is not clear how to deal with missing data or incorporate prior beliefs and domain knowledge.

Handwritten Digit Example



Some of the train examples of the digit zero, one and seven. There are 300 train examples of each of these three digit classes.

KNN on handwritten digits (MNIST)

One versus Zero

- Each digit contains $28 \times 28 = 784$ pixels. The train data consists of 300 zeros, and 300 ones.
 - To test the performance of the nearest neighbour method (based on Euclidean distance) we use an independent test set containing a further 600 digits.
 - The nearest neighbour method, applied to this data, correctly predicts the class label of all 600 test points.
 - The reason for the high success rate is that examples of zeros and ones are sufficiently different that they can be easily distinguished.
-

One versus Seven

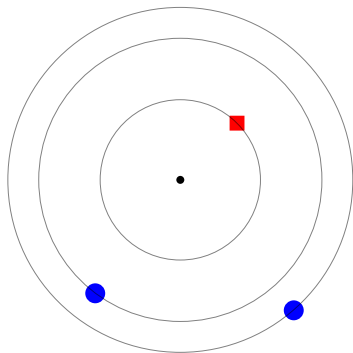
- This time, 18 errors are found using nearest neighbour classification – a 3% error rate for this two class problem.
- As an aside, the best methods classify real world digits (over all 10 classes) to an error of less than 1 percent – better than the performance of an ‘average’ human.

K -Nearest Neighbours

Making Nearest Neighbours more robust

- If your neighbour is simply mistaken (has an incorrect training class label), or is not a particularly representative example of his class, then these situations will typically result in an incorrect classification.
- By including more than the single nearest neighbour, we hope to make a more robust classifier with a smoother decision boundary (less swayed by single neighbour opinions).
- If we assume the Euclidean distance as the dissimilarity measure, the algorithm considers a hypersphere centred on the test point \mathbf{x} . The radius of the hypersphere is increased until it contains exactly K train inputs.
- The class label $c(\mathbf{x})$ is then given by the most numerous class within the hypersphere.

K -Nearest Neighbours



In K -nearest neighbours, we centre a hypersphere around the point we wish to classify (here the central dot). The inner circle corresponds to the nearest neighbour, a square. However, using the 3 nearest neighbours, we find that there are two round-class neighbours and one square-class neighbour– and we would therefore classify the central point as round-class.

Choosing K

- Whilst there is some sense in making $K > 1$, there is certainly little sense in making $K = N$ (N being the number of training points).
 - For K very large, all classifications will become the same – simply assign each novel \mathbf{x} to the most numerous class in the train data.
 - This suggests that there is an optimal intermediate setting of K which gives the best generalisation performance.
-

Using validation data

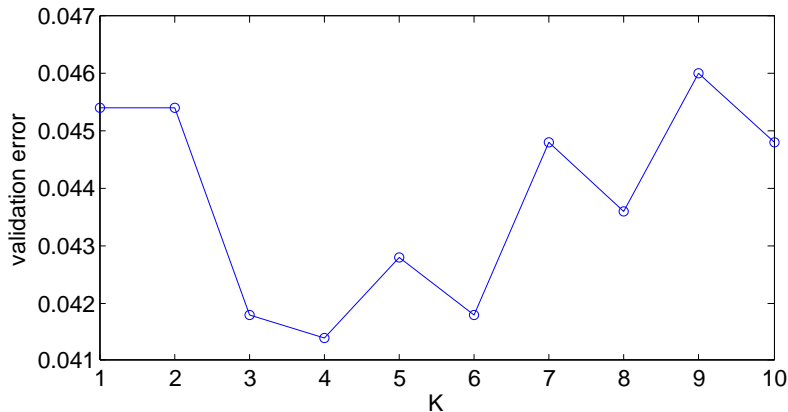
- Consider a validation set.
- For each K of interest, calculate the performance of the KNN classifier (whose class label is assigned on the basis of the train data).
- Choose K based on the best validation performance.

KNN: Choosing K

MNIST Digits (all 10 classes)

There are 15,000 train datapoints and 5000 validation points.

`demoKNNlearnK.m`



$K = 4$ is the best choice, giving an error of around 4.1%.

Probabilistic Interpretation of Nearest Neighbours

Consider the situation where we have data from two classes – class 0 and class 1. We make the following mixture model for data from class 0, placing a Gaussian on each datapoint:

$$p(\mathbf{x}|c=0) = \frac{1}{N_0} \sum_{n \in \text{class } 0} \mathcal{N}(\mathbf{x}|\mathbf{x}^n, \sigma^2 \mathbf{I})$$

where D is the dimension of a datapoint \mathbf{x} and N_0 are the number of train points of class 0, and σ^2 is the variance.

