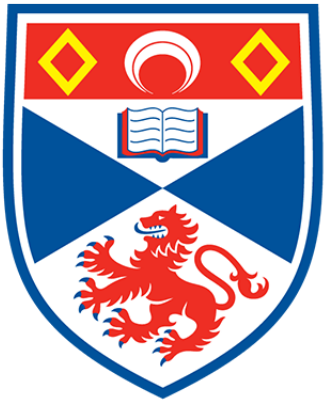


University of
St Andrews

CS5014 Machine Learning

Revision Notes

May 16, 2018

Lecturer:
David Harris-Birtill
Kasim Terzić

Submitted By:
140011146

Contents

1 Introduction

In machine learning, we wish to *learn* the mapping between input and output without having to program it explicitly.

> Maching learning is the field of study that gives computers the ability to learn without being explicitly programmed. - Arther Samuel.
>
> A computer program is said to learn from exerience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. - Tom Mitchell.

In general, most machine learning boils down to a simple prediction equation.

$$\hat{Y} = f(X, \theta) \tag{1}$$
$$Y = f(X, \theta) + \varepsilon \tag{2}$$

where f is a mathematical model for predicting Y from X. X is a list of input features that are used for prediction. They could be independent variables or some kind of predictor. $\hat{Y}$ is the output of the predicting model which are the predicted values. θ are the model parameters that

the model tries to learn and optimise for. Finally, ε is the error, which represents the difference between predicted values and desired values.

1.1 Prediction error

In order to measure the difference between the predicted values and actual values, the error for each data point must be measured or calculated in some way. To do this a **loss function** (cost function) is used to evaluate the quality of the model. The goal is to then try to minimise the loss function by altering the θ parameters to allow the model to make better predictions.

An example of a common loss function is the squared error loss.

$$L(\theta) = (Y - f(X, \theta))^2 \tag{3}$$

The loss function serves as a way to measure the quality of prediction for the set of parameters.

1.2 Machine learning process

1. Gather data
2. Prepare data (data cleaning and preprocessing)
3. Pick an algorithm (or a few different ones)
4. Pick a set of parameters
5. Evaluate performance of model
6. If the performance is not good, re-evaluate by altering parameters
7. Apply model to new/unseen data

This is generally implemented as an optimisation process, where the loss function is minimised on each iteration by changing the θ parameters. This strategy picks the optimal set of parameters. An example of such an optmisation process is gradient descent.

1.3 Supervised machine learning

The idea of supervised machine learning it to be able to predict an outcome based on given input data with tagged correct outputs. This could involve predicting a value or classifying an output class type. This requires a **labelled dataset** so it is known what the output is. There are two main types of supervised learning:

- Regression - The data predicted for regression is usually continuous and if it is decrete, it has to be rounded to the closest discrete value. Regression predicts the *value* of the data.
- Classification - The probability it is of some class. Classification predicts whether the data belongs to a certain class.

Classification uses a **decision boundary**, which everything on one side of the boundary is classified as one class and another on the other side of the boundary. It is possible to have hyper-dimensional decision boundaries for when there are more than 3 features which cannot be easily visualised.

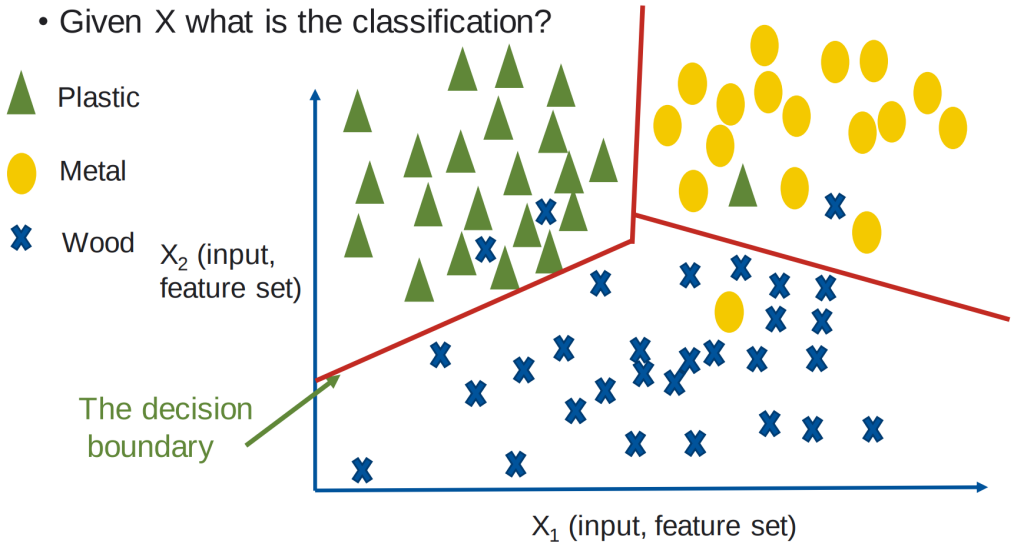


Figure 1: Classification of 3 classes using 2 input features. Having more than 3 input features (3D plot) is difficult to visualise.

One major issue with many machine learning algorithms: **overfitting**. Often the models trained become optimised too well towards the training data, leading to very low errors, but an inability to generalise to unseen data.

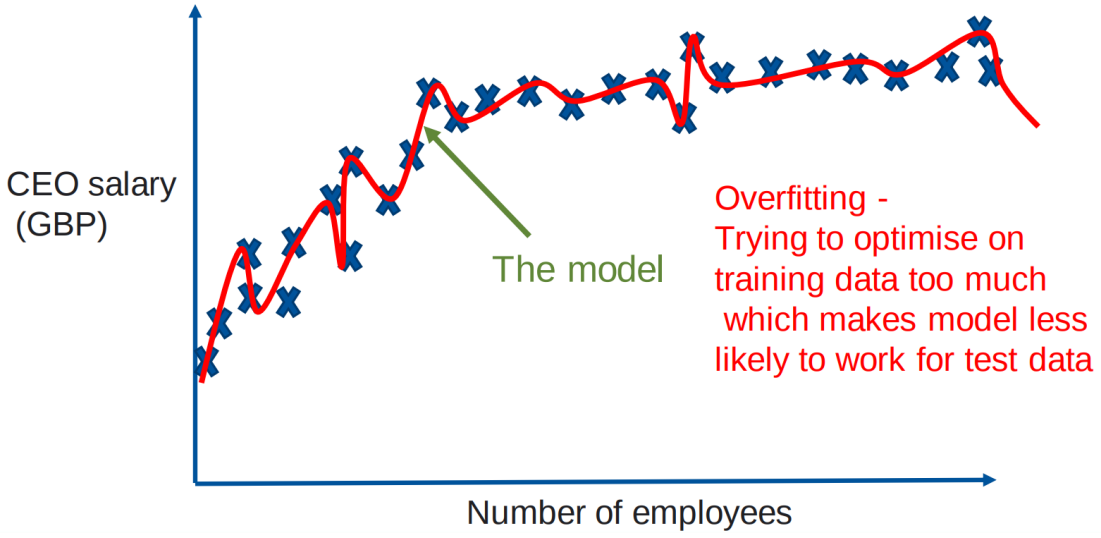


Figure 2: Example of overfitting in linear regression.

1.4 Unsupervised machine learning

In unsupervised learning, the dataset is **unlabelled**, and the output for predicted values is unknown. To learn, clusters are automatically created to separate the data into distinct groups. This is useful for discovering information where we don't know what we are looking for or to discover things we don't know about.

2 Linear regression

Linear regression models are simple but can provide a good and adequate descriptions of how the inputs affect the output. As linear regression only deals with linear models, it follows from the linear line equation $y = mx + c$ where the gradient m and intercept c are the θ parameters that the model tries to learn. This can be rewritten as:

$$f(X, \theta) = \theta_0 + \theta_1 X_1 \tag{4}$$

where θ_0 represents the c intercept and θ_1 represents the m gradient.

Furthermore, a loss function has to be defined to evaluate the quality of the fit without manual inspection. The goal of training is to reduce the error of the loss function. This is done by trying lots of different θ values and computing the error each time.

2.1 Gradient descent

Gradient descent is a method for parameter optimisation which automatically gets to the best model by minimising the lost function and automatically altering the θ parameters.

$$\theta_j := \theta_j - \alpha \frac{\delta}{\delta \theta_j} L(\theta_0, \theta_1) \tag{5}$$

where j = 0 and j = 1 update is done simultaneously. α here is the learning rate, which acts as a modifier to how much should be updated on each iteration. $\frac{\delta}{\delta \theta_j} L(\theta_0, \theta_1)$ is the derivative. The derivative of the loss function is used to see if we should increase or decrease the value of θ.

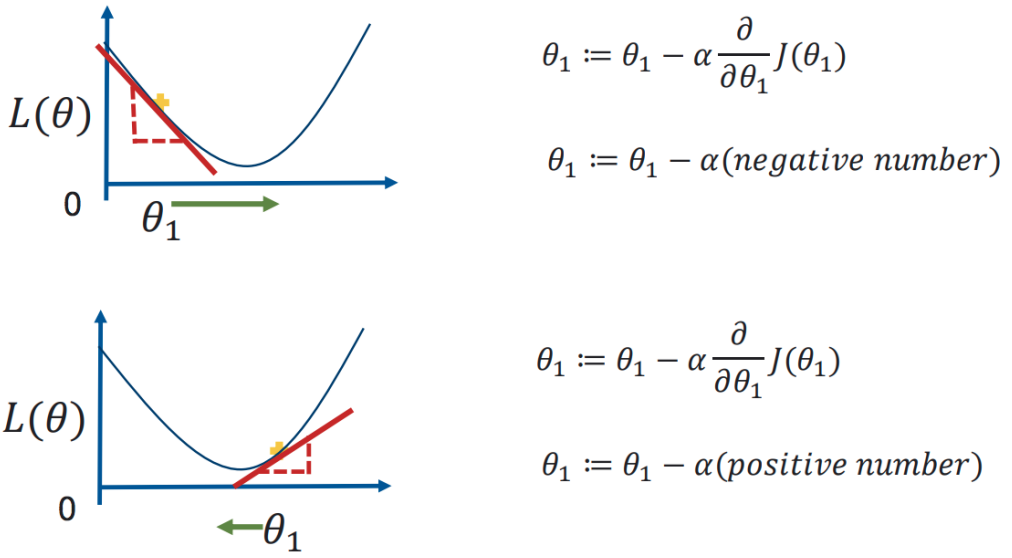


Figure 3: Gradient descent updating based on positive or negative gradient. For example, if the gradient is negative, we want to increase the θ values and recalculate the gradient.

There is an issue with gradient descent where it can get stuck in an local minimumm rather than the global minimum. There are a few ways of dealing with this:

- Try different initial values to start at different locations in the search space
- Add momentum to roll over local minima

A good way to find out what is happening with each training iteration is to plot the cost function at each iteration, then it can be seen if the cost is being minimised. The reduction of the cost function is controlled by the **learning rate** α. If α is too small, the gradient descent will be slow, requiring a large number of training iterations to minimise the cost and converge at the minimum. However, if the learning rate is too high, then it can overshoot the minimum and never converge. This effect can be seen by plotting the cost function and seeing if the graph converges to 0 or diverges.

2.2 Multivariate linear regression

Linear models can have many features and by extending to n features with n θ parameters. This extends the linear equation to be:

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ...\theta_n x_n \tag{6}$$

The input features X and parameters θ can be more easily represented with matrix form:

$$X = \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix} \tag{7}$$

Using matrix operations, all θ parameters can be updated simultaneously on every step rather than separately.