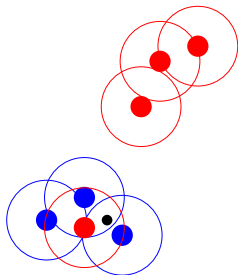
Similarly, for data from class 1:

$$p(\mathbf{x}|c=1) = \frac{1}{N_1} \sum_{n \in \text{class } 1} \mathcal{N}(\mathbf{x}|\mathbf{x}^n, \sigma^2 \mathbf{I})$$

To classify a new datapoint \mathbf{x}^* , we use Bayes' rule

$$p(c=0|\mathbf{x}^*) = \frac{p(\mathbf{x}^*|c=0)p(c=0)}{p(\mathbf{x}^*|c=0)p(c=0) + p(\mathbf{x}^*|c=1)p(c=1)}$$

Probabilistic Interpretation



A probabilistic interpretation of nearest neighbours. For each class we use a mixture of Gaussians to model the data from that class $p(\mathbf{x}|c)$, placing at each training point an isotropic Gaussian of width σ^2 . The width of each Gaussian is represented by the circle. In the limit $\sigma^2 \rightarrow 0$ a novel point (small black dot) is assigned the class of its nearest neighbour. For finite $\sigma^2 > 0$ the influence of non-nearest neighbours has an effect, resulting in a soft version of nearest neighbours.

Maximum Likelihood

- The maximum likelihood setting of $p(c = 0)$ is $N_0/(N_0 + N_1)$, and $p(c = 1) = N_1/(N_0 + N_1)$.
- An analogous expression holds for $p(c = 1|\mathbf{x}^*)$.
- To see which class is most likely we may use the ratio

$$\frac{p(c = 0|\mathbf{x}^*)}{p(c = 1|\mathbf{x}^*)} = \frac{p(\mathbf{x}^*|c = 0)p(c = 0)}{p(\mathbf{x}^*|c = 1)p(c = 1)}$$

If this ratio is greater than one, we classify \mathbf{x}^* as 0, otherwise 1.

Limiting case

$$\frac{p(c = 0|\mathbf{x}^*)}{p(c = 1|\mathbf{x}^*)} = \frac{p(\mathbf{x}^*|c = 0)p(c = 0)}{p(\mathbf{x}^*|c = 1)p(c = 1)}$$

- If σ^2 is very small, the numerator is dominated by that term for which datapoint \mathbf{x}^{n_0} in class 0 is closest to the point \mathbf{x}^* . Similarly, the denominator will be dominated by that datapoint \mathbf{x}^{n_1} in class 1 which is closest to \mathbf{x}^* .
- Taking the limit $\sigma^2 \rightarrow 0$, with certainty we classify \mathbf{x}^* as class 0 if \mathbf{x}^* is closer to \mathbf{x}^{n_0} than to \mathbf{x}^{n_1} .
- The nearest (single) neighbour method is therefore recovered as the limiting case of a probabilistic generative model,.

Probabilistic Interpretation

- The motivation for K nearest neighbours is to produce a classification that is robust against unrepresentative single nearest neighbours.
- To ensure a similar kind of robustness in the probabilistic interpretation, we may use a finite value $\sigma^2 > 0$.
- This smooths the extreme probabilities of classification and means that more points (not just the nearest) will have an effective contribution.
- The extension to more than two classes is straightforward, requiring a class conditional generative model for each class.
- By using a richer generative model of the data we may go beyond the Parzen estimator approach.

When your nearest neighbour is far away

- For a novel input \mathbf{x}^* that is far from all training points, Nearest Neighbours, and its soft probabilistic variant will confidently classify \mathbf{x}^* as belonging to the class of the nearest training point.
- This is arguably opposite to what we would like, namely that the classification should tend to the prior probabilities of the class based on the number of training data per class.
- A way to avoid this problem is, for each class, to include a fictitious large-variance mixture component at the mean of all the data, one for each class.
- For novel inputs close to the training data, this extra fictitious component will have no appreciable effect. However, as we move away from the high density regions of the training data, this additional fictitious component will dominate since it has larger variance than any of the other components.
- As the distance from \mathbf{x}^* to each fictitious class point is the same, in the limit that \mathbf{x}^* is far from the training data, the effect is that no class information from the position of \mathbf{x}^* occurs.

Naive Bayes

We form a joint model of a D -dimensional attribute (input) vector \mathbf{x} and the corresponding class c

$$p(\mathbf{x}, c) = p(c) \prod_{i=1}^D p(x_i | c)$$

Coupled with a suitable choice for each conditional distribution $p(x_i | c)$, we can then use Bayes' rule to form a classifier for a novel input vector \mathbf{x}^* :

$$p(c | \mathbf{x}^*) = \frac{p(\mathbf{x}^* | c)p(c)}{p(\mathbf{x}^*)} = \frac{p(\mathbf{x}^* | c)p(c)}{\sum_c p(\mathbf{x}^* | c)p(c)}$$

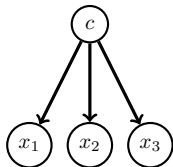


Figure: The central assumption is that given the class c , the attributes x_i are independent.

Naive Bayes example

- Consider the following vector of binary attributes:

(shortbread, lager, whiskey, porridge, football)

$\mathbf{x} = (1, 0, 1, 1, 0)^T$ is a person that likes shortbread, does not like lager, drinks whiskey, eats porridge, and has not watched England play football.

- Together with each vector \mathbf{x} , there is a label *nat* describing the nationality of the person, $\text{dom}(\text{nat}) = \{\text{scottish}, \text{english}\}$.
- We wish to classify the vector $\mathbf{x} = (1, 0, 1, 1, 0)^T$ as either scottish or english

Training data

0	1	1	1	0	0
0	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	0	1
1	0	1	0	1	0

(a) English

1	1	1	1	1	1	1
0	1	1	1	1	0	0
0	0	1	0	0	1	1
1	0	1	1	1	1	0
1	1	0	0	1	0	0

(b) Scottish

Naive Bayes example

$$p(\text{scottish}|\mathbf{x}) = \frac{p(\mathbf{x}|\text{scottish})p(\text{scottish})}{p(\mathbf{x}|\text{scottish})p(\text{scottish}) + p(\mathbf{x}|\text{english})p(\text{english})}$$

where $p(\text{scottish}) = 7/13$ and $p(\text{english}) = 6/13$.

$$p(\mathbf{x}|\text{nat}) = p(x_1|\text{nat})p(x_2|\text{nat})p(x_3|\text{nat})p(x_4|\text{nat})p(x_5|\text{nat})$$

$p(x_1 = 1 \text{english})$	$= 1/2$	$p(x_1 = 1 \text{scottish})$	$= 1$
$p(x_2 = 1 \text{english})$	$= 1/2$	$p(x_2 = 1 \text{scottish})$	$= 4/7$
$p(x_3 = 1 \text{english})$	$= 1/3$	$p(x_3 = 1 \text{scottish})$	$= 3/7$
$p(x_4 = 1 \text{english})$	$= 1/2$	$p(x_4 = 1 \text{scottish})$	$= 5/7$
$p(x_5 = 1 \text{english})$	$= 1/2$	$p(x_5 = 1 \text{scottish})$	$= 3/7$

For $\mathbf{x} = (1, 0, 1, 1, 0)^T$, we get

$$p(\text{scottish}|\mathbf{x}) = \frac{1 \times \frac{3}{7} \times \frac{3}{7} \times \frac{5}{7} \times \frac{4}{7} \times \frac{7}{13}}{1 \times \frac{3}{7} \times \frac{3}{7} \times \frac{5}{7} \times \frac{4}{7} \times \frac{7}{13} + \frac{1}{2} \times \frac{1}{2} \times \frac{1}{3} \times \frac{1}{2} \times \frac{1}{2} \times \frac{6}{13}} = 0.8076$$

Naive Bayes

- Naive Bayes is very fast to train.
- For discrete attributes, we just count the data.
- In the case of low counts, the method can be overly confident. One approach to deal with this is to add a fixed number of pseudocounts to each variable attribute for each class.
- Naive Bayes deals easily with more than two classes.
- Missing data is easily dealt with.
- We can easily deal with attributes that take more than two states or are continuous.
- SpamBayes is a popular spam/ham junk mail classifier – lightening fast to train with low storage costs.

Discriminative Models

Parametric Classification

Parametric methods define a classifier directly as a function of the input \mathbf{x} and some (to be learned) parameters $\boldsymbol{\theta}$.