2.3 Feature scaling

When there are many input features, they can come from different scales. For example, one feature may have the range 0-1 while another has values ranging from 100-1000. Because of these different scales, the larger values may dominate the model even if both features are equally important. To deal with this issues, all features can be scaled to the same range through normalisation.

$$x_i = \frac{x_i - \mu_i}{\text{range}(x_i)} \tag{8}$$

One method of feature scaling is mean normalisation, where each input feature x_i is replaced with $x_i - \mu_i$ to make features with a mean of approximately zero.

2.4 Normal equation

$$\theta = (X^T X)^{-1} X^T y \tag{9}$$

where X^T is the transpose of X and $(X^T X)^{-1}$ is the inverse of $X^T X$. Feature scaling is not needed for the normal equation because each θ parameter is proportional to the input feature it is calculated from. X and y here are matrices of the inputs and outputs. With the normal equation, the θ parameters can be calculated analytically rather than require the many training iterations. It is sometimes preferable to use the normal equation over optimisation like gradient descent as there is not need for computation iterations and choice of learning rate. However, the normal equation is slow to calculate if the number of input features is very large, which gradient descent deals with well.

Normal equation

Pros:
- No need for iteration
- Don't need to choose α

Cons:
- Slow if number of input features is very large

Gradient descent

Cons:
- Needs many iterations
- Need to choose learning rate α

Pros:
- Works well when the number of features is large

3 Logistic regression

Logistic regression is a machine learning model to classify input data into output classes. A typical example of using logistic regression is to classify cancer or not cancer. We cannot use linear regression for classification tasks as it does not fit the data well and therefore is not well suited. In classification, a **sigmoid function** is used as it gives values between 0 and 1, like a switch between binary values. Coincidentally, sigmoid functions are also used as activation functions in neural networks.

The sigmoid function is defined as follows:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{10}$$

where $z = \theta^T x$. With classification, the output is typically a probability that of a certain class $0 \leq h(x) \leq 1$. The function is therefore:

$$h(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{11}$$

where $h(x)$ is the estimated probability that $y = 1$ on input x. In an example of multiple input features, we get get multiple θ values. For example, given blood and urine test values, does the patient have cancer:

$$h(x) = g(\theta^T x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2) = \frac{1}{1 + e^{-\theta^T x}} \tag{12}$$

where x_1 is the blood test values and x_2 is the urine test values. $h(x)$ is the probability that the patient has cancer.

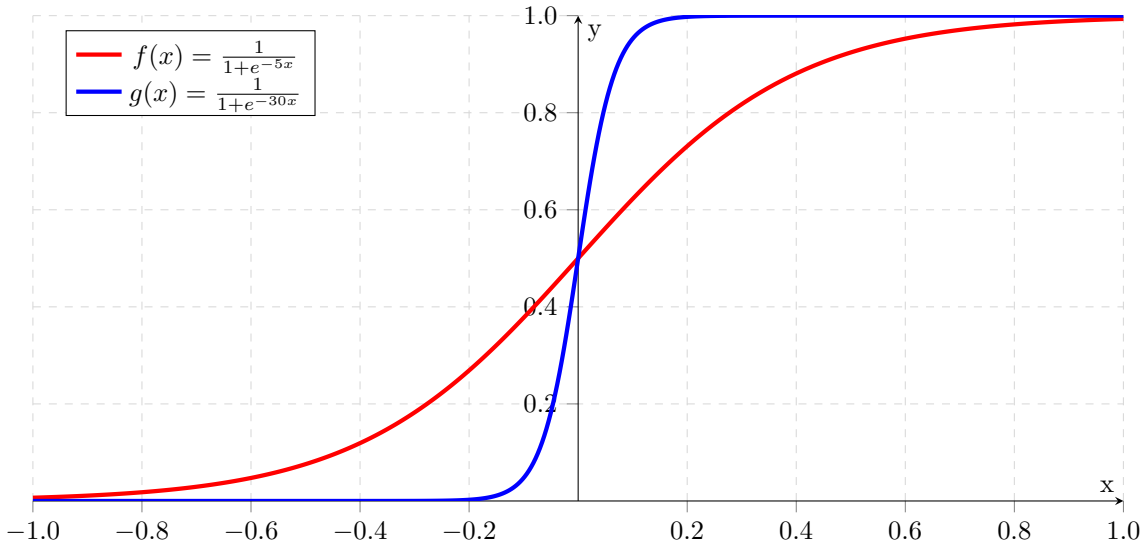


Figure 4: Example of a sigmoid function. The parameters θ^T determine the steepness.

For some classification problems, a linear logistic regression model is not enough. This happens when a linear decision boundary is not enough to distinguish between the different classes. In these cases, a non-linear function is needed.

3.1 Cost function

In logistic regression, the same squared error cost used in linear regression cannot be used because it doesn't make much sense. A cost function such as the squared error measures the *distance* between the predict value and actual value, however, this distance is not easily identified for a classification problem, where the actual value is which class the data belongs in.

$$Cost(h(x), y) = \begin{cases} -log(h(x)) & \text{if } y = 1 \\ -log(1 - h(x)) & \text{if } y = 0 \end{cases} \tag{13}$$

$$Cost(h(x), y) = -ylog(h(x)) - (1 - y)log(1 - h(x)) \tag{14}$$

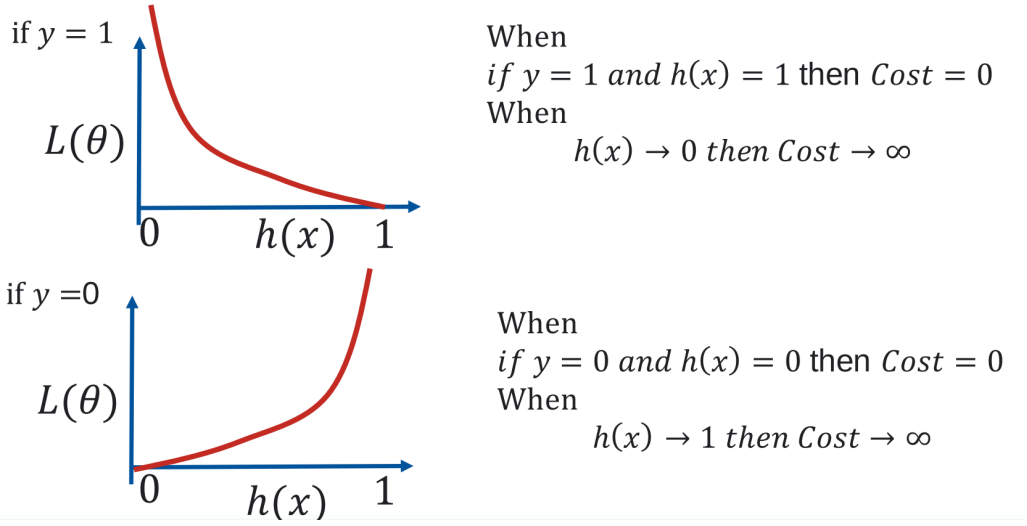


Figure 5: Cost function for logistic regression.

Log functions are used as the cost function so that the cost for a prediction of $h(x) \approx 1$ tends to 0 and a prediction of $h(x) \approx 0$ tends to ∞. This this cost function, logistic regression can also use gradient descent to tweak θ parameters.

3.2 Multiclass classification

There are often cases in classification where it is not simply a binary classification such as cancer or not cancer. This is multiclass classification where there are multiple classes, such as classifying the colour based on wavelength. Here, simply applying the previous techniques and cost functions is not enough as it will only give the probability of one certain class. To deal with multiclass problems, the typical solution is to use a **one-vs-all** method where each class is treated separately as a binary classification task. There are three main steps for doing this:

- Choose one class apply binary classification to all data points
- Repeat for all other classes
- Generate probability for each class based on the binary classification and pick the classification with the highest probability

4 Regularisation

When using too many features, it is often the case that overfitting happens where the model fits well to the training data, but does not generalise to unseen data. A typical way to deal with overfitting is to reduce the number of features, either manually or automatically. This is done because some input features may not be very relevant to the general problem, but still affect the prediction.

Regularisation is another method to reduce the effect of some input features without removing them completely. It works by keeping all input features, but using multipliers to reduce the scale of certain θ parameters. This allows more relevant features to have more impact while still keeping a lower effect of less relevant features. The parameters are penalised using a large multiplier to cause them to be minimised because of the increased constant. For example $... + 1000\theta_3 + 1000\theta_4$. This changes the cost function by adding a regularisation parameter λ.

$$\text{Cost} = \frac{1}{2m}\left[\sum_{i=0}^{m}(h(x^i) - y^i)^2 + \lambda\sum_{j=1}^{n}\theta_j\right] \tag{15}$$

The regularisation parameter λ determines how much to suppress the θ parameters. If λ is too large, then we may underfit as we have minimised the θ parameters too much. If λ is too small, then we may continue to overfit as it does not do enough to reduce the effect of θ parameters.

The regularisation parameter can also be added to the normal equation.

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\right)^{-1} X^T y \tag{16}$$

In general, regularisation can be thought of as adding an extra component to the equation of minimising the cost function.

$$J(\theta) = L(\theta) + R(\theta) \tag{17}$$

where $J(\theta$ is the cost, $L(\theta)$ is the loss and $R(\theta)$ is the regularisation. Further, regularisation has a computational advantage over feature selection, as additional training and searching of models is not needed.

4.1 Ridge regression

The $\lambda\sum_{j=1}^{n}\theta_j$ term for regularisation can be calculated using the l_2 norm, usually denoted as $||\theta||_2^2$. The l_2 norm is the euclidean norm, which is a sum of squares of all elements, defined as:

$$||\theta||_2 = \sqrt{\sum_{j=1}^{n}\theta_j^2} \tag{18}$$

It measures the distance of θ from zero. As λ increases, the l_2 norm will *always* decrease. The reason regularisation has an advantage of a simple least squares regression is based on the **bias-variance trade-off**. As λ increases, the flexibility of the model decreases, leading to decreased variance, but increased bias. For some values of λ up to a limit, the variance will decrease rapidly with only a small increase in bias, which reduces the overall mean squared error of the model.

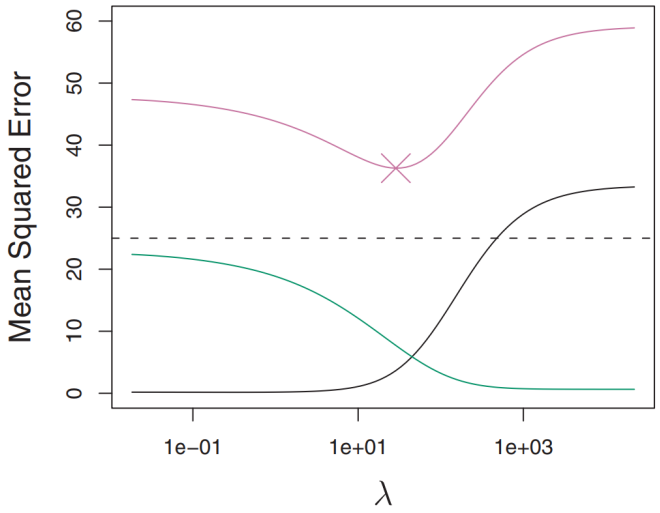


Figure 6: Plot showing why ridge regression leads to lower MSE for some values of λ. Squared bias (black), variance (green) and test MSE (purple) for the ridge regression predictions on a simulated dataset as a function of λ. Taken from *An Introduction to Statistical Learning*.

4.2 Lasso regression

The disadvantage of ridge regression is that while it will shrink all coefficients towards zero, it will not set them to be exactly zero unless $\lambda = \infty$. In other words, for datasets with a large number of features, increasing *lambda* will reduce their effect but not exclude any of them completely.