Canonical example parametric two class method

$$c(\mathbf{x}; \boldsymbol{\theta}) = \begin{cases} 1 & \text{if } \boldsymbol{\theta}^T \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- This is called a perceptron and one of the earliest methods.
- It is called a linear classifier (it is a linear function of the parameters $\boldsymbol{\theta}$).
- Often consider extending \mathbf{x} to some non-linear feature vector with components $\psi_i(\mathbf{x})$ and use $\boldsymbol{\theta}^T \boldsymbol{\psi}(\mathbf{x})$ as the classifier.
- For example

$$\boldsymbol{\psi}(x_1, x_2) = \begin{pmatrix} x_1 \\ x_1 x_2 \\ \sin(x_2) \\ x_2^3 \end{pmatrix}$$

Loss functions

- For each input \mathbf{x} our predictor (which is a function $c(\mathbf{x}; \boldsymbol{\theta})$ of this input and the parameter $\boldsymbol{\theta}$) will produce a class label.
- We want this to match the train data label.
- If our predictor function has adjustable parameters, we will set them to minimise some loss criterion

$$\min_{\boldsymbol{\theta}} \sum_{n=1}^N L(c^n, c(\mathbf{x}^n; \boldsymbol{\theta}))$$

where $L(c^{true}, c^{pred})$ is a loss function that specifies how bad it is if we get the class label incorrect.

Zero-one loss

$$L(c^{true}, c^{pred}) = \begin{cases} 0 & \text{if } c^{pred} = c^{true} \\ 1 & \text{if } c^{pred} \neq c^{true} \end{cases}$$

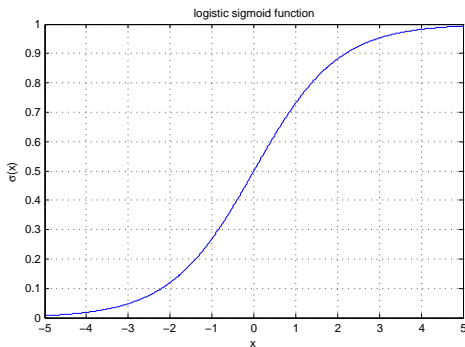
- This is in some sense an ‘ideal’ loss function.
- Error surface is very complex – difficult to train.

Probabilistic Models for Binary classification, $c \in \{0, 1\}$

$$p(c = 1|\mathbf{x}) = \sigma(b + \mathbf{x}^\top \mathbf{w}), \quad p(c = 0|\mathbf{x}) = 1 - p(c = 1|\mathbf{x})$$

here the parameters are given by the 'weight vector' \mathbf{w} 'bias' b and the 'logistic sigmoid' is defined by

$$\sigma(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

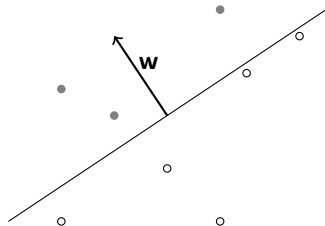


The decision boundary

The decision boundary is defined as that set of \mathbf{x} for which $p(c = 1|\mathbf{x}) = 0.5$:

$$b + \mathbf{x}^T \mathbf{w} = 0$$

If \mathbf{x} is on the decision boundary, so is \mathbf{x} plus any vector perpendicular to \mathbf{w} . In D dimensions, the space of vectors that are perpendicular to \mathbf{w} occupy a $D - 1$ dimensional hyperplane.



The decision boundary $p(c = 1|\mathbf{x}) = 0.5$ (solid line). For two dimensional data, the decision boundary is a line. If all the training data for class 1 (filled circles) lie on one side of the line, and for class 0 (open circles) on the other, the data is said to be linearly separable. More generally, \mathbf{w} defines the normal to a hyperplane and data is linearly separable if data from each of the two classes lies on opposite sides of the hyperplane.

Linear Separability

If all the data for class 1 lies on one side of a hyperplane, and for class 0 on the other, the data is said to be linearly separable.

Non linearly separable example



The XOR problem with each component of the training inputs on the cube $x_i \in \{+1, -1\}$. This is not linearly separable.

Making the data linearly separable

- We can map using a non-linear vector function $\psi(\mathbf{x})$.
- Eg

$$\psi(x_1, x_2) = \begin{pmatrix} (x_1 - 1)(x_2 - 1) \\ (x_1 + 1)(x_2 + 1) \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

- For the grey class data, $\mathbf{w}^T \psi$ will be 4. For the other class it will be 0.
- In this case the ψ datapoints are linearly separable.
- Similarly we can map to a higher dimension so that the data becomes linearly separable.