Lasso regression works the same way as ridge regression, except instead of using the l_2 norm, the l_1 norm is used. Rather than being a sum of squares, the l_1 norm is simply a sum of all elements.

$$||\theta||_1 = \sum_{j=1}^{n} |\theta_j| \tag{19}$$

Lasso regression also shrinks the coefficient estimates towards zero. However, the l_1 penalty is able to force some coefficient estimates to be equal to zero when the *lambda* value is large enough. In this sense, lasso regression performs variable selection like selecting the best features. It can be said that lasso regression yields *sparse* models because the models may only involve a smaller subset of features.

5 Evaluation

The goal of supervised machine learning is to train the models on known data in order to predict new data. To be able to evaluate how well the model works on new data, any given data has to be split into separate training and test data sets. The testing data is used as a final step after training models to evaluate how well they perform. Testing data should *never* be used to train, as otherwise this would bias the model heavily towards the testing data, rather than towards general unseen data. Further, testing data should not be used for any step of the machine learning process except for checking the performance of trained models at the very end.

5.1 Source of errors

- Not enough data which leads to overfitting to available data and inability to generalise

- Fine tuning parameters on test data which still causes overfitting because the model is designed to work well only on available data
- Training or making decisions based on testing data could lead to very low error rates on available data, but the heavy bias is undesirable

Most often, the source of errors in machine learning comes from overfitting in some way, either from looking and training towards the test data, or from not enough data or too many input features.

5.2 Classification metrics

Classification differs from regression in that all errors are treated the same. It is not as easy as using functions such as the root mean squared error or R^2 coefficient of determination. Classification are either correct or incorrect, which makes errors simpler as there is no need to calculate how far the predicted value is.

- Correct classification = true positives + true negatives
- Incorrect classification = false positives + false negatives

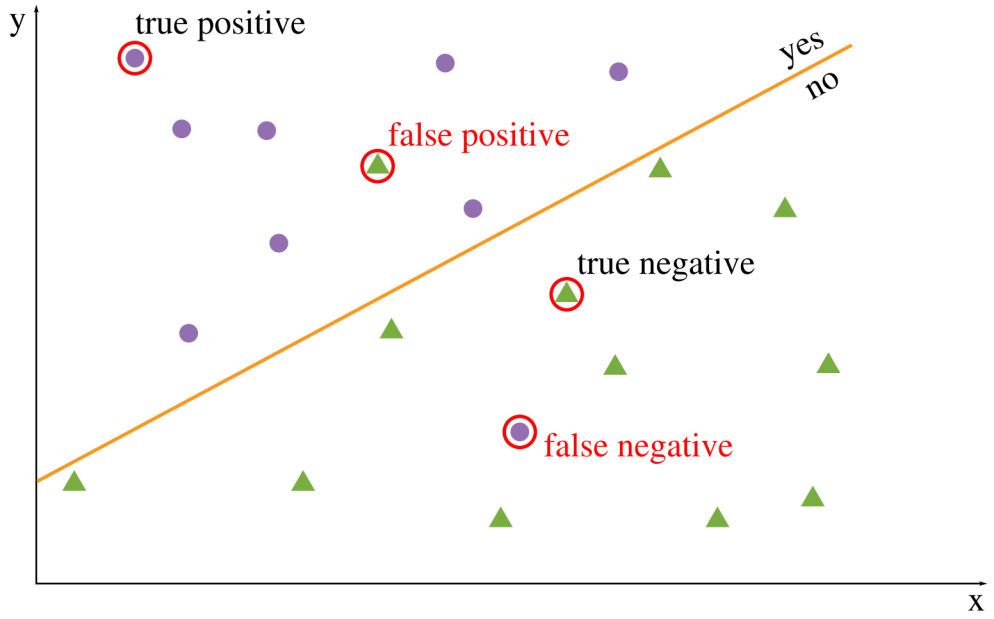


Figure 7: Examples of the different types of correct/incorrect classification

With different combinations of true/false positives/negatives, different metrics can be calculated for classification problems. For example, the accuracy (or classification rate) is defined as:

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{total positives} + \text{total negatives}} \tag{20}$$

There are many further metrics that can be used for classification, most notably are **precision** and **recall**.

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{21}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{22}$$

The precision is the fraction of correct classifications among all classified data points and the recall is the fraction of correct classifications among all relevant data points. In other words, the precision tells us how much of what was classified is correctly classified, while the recall tells us how much of the positive classifications were found out of the total positive data points. The two scores represent a trade-off as to how correct the classifications are against how many relevant data points were found. This trade-off depends on the actual classification problem. For example, in the case of cancer identification, it may be better to bias towards false positives (higher recall, lower precision) to be on the safe side.

The trade-off between precision and recall can often be visualised by varying a threshold on the classification and plotting a precision-recall curve.

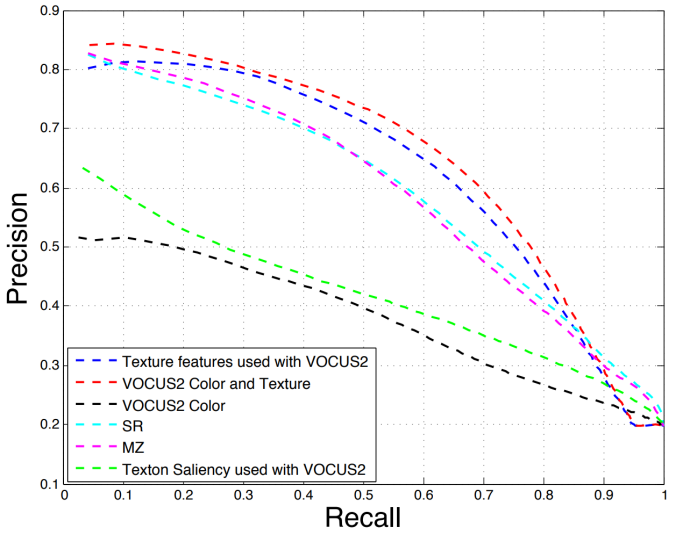


Figure 8: Precision-recall curves.

It is often difficult to compare precision-recall curves as different curves could represent different trade-offs. It is more useful to have a single number to compare, so the area under the curve is used as an alternative. The larger the area, the better the performance. This is because a good model that achieves both high precision and recall will have a larger area under the curve. This area is the **average precision** for one binary classification task averaged over all possible parameter values. With multiple classes, a precision-recall curve would be needed for each class in the one-vs-all method. To better visualise this for an average over all classes, a **confusion matrix** can be used.

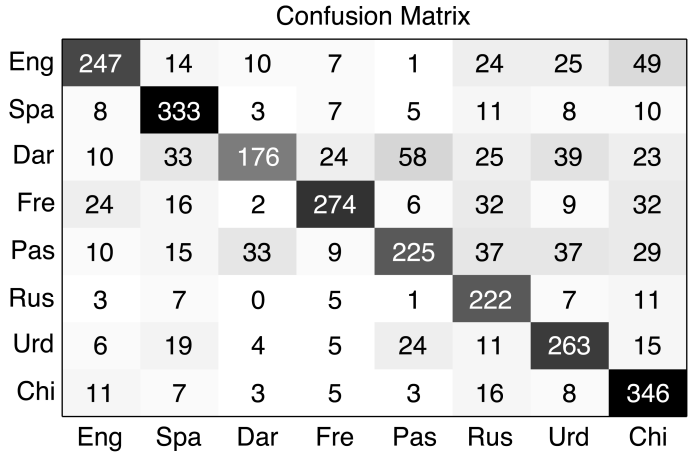
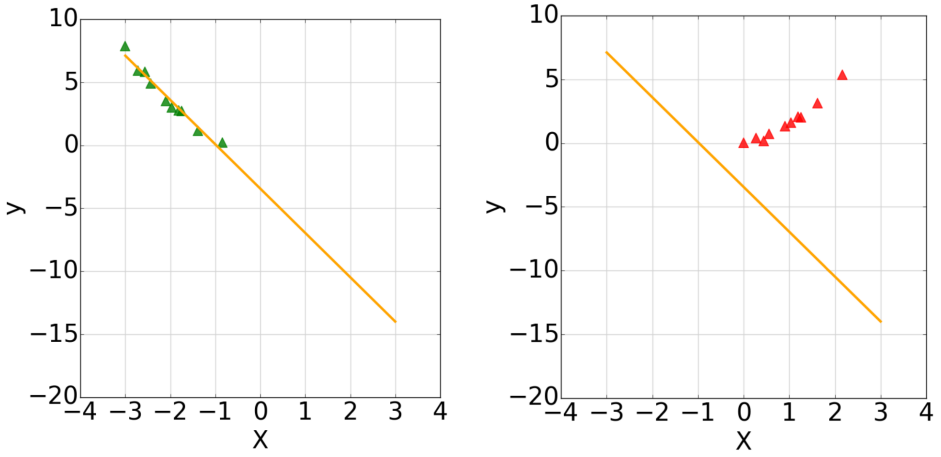


Figure 9: Example of a confusion matrix.

The diagonal represents correct detections for that class. The rows and columns represent false positives and false negatives, as they are predicted the wrong class. Confusion matrices are useful to see which classes are better predicted compared to other classes

5.3 Validation

Because testing data is sometimes completely hidden, the given training data may need to be split into a validation set, which is used to evaluate the quality of the model. There is an issue during the splitting phase of training and validation data where the two sets are not good representation samples from the dataset. For example, if the dataset is split in half exactly, many important features may be missed.



(a) Prediction on training data

(b) Prediction on validation data

Figure 10: Extreme example of an issue that can occur in splitting training/validation data.

The split training/validation datasets have to be representative samples from the domain of the entire dataset. However, this can be difficult to do without looking at the dataset, which compromises the experiment. A method to deal with this is called **stratified sampling**. In classification, this would ensure a representative number of classes are in each dataset. For regression problems, certain important features need to be used as the feature for stratification, for example gender proportions in data about people.

Another issue with splitting the data into a validation set is that it reduces the amount of data available for training. Further, randomly splitting the dataset, even with stratified sampling could lead to getting "easier" or "harder" datasets to work with. Another method to evaluate the performance of trained models is **k-fold cross validation**.

5.3.1 k-fold cross validation

The idea behind k-fold cross validation is to split the dataset into k equally sized subsets and run training k times, leaving out one of the subsets for testing. Each k samples is used for validation exactly once. Larger values of k means more splitting up of the data, which gives more reliable results, but requires more computation.

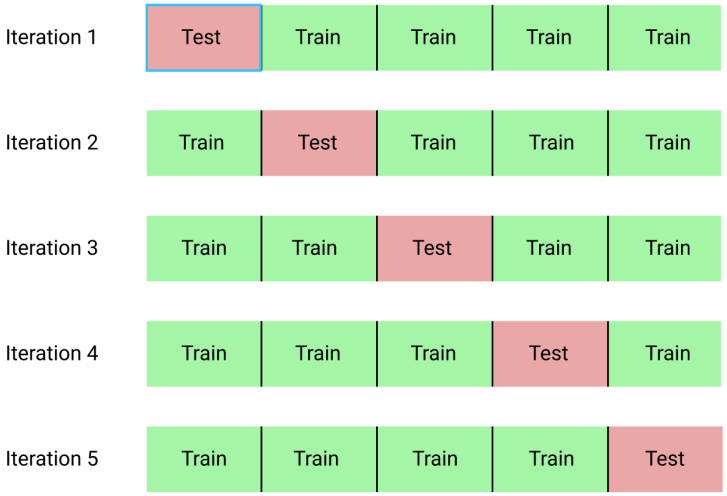


Figure 11: Example of 5-fold cross validation.

6 Data preparation

Often times the raw data received at the start of the machine learning process requires cleaning or preprocessing. This can be due to errors during data gathering, for example negative readings that aren't possible, or the way the data is presented, such as categorical values like colour (red, blue, green) or time. Data such as this has to be processed to be able to be passed to any machine learning algorithm, as we need numerical inputs in some cases.