Maximum likelihood training

IID assumption

Assuming the data is identically and independently drawn from the same mechanism, the likelihood of the data is

$$\prod_{n=1}^N p(c^n | \mathbf{x}^n, b, \mathbf{w}) = \prod_{n=1}^N p(c = 1 | \mathbf{x}^n, b, \mathbf{w})^{c^n} (1 - p(c = 1 | \mathbf{x}^n, b, \mathbf{w}))^{1-c^n}$$

where we have used the fact that $c^n \in \{0, 1\}$.

Log likelihood function

For logistic regression this gives the log likelihood as

$$\mathcal{L}(\mathbf{w}, b) = \sum_{n=1}^N c^n \log \sigma(b + \mathbf{w}^\top \mathbf{x}^n) + (1 - c^n) \log (1 - \sigma(b + \mathbf{w}^\top \mathbf{x}^n))$$

Optimisation

Maximise \mathcal{L} numerically with respect to the weights \mathbf{w} and bias b .

Gradient ascent optimisation

One of the simplest methods is gradient ascent for which the gradient is given by

$$\nabla_{\mathbf{w}} \mathcal{L} = \sum_{n=1}^N (c^n - \sigma(\mathbf{w}^T \mathbf{x}^n + b)) \mathbf{x}^n$$

The gradient ascent procedure then corresponds to updating the weights using

$$\mathbf{w}^{new} = \mathbf{w} + \eta \nabla_{\mathbf{w}} \mathcal{L}$$

where the learning rate η is chosen small enough to ensure convergence. The application of the above rule will lead to a gradual increase in the log likelihood. The bias is updated similarly.

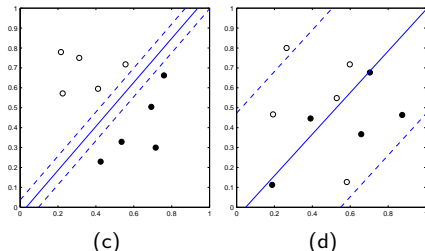


Figure: The decision boundary $p(c = 1|\mathbf{x}) = 0.5$ (solid line) and confidence boundaries $p(c = 1|\mathbf{x}) = 0.9$ and $p(c = 1|\mathbf{x}) = 0.1$ after 10000 iterations of batch gradient ascent with $\eta = 0.1$. **(a):** Linearly separable data. **(b):** Non-linearly separable data. Note how the confidence interval remains broad.

Example: Classifying handwritten '1' versus '7'

- We apply logistic regression to the 600 handwritten digits in which there are 300 ones and 300 sevens in the data.
- Using gradient ascent training with a suitably chosen stopping criterion, the number of errors made on the 600 test points is 12, compared with 14 errors using Nearest Neighbour methods.

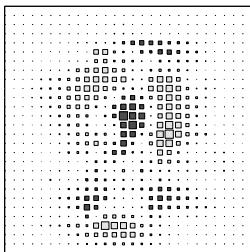


Figure: Logistic regression for classifying handwritten digits 1 and 7. Displayed is a Hinton diagram of the 784 learned weight vector w , plotted as a 28×28 image for visual interpretation. Light squares are positive weights and an input x with a (positive) value in this component will tend to increase the probability that the input is classed as a 7. Similarly, inputs with positive contributions in the dark regions tend to increase the probability as being classed as a 1 digit. Note that the elements of each input x are either positive or zero.

Geometry of the error surface

Hessian

The Hessian of the log likelihood $\mathcal{L}(\mathbf{w})$ is the matrix with elements

$$H_{ij} \equiv \frac{\partial^2 \mathcal{L}}{\partial w_i \partial w_j} = - \sum_n x_i^n x_j^n \sigma^n (1 - \sigma^n)$$

Concavity

This is negative semidefinite since, for any \mathbf{z} ,

$$\sum_{ij} z_i H_{ij} z_j = - \sum_{i,j,n} z_i x_i^n z_j x_j^n \sigma^n (1 - \sigma^n) \leq - \sum_n \left(\sum_i z_i x_i^n \right)^2 \leq 0$$

This means that the error surface is concave (an upside down bowl) and batch gradient ascent converges to the optimal solution, provided the learning rate η is small enough.

Beyond first order gradient ascent

- Gradient ascent is easy to implement but slow to converge.
- Since the surface has a single optimum, a Newton update

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \eta \mathbf{H}^{-1} \mathbf{g}$$

where \mathbf{H} is the Hessian matrix as above and $0 < \eta < 1$, will typically converge much faster than gradient ascent.

- However, for large scale problems with $\dim(\mathbf{w}) \gg 1$, the inversion of the Hessian is computationally demanding and limited memory BFGS or conjugate gradient methods are more practical alternatives.