Example data preparation process:

- Clean data
- Convert data types (categorical with dummy variables)
- Find mean and standard deviation of features
- Normalise data
- Select a subset of features
- Split into training and test/validation set
- Train algorithm
- Evaluate on test set

6.1 Categorical inputs

Make	Horsepower	Color	Accidents
Aubi TT	220	Red	13
Citreom C4	160	Blue	1
Fauxhall Corso	98	Red	2
Citreom C4	124	Black	0
Aubi TT	196	Red	7

Table 1: Example data with categorical inputs (colour)

Some algorithms such as decision trees and random forests can deal with categorical data, but for models such as linear regression, numerical data is required. There are two main methods of transforming categorical inputs into numeric inputs: **enumeration** and **dummy variables**.

6.1.1 Enumeration

To encode categories by enumeration, each class of the category can be assigned a number, for example assigning red = 1, blue = 2 etc. This is a very simple way to transform categories to numbers but creates a new issue. The distance between the numbers subtly implies a similarity between the values. For example if red = 1, blue = 2 and black = 3, we are implying that blue is similar to both red and black while red and black are not as similar to each other. Sometimes this can be desirable if encoded correctly to give more detailed information, such as if the colour numbers were ordered in a way that represented the difference.

Make	Horsepower	Color	Accidents
1	220	1	13
2	160	2	1
3	98	1	2
2	124	3	0
1	196	1	7

Table 2: Transformed categorical make and colour data using enumeration

6.1.2 Dummy variables

Instead of enumerating all the categorical values, a vector of ones and zeroes is used where each value corresponds to each possible category and the 1 or 0 represent true and false.

TT	C4	Corso	Horsepower	Red	Blue	Black	Accidents
1	0	0	220	1	0	0	13
0	1	0	160	0	1	0	1
0	0	1	98	1	0	0	2
0	1	0	124	0	0	1	0
1	0	0	196	1	0	0	7

Table 3: Transformed categorical make and colour data using enumeration

6.2 Input cleaning

When data is collected, there are often cases where mistakes or other errors appear and have to be cleaned beforehand. For example, all columns having the same data type and all rows having

the same number of columns. Further errors can be dealt with in two ways, either try to fix them, for example inserting an average value to replace a NaN value, or remove the row entirely. The latter case may not be desirable in cases with little data. Other examples include spelling errors, negative values, faulty sensors and infinity values.

6.3 Feature selection

It was mentioned before that a reduced feature set can help reduce overfitting in the case of regularisation. In some cases, some features should be discarded entirely, especially if they are not relevant. There are a few good reasons for selecting only relevant features to use:

- It can improve accuracy with some models and reduce overfitting
- It can improve interpret-ability, as it is easier to understand models based on fewer parameters
- It can make infeasible problems feasible by significantly reducing the computational requirements

Feature selection can be done via **univariate tests** where the correlation between each features and the output is calculated and the good well correlated features are selected. Further, correlation between feature pairs can be calculated, as features that are well correlated to each other may be redundant.

There are also automatic algorithms for feature selection to select the k best features which give the best performance. Most notable, forward and backward stepwise selection.

6.3.1 Stepwise selection

In stepwise selection, features are either incrementally added or removed until the best number of features is found. In forward stepwise selection, each feature is added incrementally, trained on a new model and evaluated on validation or testing data. The best model is kept and another feature is added. This is repeated until the models no longer improve in performance and the best feature subset is selection.

Backwards stepwise selection works the other way by starting with all input features and removing features which affect the model the least.

6.4 Data augmentation

Data augmentation is a good technique both to increase size of the dataset and help models better generalise to new data. This is especially good for complex and modern classifiers, which may only work in very specific cases without data augmentation and need very large amounts of data. Augmentation can be done by transforming existing data in some way. For example shift, rotate, scale, distort, flip, change contrast in images. Domain knowledge is useful here to know how to augment the data.

Figure 12: Example of data augmentation on images of a stop sign.

7 Basis expansion

7.1 Polynomial regression

To extend linear regression to settings where the function is non-linear, basis functions are used to construct a larger space of functions. A basis function (or number of basis functions) are defined on the input X. For example

$$h_1(X) = 1, h_2(X) = X, h_3(X) = X^2 \tag{23}$$

forms three basis functions which is used to transform the input. This transformation replaces the standard linear model

$$f(X) = \theta_0 + \theta_1 X \tag{24}$$

with new inputs

$$[h_1(X), h_2(X), h_3(X)]^T \rightarrow [1, X, X^2]^T \tag{25}$$

to get a polynomial function

$$f(X) = \theta_0 h_1(X) + \theta_1 h_2(X) + \theta_2 h_3(X) \tag{26}$$

$$= \theta_0 + \theta_1 X + \theta_2 X^2 \tag{27}$$

Now the same linear regression can be run on the new input as the output is a linear combination of functions, which a linear regression model can still be applied on. The functions may be non-linear, but the input space has been transformed through the non-linear mapping. Of course, this can naturally be extended to m polynomials, but generally using more than 3 or 4 polynomials leads to high chances of overfitting.

7.2 Piecewise linear regression

Basis expansions are in fact more general than polynomial expansion. Basis functions can also be defined to create a piecewise linear regression model.

7.3 Regression splines

A spline is a piecewise *polynomial* that uses $m + k$ basis functions

$$h_i(X) = X^{i-1}, \forall\ i = 1, 2, ..., mh_{m+j}(X) \qquad = (X - \xi_j)_+^{m-1}, \forall\ j = 1, 2, ..., k \tag{28}$$

where m is th order of the spline + 1, for example a cubic spline is order 4 and k is the number of knots in the piecewise.

The first set of basis functions $h_i(X)$ are the normal basis functions for a polynomial expansion while the second set of basis functions $h_{m+j}(X)$ determine the knots. In particular, ξ_j determines the location of the jth knot.

8 Bayesian classification

In Bayesian terms, input variables are called **evidence**. The Bayes rule allows the probability and evidence to be expressed into simpler distributions:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)} \tag{29}$$

where

- $P(C_i|X)$ is the **posterior probability**. In other words, if we know X happened, how likely is C?
- $P(X|C_i)$ is the **likelihood**. If we know that the right answer is C, how likely is observation X?
- $P(C_i)$ is the **prior**, which is how often C_i is the correct answer in total. This can often be measured or estimated.
- $P(X)$ is the **probability of evidence**, which is how often this particular observation is obtained.

Many classifiers attempt to calculate the posterior directly, for example logistic regression. The underlying assumption is that the priors will not change.

8.1 Maximum a Posteriori classification

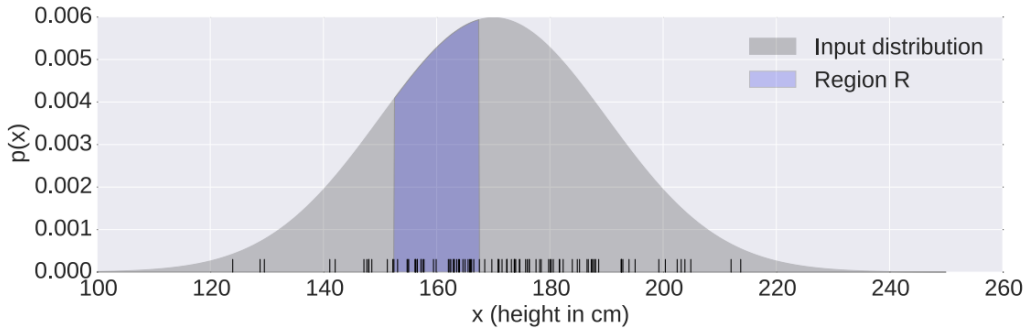
The maximum likelihood classifier picks the class most likely to generate X. It only makes sense when the classes are approximately equally likely.

$$\hat{Y} = \text{argmax}_c P(X|C) \tag{30}$$

Next, a maximum a posteriori (MAP) classifier can be defined which maximises the posterior

$$\hat{Y} = \text{argmac}_c P(X|C)P(C) \tag{31}$$

This allows for easy integration of priors, which can adapt to changing conditions. The probabilities are typically found be the integral of the probability density function.



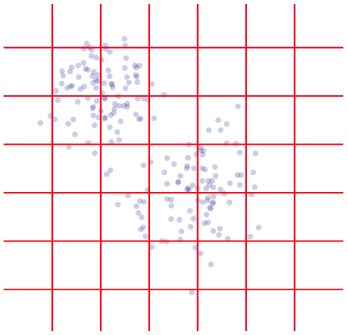
$$P_R = \int_R P(X)dX \tag{32}$$

Of course, this only applies to distribution of one dimension, where the distribution can be easily calculated and visualised. In distributions of higher dimensions, the probability must be estimated using the local neighbourhood. The idea behind this is **local density estimation**, where the probability density as position X from the number of samples K inside a volume V

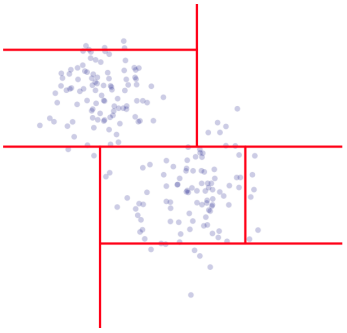
$$X = [X_1, X_2, ..., X_P]^T \tag{33}$$

$$p(X) = \frac{K}{NV} \tag{34}$$

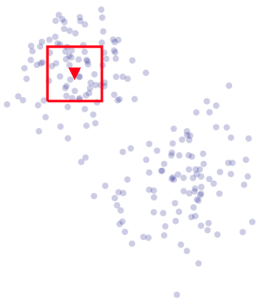
where P is the number of dimensions and N is the number of samples. The assumption is that density is roughly constant locally. With these assumptions, there are several methods based on this principle, which differ in how they set K and V and how they divide the feature space into subvolumes.



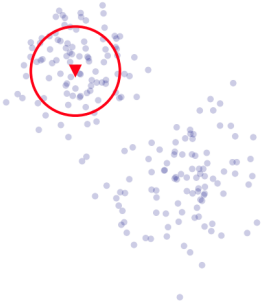
(a) Histogram



(b) Decision tree



(c) Tophat kernel



(d) kNN

8.1.1 Histograms

A histogram estimation imposes a strict grid on the feature space, which fixes V while allowing K to vary. This method is very sensitive to the bin size (size of the defined grid) and quite crude.

8.1.2 Parzen density estimation

The idea here is to represent the distribution as a sum of statistical **kernels**.

$$p(X) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{V}k(\frac{X - x_N}{h}) \tag{35}$$

where h is the bandwidth parameter. The simple example is the **tophat kernel**

$$k(U) = \begin{cases} 1 & |U_i| \leq \frac{1}{2}, i = 1, 2, ..., P \\ 0 & \text{otherwise} \end{cases} \tag{36}$$

The distance to each previously seen sample needs to be calculated, which is an expensive computation when there is a lot of data. The advantage gained is no training stage is needed, and there are some efficient data structures which help speed up this process.

8.1.3 k-Nearest Neighbours

The kNN algorithm chooses a K value to keep constraint while growing the volume V around the data point x as needed:

$$p(X) = \frac{K}{NV} \tag{37}$$

The conditional probabilities by counting data points is generated by each class C_i

$$p(X|C_i) = \frac{K_i}{N_i V} \tag{38}$$

and after applying Bayes' theorem:

$$p(C_i|X) = \frac{p(X|C_i)P(C_i)}{p(X)} = \frac{K_i}{K} \tag{39}$$

8.1.4 Bandwith parameter

The bandwidth parameter h in the density estimation equations can be varied to control the amount of smoothing. Small values of h will increase the variance (overfit the data) while large values of h will increase the bias (underfit the data). Typically the choice of h is not obvious and it must be found via cross-validation.

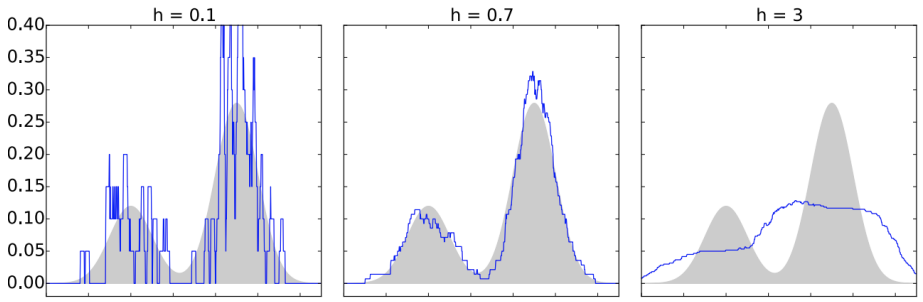


Figure 14: The influence of the bandwidth h.

9 Support vector machines

9.1 Maximal margin classifer

Recall in logistic regression, there is a decision boundary where data points on one side are classified as a class, and data points on the other side are classified another class. For a problem in n-dimensional space, a $n-1$-dimensional hyperplane is defined as the decision boundary. For example in a two-dimensional (x,y) space, the decision boundary is a one-dimensional hyperspace, in other words, a line.

The definition of a hyperplane is quite simple, for example a two-dimensional hyperplane is defined by the equation

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \tag{40}$$

for the parameters β_0, β_1 and β_2. Any point X in which the equation holds is a point that is on the hyperplane. Of course, the definition is easily extended to a n-dimensions:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n = 0 \tag{41}$$

$$\beta_0 + \sum_{i=1}^{n} \beta_i X_i = 0 \tag{42}$$

Any point $X = (X1, X2, ..., X_p)^T$ that satisfies the equation for the hyperplane lies on the hyperplane. Suppose a point X does not satisfy the equation, and instead,

$$\beta_0 = \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n > 0 \tag{43}$$

then X lies on one side of the hyperplane. Conversely, if

$$\beta_0 = \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n < 0 \tag{44}$$

then X lies on the other side of the hyperplane.

This makes a lot of sense for classification problems, as the hyperplane divides the dataset into two halves. It is then also easy to calculate which side of the hyperplane a point lies by calculating the left hand side.

9.1.1 Classification with a hyperplane

With a training dataset with data points $x_1, x_2, ..., x_n$ that falls into two classes specified by $y_1, y_2, ..., y_n$. It is observed that the output y variables have the range $\{-1, 1\}$ where -1 represents one class and 1 the other class. Assuming that it is possible to construct a hyperplane that separates the training data points perfectly into the two classes.

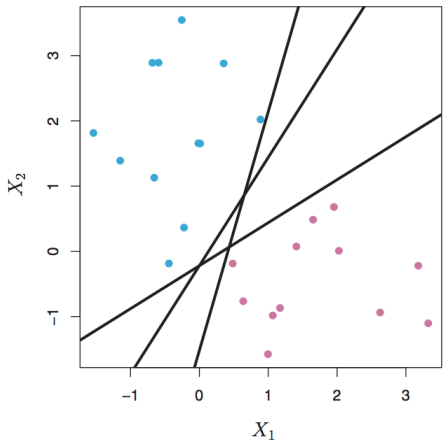


Figure 15: Example of three separating hyperplanes, out of many possible to perfectly split the blue and purple data points.

This gives the separating hyperplane has the property that

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_n x_{in} > 0 \text{ if } y_i = 1, \tag{45}$$

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_n x_{in} < 0 \text{ if } y_i = -1 \tag{46}$$

for all $i = 1, 2, ..., n$. In fact, there are infinitely many such hyperplanes that can exist by tweaking the plane slightly up or down, or rotating it slightly without touching any existing data points.

9.1.2 The maximal margin classifier

In order to create a classifier based on a separating hyperplane, only one hyerplane should be chosen out of the infinitely possible hyperplanes. A natural choice is the **maximal margin hyperplane**, which chooses the separating hyperplane that is the farthest away from the training data points. The distance between each training data to a given separating hyperplane can be calculated; the smallest such distance from each class is the minimal distance from the hyperplane and is known as the **margin**. The maximal margin hyperplane is defined as the separating hyperplane for which the margin is largest, in other words, the hyperplane that has the farthest minimum distance to the training data points. New test data can then be classified based on which side of the hyperplane they lie on. This is a good choice because the large margin means it takes a bigger difference "jump" to cross the boundary and lead to a misclassification.

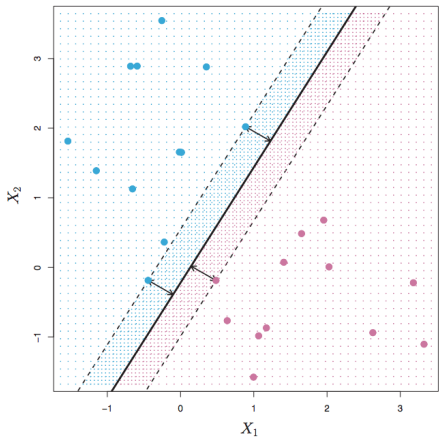


Figure 16: The maximal margin hyperplane shown which separates the blue and purple data. The three points on the dashed lines that exactly lie at the width of the margin are the support vectors.

Any points that lie exactly on the width of the margin are known as **support vectors** as they support the maximal margin hyperplane because if they moved, then the maximal margin hyperplane would move as well. The chosen hyperplane therefore depends directly on these support vectors and not on any other data points unless they move past the margin.

9.1.3 Maximal margin optimisation problem

To actually construct the maximal margin hyperplane from the training examples is to find the solution to the following optimisation problem:

$$\text{maximise} M \tag{47}$$

$$\text{subject to } \sum_{j=1}^{n} \beta_j^2 = 1, \tag{48}$$

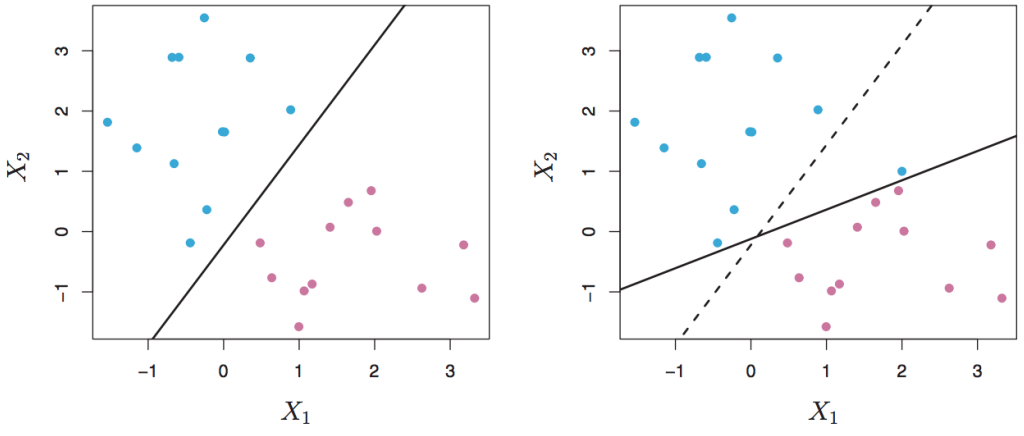
$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_n x_{in}) \geq M \ \forall i = 1, 2, ..., n \tag{49}$$

The third constraint requires that each training data point be on the correct side of the hyperplane. The second constraint

$$\text{subject to } \sum_{j=1}^{n} \beta_j^2 = 1$$

in combination with the third constraint ensures that all training points are on the correct side of the hyperplane *and* are at least a distance M away from the hyperplane.

Of course this would not work for more complicated datasets that are not easily separable, as this only works with the assumption that *a separating hyperplane exists*. If a separating hyperplane does not exist, then the maximal margin classifier cannot be constructed. Furthermore, there is an issue of maximal margin classifiers too easily overfitting to the training data as it is required that the maximal margin hyperplane perfectly split the two sets of data.



IGURE 9.5. Left: *Two classes of observations are shown in blue and i*

Figure 17: The addition of another data point on the right leads to a dramatic shift of the maximal margin hyperplane in order to keep the perfect split between the two sets of data.

9.2 Support vector classifiers

From the observation that a maximal margin classifier does not work for non-separable data and further is highly sensitive to changes in data, the next step is to define a classifier that uses the same concepts of margins, but allows for a hyperplane that does *not* perfectly separate the data. There are two advantages in this approach:

1. Greater robustness to individual data points
2. Better classification of *most* training data points

In other words, it may be worth misclassifying some samples to increase the generality of the model. This is known as the **support vector classifier**, or the **soft margin classifier**. Here, training points are allowed to be on the wrong side of the margin, and even on the wrong side of the hyperplane.

9.2.1 Support vector optimisation problem

The optimisation problem from the maximal margin classifier is altered to allow for incorrect classifications. To do so, new slack variables ϵ are introduced into the optimisation problem

$$\text{maximise } M \tag{50}$$

$$\text{subject to } \sum_{j=1}^{n} \beta_j^2 = 1, \tag{51}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_n x_{in}) \geq M(1 - \epsilon_i) \tag{52}$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C \tag{53}$$

where C is a non-negative tuning parameter. $\epsilon_1, ..., \epsilon_n$ are the slack variables that allow individual observations to be on the wrong side of the margin or the hyperplane. The slack variable ϵ_i tells us where the ith observation is located relative to the hyperplane and margin.

- If $\epsilon_i = 0$, then the data point is on the correct side of the margin
- If $\epsilon_i > 0$, then the data point is on the wrong side of the margin
- If $\epsilon_i > 1$, then the data point is on the wrong side of the hyperplane

The tuning parameter C is important as it bounds the sum of the slack variables. It controls the number and severity of the violations to the margin that are allowed. In other words, it is the amount of tolence that the margin can be violated by n observations. For example, if $C = 0$ then there is no allowance for violations, which is the same as the maximal margin classifier. For any $C > 0$, no more than C observations are allowed on the wrong side of the hyperplane, as $\epsilon_i > 1$ if the point is on the wrong side. In practice, C is chosen via cross-validation. As with most tuning parameters, C controls the bias-variance trade-off.

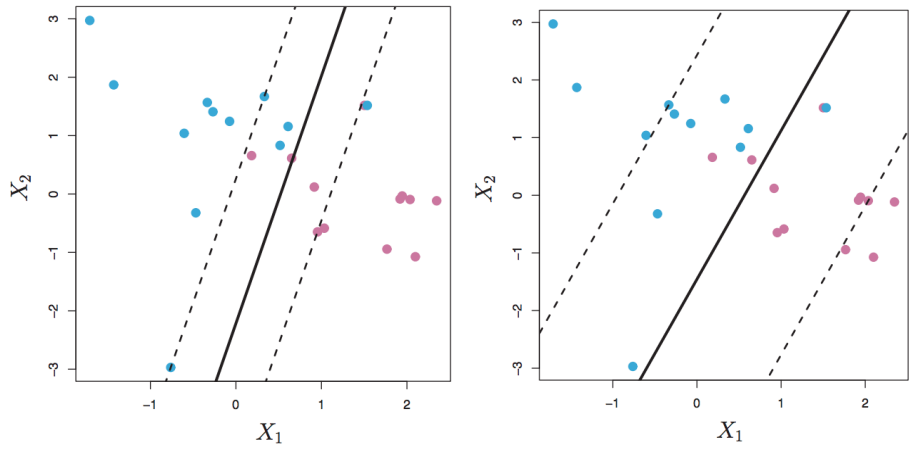


Figure 18: Example of support vector classification with different values for the tuning parameter C.