Sparse data

- If only a small number of the elements of \mathbf{x} are non-zero, the data is called sparse.
- In this case we can implement conjugate gradient methods very efficiently (see notes).
- We can easily train logistic regression using millions of dimensions and numbers of training datapoints.

Avoiding overconfident classification

- Provided the data is linearly separable the weights will continue to increase and the classifications will become extreme.
- This is undesirable since the resulting classifications will be over-confident.
- One way to prevent this is early stopping in which only a limited number of gradient updates are performed.
- An alternative method is to add a penalty term to the objective function

$$\mathcal{L}'(\mathbf{w}, b) = \mathcal{L}(\mathbf{w}, b) - \alpha \mathbf{w}^T \mathbf{w}.$$

The scalar constant $\alpha > 0$ encourages smaller values of \mathbf{w} (remember that we wish to maximise the log likelihood). An appropriate value for α can be determined using validation data.

- The objective \mathcal{L}' is still concave so that training is numerically easy.

More than two classes

Softmax regression

For more than two classes $C > 2$ the logistic regression model is easily extendible using

$$p(c|x) = \frac{e^{\mathbf{w}_c^T \phi(x)}}{\sum_d e^{\mathbf{w}_d^T \phi(x)}}$$

where ϕ is a defined vector function of x and we have a vector \mathbf{w} for each class. The function

$$f(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

is called the softmax function and is a smooth version of the max function.

Training surface

One can show that the log likelihood is a jointly concave function of $\mathbf{w}_1, \dots, \mathbf{w}_C$.

Support Vector Machines for binary classification

- SVMs are linear classifiers that encourage good generalisation performance.
 - SVMs do not fit comfortably within a probabilistic framework
-

Maximum margin linear classifier

In the SVM literature it is common to use $+1$ and -1 to denote the two classes. For a hyperplane defined by weight \mathbf{w} and bias b , the classifier is

$$\mathbf{w}^T \mathbf{x} + b \begin{cases} \geq 0 & \text{class } +1 \\ < 0 & \text{class } -1 \end{cases}$$

To make the classifier robust we impose that the decision boundary should be separated from the data by some finite amount ϵ^2 (assuming in the first instance that the data is linearly separable):

$$\mathbf{w}^T \mathbf{x} + b \begin{cases} \geq \epsilon^2 & \text{class } +1 \\ < -\epsilon^2 & \text{class } -1 \end{cases}$$

SVM objective function

To classify the training labels correctly and maximise the margin, the optimisation problem is equivalent to:

$$\text{minimise } \frac{1}{2} \mathbf{w}^T \mathbf{w} \quad \text{subject to } y^n (\mathbf{w}^T \mathbf{x}^n + b) \geq 1, \quad n = 1, \dots, N$$

This is a quadratic programming problem.

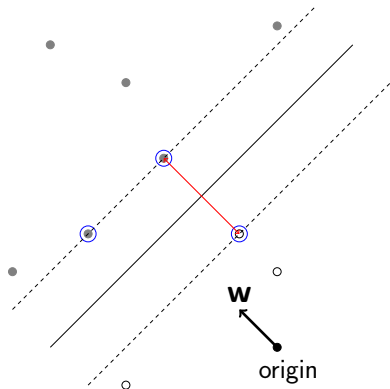


Figure: SVM classification of data from two classes (open circles and filled circles). The decision boundary $\mathbf{w}^T \mathbf{x} + b = 0$ (solid line). For linearly separable data the maximum margin hyperplane is equidistant from the closest opposite class points. These support vectors are highlighted in blue and the margin in red. The distance of the decision boundary from the origin is $-b/\sqrt{\mathbf{w}^T \mathbf{w}}$, and the distance of a general point \mathbf{x} from the origin along the direction \mathbf{w} is $\mathbf{x}^T \mathbf{w} / \sqrt{\mathbf{w}^T \mathbf{w}}$.

Non-linearly separable data

- To account for potentially mislabelled training points (or for data that is not linearly separable), we relax the exact classification constraint and use instead

$$y^n (\mathbf{w}^\top \mathbf{x}^n + b) \geq 1 - \xi^n$$

where the 'slack variables' are $\xi^n \geq 0$.