- If C is small, there is less tolerence to data points on the wrong side of the margin and hyperplane, so the classifier is more likely to overfit the data
- Conversely, if C is large, more violations are allowed, so the classifier has a higher chance of underfitting the data

Finally, an interesting point is that only data points within the margins of the classifier affect the hyperplane. Any data points that lie outside the margin have no effect on the hyperplane or classifier just like the maximal margin classifier, where only support vectors affect the classifier. By basing the decision on only a smaller subset of training points that are more likely to affect the behaviour, the method is robust to the behaviour of data far away from the hyperplane. Additional data points not close to the hyperplane do not affect the classifier in any way.

9.3 Support vector machines

We have seen how support vector classifiers can allow for data points to lie on the wrong side of the margin and hyperplane and this cost leads to building a more generalisable model. However all these classifiers so far only work with linear-decision boundaries. So the linear classifier must be converted to produce non-linear decision boundaries.

9.3.1 Non-linear decision boundaries

In polynomial regression, the same problem of converting a linear regression model to non-linear curves as found. To address the issue in support vector classifiers, the same method can be applied by enlarging the feature space using higher-order polynomial functions of the predictors. For example, instead of fitting the support vector classifier with n features

$$X_1, X_2, ..., X_n$$

$2n$ features with a quadratic function can be used instead

$$X_1, X_1^2, X_2, X_2^2, ..., X_n, X_n^2$$

Although the decision boundary found in the enlarged feature space is still linear, it is non-linear in the original feature space as it is in the form $q(x) =$ where q is a quadratic or higher-order polynomial. However, this has the issue of creating too many features for a high number of polynomials.

9.3.2 Priaml and dual optimisation problem

The support vector optimisation problem

$$\begin{aligned}
&\text{maximise } M \\
&\text{subject to } \sum_{j=1}^{n} \beta_j^2 = 1, \\
&y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_n x_{in}) \geq M(1 - \epsilon_i) \\
&\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C
\end{aligned}$$

can be turned into an objective function by **Lagrange relaxation**

$$L_P = \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n} \epsilon_i - \sum_{i=1}^{n} \alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \epsilon_i)] - \sum_{i=1}^{n} \mu_i\epsilon_i \tag{54}$$

where C, α_i and μ_i are lagrange multipliers, which serve as a substitute for hard constraints and must be positive. This objective function is called the **primal** problem.

- $\frac{1}{2}||\beta||^2$ maximises the margin by minimising $||\beta||$
- $C\sum_{i=1}^{n}\epsilon_i$ is the tolerance for violations of the margin and hyperplane
- $\sum_{i=1}^{n}\alpha_i[y_i(x_i^T\beta+\beta_0)-(1-\epsilon_i)]$ punishes errors as the expression becomes negative if a data point is on the wrong side of the margin
- $\sum_{i=1}^{n}\mu_i\epsilon_i$ makes ϵ_i positive by punishing negative values

The primal problem is difficult to optimise for. The constraints are implicit rather than hard and violations are allowed, but punished, which follows from the soft margin classifier of allowing violations. The primal problem can be transforms into the **dual problem** which is easier to solve. This is done by differentiating with respect to β, β_0 and ϵ_i, which gives

$$\beta = \sum_{i=1}^{n}\alpha_i y_i x_i \tag{55}$$

$$0 = \sum_{i=1}^{n}\alpha_i y_i \tag{56}$$

$$\alpha_i = C - \mu_i \tag{57}$$

These values can be put back into the primal function to give the dual function

$$L_D = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{i'=1}^{n}\alpha_i\alpha_{i'}y_i y_{i'}x_i^T x_{i'} \tag{58}$$

which is easier to put into a standard solver.

9.3.3 Support vector classifier solution

By solving the primal or dual functions, the solution to the support vector classifier can be found. It turns out that the solutions only involves the *inner product* of the data points, defined as

$$\langle x_i, x_{i'}\rangle = \sum_{j=1}^{n} x_{ij}x_{i'j} \tag{59}$$

The support vector classifier can then be represented as

$$f(x) = \beta_0 + \sum_{i=1}^{p}\alpha_i\langle x, x_i\rangle \tag{60}$$

where there are p parameters α_i per training example. To esimate the parameters, all that is needed is the dot product of the training pair $\langle x, x_i\rangle$. Although this means that to evaluate the function $f(x)$, the dot product must be computed for each new point x and every existing training point x_i. However, α_i is only nonzero for support vectors, so a lot of computation can be saved by only computing the functions for the set of support vectors rather than every single training point.

In summary, only the dot product $\langle x, x_i\rangle$ is needed in computing the coefficients of the linear classifier $f(x)$.

9.3.4 Kernel trick

Now the trick to extend the support vector classifier to a support vector machine that works on non-linear data is to replace the dot product with a *generalisation* in the form

$$K(x_i, x_{i'}) \tag{61}$$

where K is some kernel function. This is very similar to basis expansions as the kernel K defines a basis function to apply on x_i and $x_{i'}$.

$$K(x_i, x_{i'}) = \langle h(x_i), h(x_{i'}) \rangle \tag{62}$$

where h is the basis expansion. There are many common kernels that can be used, including the original linear kernel,

- Linear kernel

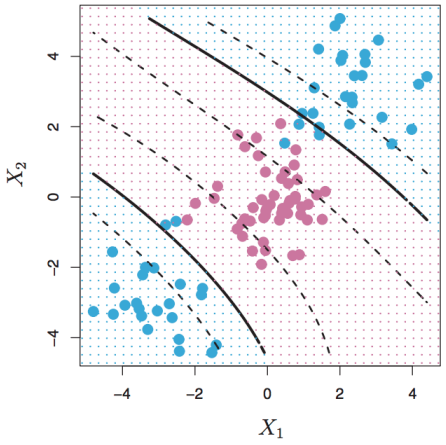
$$K(x_i, x_{i'}) = \sum_{j=1}^{n} x_{ij} x_{i'j} \tag{63}$$

- Polynomial kernel where d is the polynomial degree

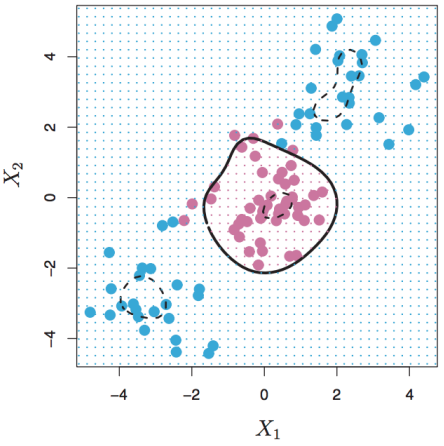
$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^{n} x_{ij} x_{i'j})^d \tag{64}$$

- Radial kernel

$$K(x_i, x_{i'}) = \exp(\gamma \sum_{j=1}^{n} (x_{ij} - x_{i'j})^2) \tag{65}$$



(a) SVM with a polynomial kernel of degree 3



(b) SVM with a radial kernel

9.4 Hinge loss function

It turns out that SVMs have a close connection to logistic regression. The optimisation problem for SVMs can be rewritten in the form

$$J(\beta) = \frac{1}{N} \sum_{i=1}^{n} (1 - y_i(x_i^T \beta + \beta_0)) + \lambda \sum_{j=1}^{n} \beta_j^2 \tag{66}$$

where

- $J(\beta)$ is the cost function
- $(1 - y_i(x_i^T \beta + \beta_0))$ is the **hinge loss**
- $\lambda \sum_{j=1}^{n} \beta_j^2$ is the regularisation

This function looks very similar to the functions for logistic regression, but using a hingle loss function rather than a squared error loss. Further, the regularisation term $\lambda \sum_{j=1}^{n} \beta_j^2$ is the same as a ridge l_2 norm in linear regression.

The difference between support vector classification, which uses the hinge loss, and logistic regression, which uses a squared error loss is the data points that are on the correct side of the margin do not affect the classifier. This comes from the hinge loss characteristic where the loss function is exactly zero when $y_i(\beta_0 + \sum_{j=1}^{n} \beta_j x_{ij})/geq1$, corresponding to data points on the right side of the margin. In contrast, the loss function for logistic regression is never zero, though it can be very small for data points that are far away.

10 Unsupervised learning

In unsupervised learning, only the set of features $X_1, X_2, ..., X_n$ measure from n data points are available, with no associated output variable Y. As such, the goal is no longer to try and predict, as there is no "correct" prediction. Instead, the goal is to discover interesting things about the data. For example is there an informative way to visualise the data? Can subgroups be discovered among the variables or data points?

Unsupervised learning is often more challenging than supervised learning. It tends to be more subjective with no set goal for analysis. Furthermore, it can be hard to assess the results as there is no universal method for validation of the results. In fact, there is no way to check the result at all as there is no true answer. Despite these difficulties, unsupervised learning techniques are of growing importance, especially given today's data driven systems to try and gain a better understanding in some field or area.

10.1 K-means clustering

Clustering techniques is a very broad term refering to finding subgroups or **clusters** in a dataset. When data points are clustered, the goal is to partition them into distinct groups so that the data points within each group is similar to each other while points in different groups are quite different from each other. What is means to be similar or different must be more concretely defined for this.

K-means clustering is a simple approach to partition the data set into K distinct, non-overlapping clusters. The K-means algorithm assigns each data point to exactly one of the K clusters. There are two properties which the K clusters satisfy

1. $C_1 \cup C_2 \cup ... \cup C_K = \{1, ..., n\}$ - Each data point belongs to **at least** one of the K clusters
2. $C_k \cap C_{k'} = \emptyset$ for all $k \neq k'$ - The clusters **do not** overlap, meaning no data point belongs to more than one cluster

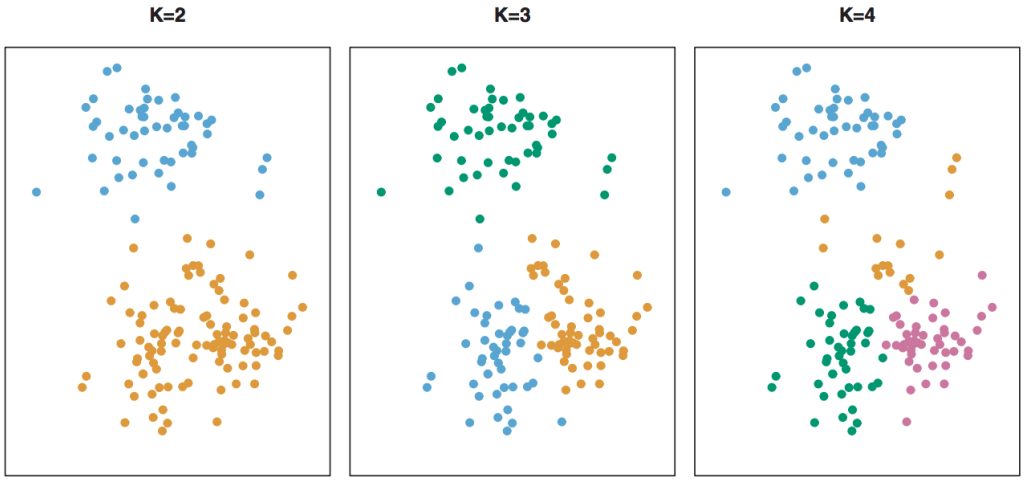


Figure 20: Example of the same data set with a different value of K.

10.1.1 Cost function

The idea of a *good* clustering is one for which the *within-cluster variation* is as small as possible. This is a measure $W(C_k)$ of the amount by which the data points within a cluster differ from each other. In other words, the optimisation problem becomes

$$\text{minimise} \left\{ \sum_{k=1}^{K} W(C_k) \right\} \tag{67}$$

This means we wish to partition the observations into K clusters such that the total variation within each cluster, summed over all clusters, is as small as possible. To do this, the measure $W(C_k)$ must be defined. There are various ways to do this, and we shall look into the squared euclidean distance $||x||^2$.