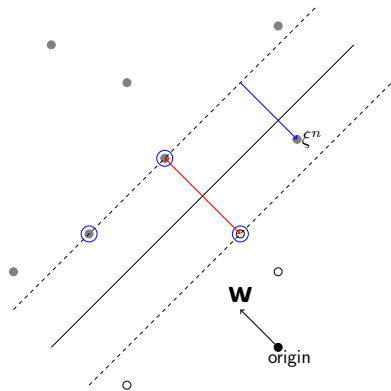
- Here each ξ^n measures how far \mathbf{x}^n is from the correct margin. For $0 < \xi^n < 1$ datapoint \mathbf{x}^n is on the correct side of the decision boundary. However for $\xi^n > 1$, the datapoint is assigned the opposite class to its training label.
- Ideally we want to limit the size of these 'violations' ξ^n .

The 2-norm soft-margin objective is

$$\text{minimise } \frac{1}{2} \mathbf{w}^\top \mathbf{w} + \frac{C}{2} \sum_n (\xi^n)^2 \quad \text{with } y^n (\mathbf{w}^\top \mathbf{x}^n + b) \geq 1 - \xi^n, \quad n = 1, \dots, N$$

where C controls the number of mislabellings of the data. The constant C needs to be determined empirically using a validation set.

2-Norm soft-margin



Slack margin. The term ξ^n measures how far a variable is from the correct side of the margin for its class. If $\xi^n > 1$ then the point will be misclassified and treated as an outlier.

Motivating Kernels

Consider a general objective of the form

$$E(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_n f(\mathbf{w}^T \mathbf{x}_n)$$

This has a minimum when \mathbf{w} lies in the span of the data $\mathbf{x}_1, \dots, \mathbf{x}_N$

$$\mathbf{w} = \sum_n f'(\mathbf{w}^T \mathbf{x}_n) \mathbf{x}_n$$

We can therefore assume $\mathbf{w} = \sum_n \alpha_n \mathbf{x}_n$ and form an equivalent objective

$$E(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{m,n} \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m - \sum_n f \left(\sum_m \alpha_m \mathbf{x}_m^T \mathbf{x}_n \right)$$

- This explains why $E(\mathbf{w})$ can be expressed as only a function of the scalar product between datapoints. An alternative is therefore to define the scalar product via a kernel function.
- The 'kernel' trick applies to any objective of the above form. In addition to the SVM, one can 'kernelise' linear regression, logistic regression, *etc.*

Using Kernels

- One can re-express the objective so that it depends on the inputs \mathbf{x}^n only via the scalar product $(\mathbf{x}^n)^\top \mathbf{x}^n$. If we map \mathbf{x} to a vector function of \mathbf{x} , then we can write

$$K(\mathbf{x}^n, \mathbf{x}^m) = \phi(\mathbf{x}^n)^\top \phi(\mathbf{x}^m)$$

- This means that we can use any positive semidefinite kernel K and make a non-linear classifier.
-

Kernels

- A kernel (or covariance function) k is a special function such that for a collection of vectors $\mathbf{x}^1, \dots, \mathbf{x}^N$ the matrix K with elements

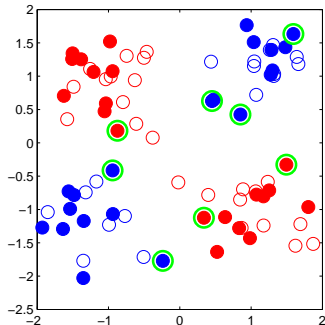
$$K_{mn} = k(\mathbf{x}^m, \mathbf{x}^n)$$

is positive semidefinite.

- The most well known is the 'Gaussian' or 'squared exponential'

$$k(\mathbf{x}^m, \mathbf{x}^n) = e^{-(\mathbf{x}^m - \mathbf{x}^n)^2}$$

SVM demo with a squared exponential kernel



- The solid red and solid blue circles represent data from different classes.
- The support vectors are highlighted in green.
- For the unfilled test points, the class assigned to them by the SVM is given by the colour.
- The computational complexity is $O(N^3)$ where N is the number of training points.
- `demoSVM.m`

Decision Trees

Consider data with the following attributes:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_2 \end{pmatrix} = \begin{pmatrix} \text{Salary} \\ \text{Permanent Job} \\ \text{Criminal Conviction} \end{pmatrix}$$

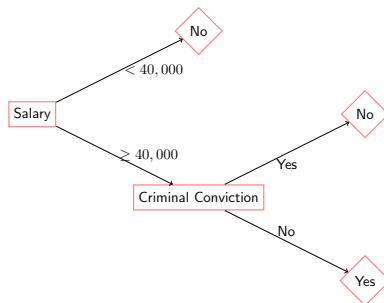
and that we have a training dataset in which bank managers have offered loans to

$$\begin{pmatrix} 100,000 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 50,000 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 75,000 \\ 1 \\ 0 \end{pmatrix}$$

and refused loans to

$$\begin{pmatrix} 50,000 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 30,000 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 10,000 \\ 1 \\ 0 \end{pmatrix}$$