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{n} (x_{ij} - x_{i'j})^2 \tag{68}$$

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{n} (x_{ij} - \mu_c(i))^2 \tag{69}$$

The first equation defines the measure as the average distance of each point x_{ij} to each other point $x_{i'j}$ while the second equation uses the centroid position $\mu_c(i)$ to which the data point x_i is currently assigned to. This defines an average distance to the centroid.

10.1.2 K-means clustering algorithm

Given K clusters with training data x with m examples:

Algorithm 1 K-means algorithm for cluster initialisation and data point assignment.

```
procedure K-MEANSALGORITHM(K)
    Randomly initialise K cluster centroids μ_1, μ_2, ..., μ_k
    repeat
        for i = 1 to m do
            Assign x_i to cluster c_i which minimises the equclidean distance to μ_i
        for k = 1 to K do
            μ_k = mean position of data points assigned to cluster c_k
    until Cluster assignments stop changing
```

The initial location of cluster centroids are randomly initialised to prevent any bias. Moreover, the algorithm can be repeated many times with randomly initialised centroid positions and the clustering with the lowest cost $W(C_k)$ is the final clustering to pick. This is done because at the end of the algorithm, only a local minimum is found, the repetition leads to a higher chance of finding the global minimum.

10.2 Principle component analysis

Sometimes, instead of increasing the feature space with basis expansions, we wish to reduce the dimensionality of the input data. This could be done because of multiple reasons:

- Data compression to reduce the time to transfer the data and reduce the memory requirement
- Reduces time needed for computation
- Reduces amounts of redundant data

Principle component analysis is a way to reduce dimensionality automatically. It is an unsupervised technique as no output Y is needed, but is often used as a way to produce derived variables for use in supervised learning.

10.2.1 Principle components

Suppose we are given a dataset with n features $X_1, X_2, ..., X_n$ and m data points, as part of the data visualisation and exploration stage, we may want to plot a two-dimensional scatterplot for each pair of features to find any correlations. However, there are $\frac{n(n-1)}{2}$ such scatterplots. If n is large, it becomes less feasible to look at all of them. Furthermore, it is likely none of them will be informative as each only contain a small fraction of the total information present in the dataset. As a solution to this, we wish to find a lower-dimensional representation of the data that still captures as much information as possible so we can plot the data in this lower-dimensional space.

PCA does exactly this by finding a low-dimensional representation of the dataset that contains as much information as possible. Note that mean normalisation and feature scaling should be performed *before* applying PCA.

11 Neural networks

11.0.1 Issue with logistic regression

In logistic regression, when there are many input features in combination with a complex non-linear hypothesis, the number of features used grows very dramatically. For example in a small image of 50×50 pixels, there are 2500 features, which leads to over 3 million features with a quadratic model. This is computationally too expensive, making it difficult to choose a good polynomial.

Therefore, a different, more effective method was explored, which led to neural networks.

11.1 Neural network

A neural network is a biologically inspired method which tries to copy the way the brain works for classification and prediction. This method has recently become popular because of the increase in computation power with GPUs allow for the ability to deal with large scale networks and large datasets. Previously neural networks were not very effective as they were too small and worked on small datasets.

11.1.1 Artificial neuron

An artificial neuron is modelled like a neuron in the human brain. It takes a series of inputs and outputs a single value. The weights that modifier the input values act as the θ parameters that have to be adjusted during training.

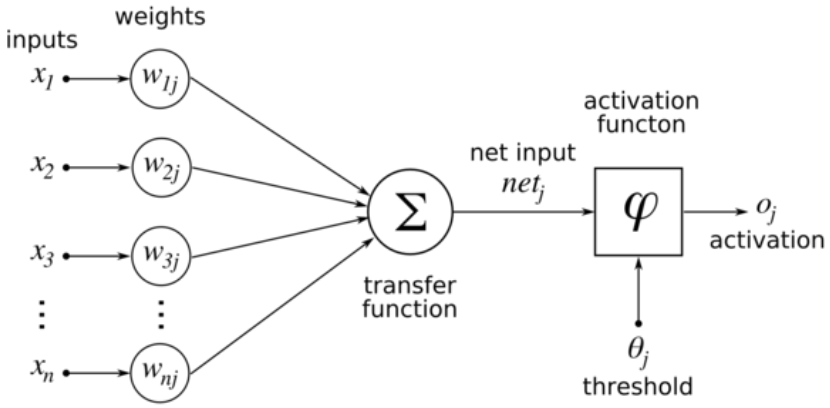


Figure 21: A single artificial neuron.

In addition to the inputs $x_1, x_2, ..., x_n$ a bias unit x_0 is often added that helps shift the activation function left or right. For the activation function, a sigmoid (logistic) activation function is often used

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{70}$$

A neural network is made up for a network of artificial neurons, most notably the input, hidden layers and output layer.

11.1.2 Artificial network

The combination of many neurons and layers creates a neural network. The network's layer are connected by passing the output of one layer in as the inputs to the next layer. In most cases, all output from one layer is used for the next layer and the weight help balance which inputs are more important for each particular neuron.

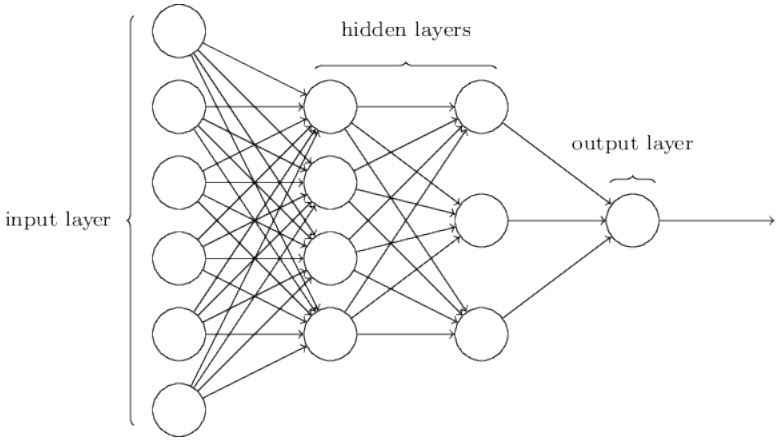


Figure 22: Example of an artificial network with 2 hidden layers.

At each step of the computation, the activation function of the next layer is the sum of the inputs and weights (w or θ), for example

$$a_1^{(2)} = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2) \tag{71}$$

for the first neuron in the second layer. $\theta_j^{(i)}$ specifies the weight parameter for the jth neuron of the ith layer. For a single two-class classification, only one output neuron is needed where the output 0 specifies the data is not of the class and output 1 specifies the data is of the class.

11.1.3 Multiclass classification

In multiclass classification, it is not as easy as using a range to specify the different classes (like 0-0.3, 0.3-0.6, 0.6-0.9). Rather we will want to use a vector to specify each class, for example

$$h_\theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ for class 1} \qquad h_\theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ for class 2} \qquad h_\theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ for class 3}$$

11.1.4 Backpropagation algorithm

Backpropagation in neural networks is a method of minimising the cost function by finding the best θ values. The initial computation from training passes through the network via forward propagation to return an output value. To adjust the weights on the next training iteration, the error between the predicted output and actual output must be propgated backwards through the network.

For example, in a four layer network, the result $a_j^{(4)}$ is the output at the output layer. The error for the result is defined as

$$\delta_j^{(4)} = a_j^{(4)} - y_j \tag{72}$$

where y_j is the actual output. This can be written as

$$\delta^{(4)} = a^{(4)} - y \tag{73}$$

to express everything in vector form. This is the error of the output layer. The error for all previous layers is then defined as

$$\delta^{(l)} = (\theta)^T \delta^{(l+1)} \times g'(z^{(l)}) \tag{74}$$

where $g'(z^{(l)})$ is the derivative of the activation function $g()$ evaluted at the input values $z^{(l)}$ Note the error $\delta^{(l+1)}$ is used to define the error of the current layer. Furthermore, the multiplication used $\times$ is an element wise multiplication of the vectors.

Algorithm 2 Backpropagation algorithm for neural networks.

```
procedure BACKPROPAGATION
    for m training samples do
        Set Δ_ij^(l) = 0 for all l, i, j
        for i = 1 to m do
            Set a^(1) = x^(i)
            FORWARDPROPAGATION to compute a^(l) for l = 2, 3, ..., L
            δ^(L) = a^(L) − y^(i)
            for l = L − 1 to 2 do
                δ^(l) = (θ^(l))^T δ^(l+1) × g'(z^(l))
            Δ_ij^(l) = Δ_ij^(l) + a_j^(l) δ_i^(l+1)
```

The backpropagation algorithm gives a final change

$$D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\theta_{ij}^{(l)} \text{ if } j \neq 0 \tag{75}$$

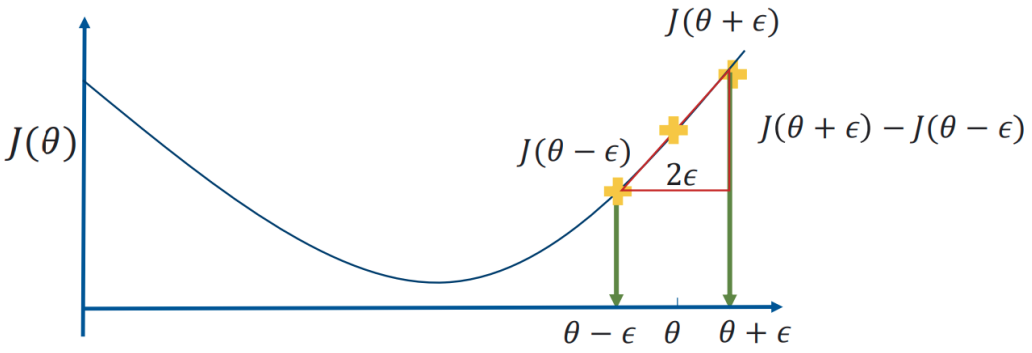
$$D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} \text{ if } j = 0 \tag{76}$$

$D_{ij}^{(l)}$ is the change to all weights, indexed by l, i, j. It is in fact the partial derivative of the cost function $J(\theta)$ with respect to each θ parameter

$$D_{ij}^{(l)} = \frac{\partial J(\theta)}{\partial \theta_{ij}^{(l)}} \tag{77}$$

11.1.5 Gradient checking

Because the backpropagation algorithm can be a difficult algorithm to debug, especially with off-by-one errors in the layer index or neuron index. A derivative checking procedure can be used to check the correctness of the backpropagation implementation. This is not done except for debugging purposes in general, because it is very slow to compute.



At any specific value of θ, the derivative can be numerically estimated as

$$\frac{dJ(\theta)}{d\theta} \approx \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon} \tag{78}$$

In practice, ϵ is set to a small value such as 10^{-4}. Gradient checking is used to compare the derivative from backpropagation against the numerical estimate.

12 Deep learning

In the basic equation for statistical prediction, we have

$$\hat{Y} = f(X, \theta) \tag{79}$$

which leads us to variations of linear functions with linear and logistic regression. There is a direct relation between the inputs and outputs but the simplicity of the model meant that it could only predict on linear data. More complex non-linear models were achieved by transforming the input data into new expanded feature spaces with an additional layer of processing

$$x \to h_m(x) \to y \tag{80}$$

where x is the original input data and $h_m(x)$ is the additional processing to transform the input space. This transformation can be done in multiple different ways for different models and techniques

- Basis expansions
- Non-linear kernels
- Hidden layers consisting of multiple neurons

We can see a pattern here of the models becoming "deeper" and requiring more processing as more complexity is needed. Moreover, feature extraction is often necessary and many steps of the feature extraction and feature mapping process were done by hand. The idea of **deep learning** is to learn all parts of the entire process from the data and be an end-to-end model.