Decision Trees



- The decision tree is learned such that the data associated with each node is as 'pure' (contains the same class labels) as possible.
- At each node, we consider all the variables and choose that one which splits the data associated to that node into as pure children as possible.
- Fast to train and interpretable ●
- Somewhat limited — decisions typically based on simple single variable splits ●

Ensemble Predictors

- Idea is to combine the predictions of several classifiers (works also for regression).
 - Many techniques based on different ways to construct predictors and also different ways to combine them.
 - Bagging is one of the simplest and most popular.
-

Bagging

- Bagging is a general 'ensemble' method.
- From our original input-output dataset, we generate B new datasets.
- Each dataset is generated by sampling (with replacement) data from the original dataset.
- For each new dataset we then train a model.
- The predictions from these B models are then combined, either by averaging or taking the majority prediction.

Random Forest

- We generate a set of decision trees and take the majority prediction of the trees.
 - Slightly different approaches to generating the trees.
 - Very popular in prediction competitions and often does surprisingly well.
-

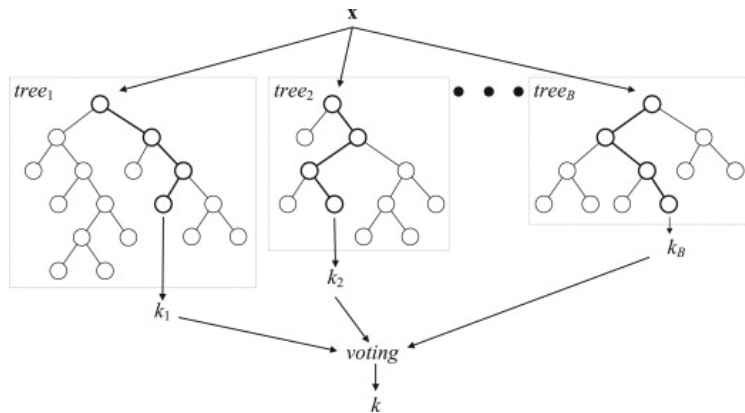
Ho's approach

- For each tree decide on a fixed random subset of the variables on which to form the tree.
 - Usually take around \sqrt{D} number of variables, where D is the total number of variables in the input vector.
-

Breiman's approach

- This is bagging applied to decision trees.
- We form the dataset for each tree using sampling with replacement.
- At each node in the tree we randomly select only a small number of the dimensions of the variable x on which to find the best split.

Random Forest



Summary

K-nearest neighbours

- Simple and intuitive ●
 - Slow to apply – typically scales $O(N)$ ●
 - Heavily dependent on sensible distance function ●
 - Can't easily deal with missing data ●
-

Logistic Regression

- Simple and fast to train ●
- Fast to apply – typically scales $O(D)$ ●
- Objective function is concave ●
- Can increase complexity by using kernels or non-linear mappings ●
- Can easily scale up to huge sparse data ●
- Can't easily deal with missing data ●

Summary

Support Vector Machine

- Has good generalisation performance ●
 - Quite fast to apply ●
 - Objective function is concave ●
 - Objective function is not differentiable, with $O(N^3)$ cost – does not scale well ●
 - Can't easily deal with missing data ●
-

Decision Tree

- Fast to train ●
- Highly interpretable ●
- Quite a weak classifier – decisions don't take simultaneous attributes into account.
- Random Forest extension has good empirical performance ●
- RF is less interpretable than the original DT ●
- Can't easily deal with missing data ●

Summary

Naive Bayes

- Fast to train and apply ●
- Can easily deal with missing data ●
- Classifier is quite weak ●

State of the art

Generative Approaches

- For $p(\mathbf{x}|c, \theta)$ we can use more powerful models than Naive Bayes.
- For example one can use class conditional (correlated) distributions such as the multivariate Gaussian.

Deep learning

- Neural Nets can be used to model for example the output probability for a binary classifier:

$$p(c|\mathbf{x}, \mathcal{W}) \propto e^{f_c(\mathbf{x}, \mathcal{W})}$$

where $f_c(\mathbf{x}, \mathcal{W})$ is the scalar output of a network for class c and \mathcal{W} is the weights of the network.

- One can then use maximum likelihood to train the parameters \mathcal{W} of this discriminative classifier.
- These can be very powerful classifiers, but are hard to train.