In general, deep learning refers to algorithms which learn from intermediate representations with more than one layer of intermediate representation, for example hierarchical Bayes networks or Markov random fields. However these days, deep learning often refers to **deep neural networks**.

12.1 Deep neural network

In essence, a deep neural network is simply a feedforward network with many hidden layers. The learning algorithm must decide how to use the layers to produce the desired output. Each layer in the deep neural network performs feature mapping, which can expand or reduce the feature space depending on both the weights and the number of neurons in each layer. A deep enough network can therefore represent arbitrarily complex functions.

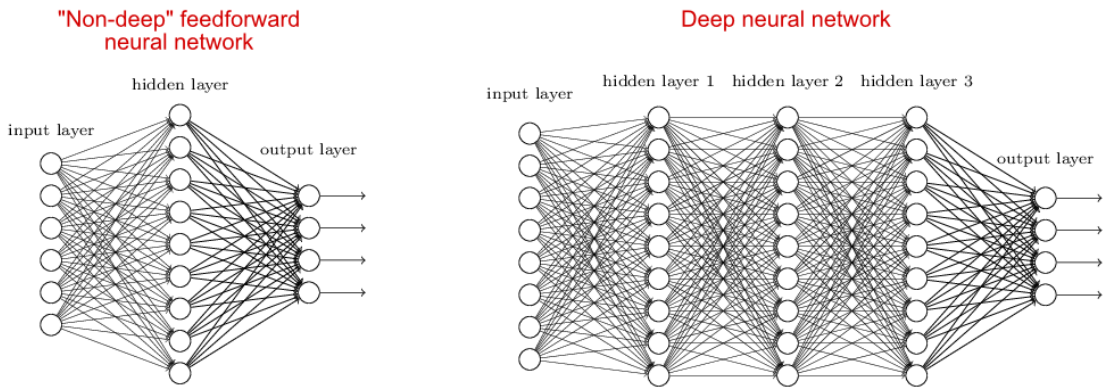


Figure 23: A deep neural network simply has many hidden layers compared to a normal neural network.

There are many problem that come with deep neural networks

- Computational complexity
- Availability of data
- Overfitting
- Non-convex cost function
- Vanishing gradient problem
- Network topology (choosing the depth and width)

12.1.1 Vanishing gradient problem

Typically in neural networks, a sigmoid function

$$f(x) = \frac{1}{1 + e^{-x}} \tag{81}$$

is used for classification, however an issue with these functions is the gradients become too small. The small gradient is propagated across the layers, which results in a very slow learning/training process.

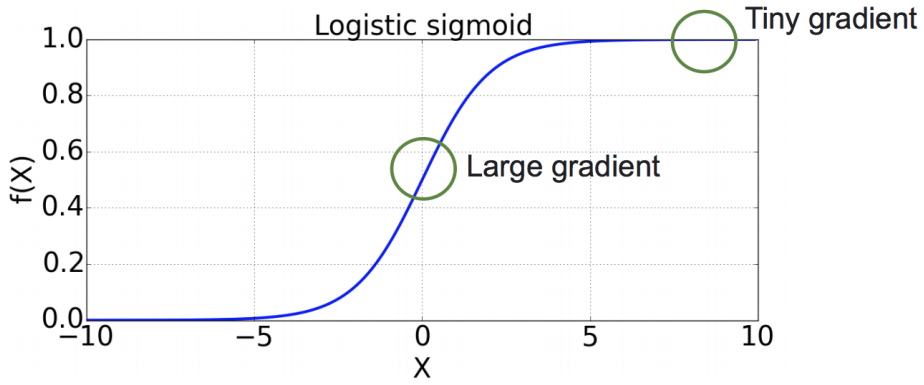


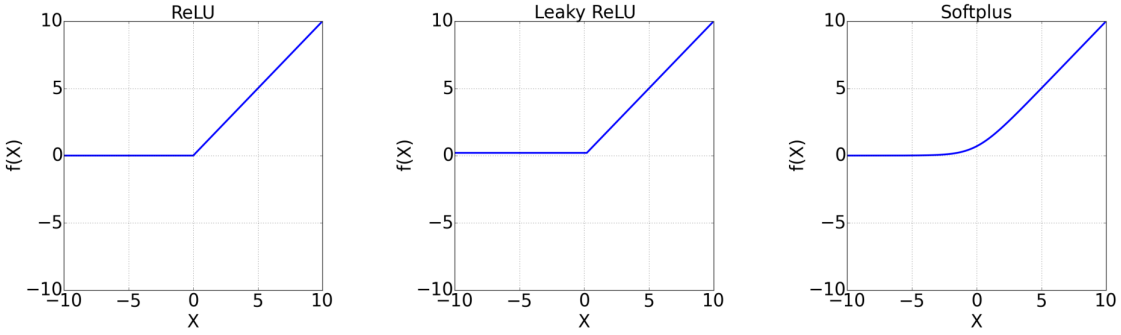
Figure 24: Example of the small gradient on sigmoid-type functions.

The issue is the simoid units saturate across most of the domain, saturating to a high value when X is very positive, a low value when X is very negative and only strongly sensitive in their input when X is near 0. This makes learning difficult, which is why sigmoid units are not used often in feedforward networks. They can still work in gradient-descent based algorithms as the cost functions can undo the saturation of the sigmoid in the output layer.

Instead **rectified units** can be used which lead to much faster learning. Rectified linear units used the activation function

$$\max\{0, x\} \tag{82}$$

The main difference with rectified units compared to linear or sigmoid units is that a rectified unit outputs zero across half of its domain. This makes the derivatives through a rectified unit remain large whenever the unit is active, making it more useful for learning.



One drawback of rectified units is that they cannot learn via gradient-descent when their activation is zero. Furthermore, the linear units lead to large values, which often need to be re-normalised for each layer separately.

12.2 Regularisation

There are many forms of regularisation, which all attempt to decrease overfitting and allow predictive models to better generalise to unseen data. This is generally a trade-off of increasing the bias by only decreasing the variance by a small amount. Some forms of regularisation in deep neural networks resemble reguarlisation methods seen before, but there are also extensions to the basic concepts that apply particularly for neural networks.

12.2.1 Norm penalities

Many regularisation approaches are based on limiting the capacity of models by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J

$$J(\theta) = L + \Omega(\theta) \tag{83}$$

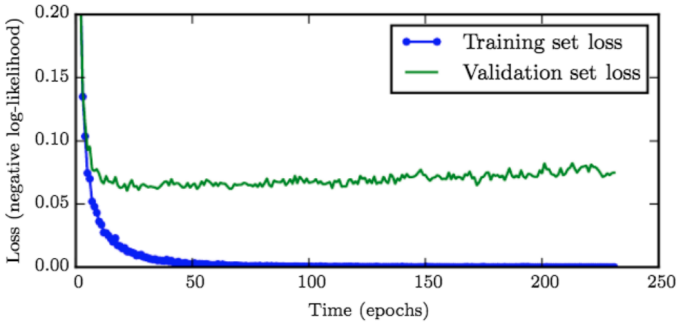
there L is the loss function used. For example, the l_2 norm can be used as the norm penalty, which gives

$$J(\theta) = L + \frac{\lambda}{2} \sum_{i,j,l,i \neq 0} (\theta) i, j^{(l)})^2 \tag{84}$$

Note the bias weights $\theta_{0,j}$ are left unregularised.

12.2.2 Early stopping

When training and adjusting weights with backpropagation, often the training error is calculated to ensure it continues to fall. However, if the model starts to overfit to the data, the training error can continue to fall even if the validation error begins to rise. This is especially common with complex models.



To prevent this, early stopping can be used to record good parameter values and stop if the validation error goes up when the training error goes down.

12.2.3 Dropout

Dropout regularisation provides an inexpensive but powerful method of regularising a broad family of models. In simple terms, dropout can be thought of as practical bagging for many large neural networks. **Bagging** is the idea of training multiple models and choosing the best one based on the evaulation with test data. This is normally impractical because of the computational cost. Dropout is the approximation to training and evaluating a bag of exponentially many neural networks.

In neural networks, particular units in the network can be effectively removed by multiplying its output value by zero. On each iteration

- Sample a different binary mask vector μ to apply to all input and hidden units in the network
- Nodes where values of μ are zero are dropped
- Only a subset of the data is trained
- Sample again and repeat

Only a tiny fraction of the possible subnetworks are each trained for a single step, and the parameter sharing causes the remaining subnetworks to arrive at good settings of the parameters.

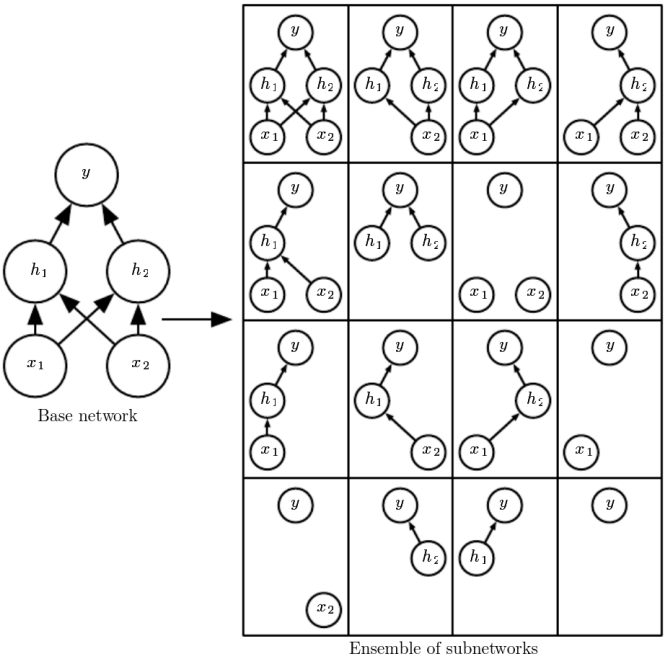


Figure 25: The subnetworks are constructed by removing non-output units from an underlying base network.

To create the subnetworks practically during forward propagation, the vector μ is randomly sampled with one entry for each input or hidden unit in the network. The entries of μ are binary (0 or 1) and are independent of each other. The probability of the entries being 1 is a hyperparameter that has to be set, it is typically 0.5 for hidden units and 0.8 for input units. Each unit in the network is then multiplied by the corresponding mask and forward propagated as normal. This is equivalent to selecting one of the sub-networks and running forward propagation through it.

12.3 Batch optimisation

As we typically wish to minimise the **mean** error in our model, we can average the error over all training samples at each iteration of the algorithm. This is known as **batch** gradient descent. Calculating the mean over the entire training set estimates the mean over all data, including the mean cost and mean gradient. In this approach, having more data leads to better estimates.

12.3.1 Minibatch optimisation

Gradients can be computed in batches of n samples points at once, or rather n epochs using m minibatches in the same time. Such learning algorithms are typically called **stochastic** as a subset of the training set is randomly chosen to propagate the error and update parameters until convergence.

The batch size is governed by a number of considerations, most notably:

- Amount of parallelism available
- Amount of memory available
- Batch size requirements (for example batches must be of size power of two)
- Larger batches provide a more accurate estimate of the gradient, but with less than linear returns

12.3.2 Stochastic gradient descent

An example of a minibatch approach to gradient descent is the stochastic gradient descent. Instead of using the entire dataset, only a batch of datapoints are sampled. The mean cost and gradient for each parameter is calculated, which updates the parameters and direction of the gradient. This is repeated with new random samples until convergence.

Furthermore, the learning rate *decreases with time*. This is done because the sampling process introduces an error and the algorithm would never converge otherwise.

12.3.3 Batch noramlisation

12.3.4 Parameter sharing

12.4 Concolutional neural networks