Stats Summary

1 Linear Regression

Linear Regression: A linear combination of n weights (θ) and features (x) defining a relationship (hypothesis) between m sets of observations and predictions:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = h_\theta(x)$$

Note that here x is a single column vector representing one data row. Using the convention that the intercept term $x_0 = 1$ (**do not forget to add!**), then:

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^t x$$

To obtain the θ_i we define a (vector valued) cost function over all m samples:

$$\begin{aligned} J(\theta) =& \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2 \\ =& \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - h_\theta(x^{(i)}))^2 \\ =& \frac{1}{2m} \sum_{i=1}^{m} (y^{(i)} - \theta^t x^{(i)})^2 \end{aligned}$$

Or, fully vectorized, where X is a $m \times n$ matrix:

$$J(\theta) = \frac{1}{2m} (y - X\theta)^T (y - X\theta)$$

Want to choose θ as to minimize $J(\theta)$. One option is to use a gradient descent algorithm, which starts with an initial guess and repeatedly performs the update:

$$\theta_k := \theta_k - \alpha \frac{\partial}{\partial \theta_k} J(\theta)$$

The update is simultaneously performed on all values of $k = 1 : n$ (we have m rows/observations, and n features).

What is $\frac{\partial}{\partial \theta_k} J(\theta)$?

$$\begin{aligned}
\frac{\partial}{\partial \theta_k} J(\theta) &= \frac{\partial}{\partial \theta_k} \frac{1}{2} \left(h_\theta(x) - y \right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_k} \left(h_\theta(x) - y \right) \\
&= \left(\theta^t x - y \right) \cdot \frac{\partial}{\partial \theta_k} \left(\theta^t x - y \right) \\
&= \left(\theta^t x - y \right) x
\end{aligned}$$

or vectorized:

$$\frac{\partial}{\partial \theta_k} J(\theta) = X^T \left(X\theta - y \right)$$

So the update is:

$$\theta = \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)} \tag{1}$$

$$= \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta^T x^{(i)} - y^{(i)} \right) x^{(i)} \tag{2}$$

Practical tricks:

- Gradient descent works well when the features are scaled to [-1 1] and with zero mean.

$$x_j := \frac{x_j - \mu_j}{\sigma_j}$$

- Hard to know in advance how many iterations GD will take to converge. Best to plot cost function progressing with number of iterations. Can also use stopping criteria to halt when change in $J(\theta)$ is small. Choosing value is tricky – use in conjunction with plot. Common value is less than 10^{-3}.
- If using normal equations scaling not needed.
- When to use normal eq vs GD? If n is large (>10,000) then $(X^T X)^{-1}$ which is $n \times n$ will be very large. Matrix inversion is $\mathcal{O}(n^3)$, so will be slow. For small number of features (< 1000) is fine.
- What if $(X^T X)$ is not invertible? Can arise if you have colinear features (non-linearly independent) , or more features than rows ($n > m$). Use `pinv` to get generalized inverse. Gives a norm(x) is smaller than the norm of any other solution. In contrast $y = A\backslash b$ gives a y has the fewest possible nonzero components (most sparse).

Normal Equations and Projection matrix: Suppose with have a design matrix, $\mathbf{X}$, with observatons on rows:

$$X = \begin{bmatrix} - - - \left(x^{(1)}\right)^T - - \\ - - - \left(x^{(2)}\right)^T - - \end{bmatrix}$$

And a linear model:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

Then the estimated coefficients are:

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}}$$
$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

The model values are given by $\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta}$, so:

$$\hat{\mathbf{y}} = \mathbf{X}\boldsymbol{\beta} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Where the projection matrix $\mathbf{P}$ which projects $\mathbf{y}$ on $\hat{\mathbf{y}}$ is given by:

$$\mathbf{P} \equiv \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$$

P is symmetric and idempotent.

In a linear regression model the leverage score for the ith observation is the ith diagonal element of the projection matrix.

2 Logistic Regression

The logistic regression model is:

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

$h(x)$ gives the probability of $y = 1$ for a particular sample x:

$$h_\theta(x) = P(y = 1|x, \theta)$$

If the decision is to predict $y = 1$ when $h(x) \geq 0.5$, then that translates into a positive prediction when $\theta^T x \geq 0.$, ie the decision boundary partitions the sample space along $\theta^T x = 0$

The cost function is the cost for each sample averaged over all samples:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{n} \text{Cost}(h_\theta(x^{(i)}, y^{(i)}) \tag{3}$$

where:

$$\text{Cost} = \begin{cases} -\log\left(h_\theta(x)\right) & \text{if } y = 1 \\ -\log\left(1 - h_\theta(x)\right) & \text{if } y = 0 \end{cases}$$

The expression for the cost can be combined to single line:

$$\text{Cost} = -y \log\left(h_\theta(x)\right) - (1 - y) \log\left(1 - h_\theta(x)\right)$$

yielding:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{n} \left[y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right) \right]$$

A vectorized implementation is:

$$h = g(X\theta)$$
$$J(\theta) = -\frac{1}{m}\left(y^T \log(h) + (1 - y^T)\log(1 - h)\right)$$

Taking the derivative of the cost function gives:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{n}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\, x_j^{(i)}$$

or vectorized:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m}(X^T\,(h - y))$$

Which can be used in the gradient descent algorithm:

$$\theta := \theta - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Which looks like this (vectorized in θ):

$$\theta := \theta - \alpha\,\frac{1}{m}\sum_{i=1}^{n}\left[\left(h_\theta(x^{(i)}) - y^{(i)}\right) x^{(i)}\right]$$

or fully vectorized:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - y)$$

Multi-class classification: One-vs-all. Convert a multi-class classification problem into multiple binary classification problems, eg. a three class problem would be turned into three binary problems.

To make a prediction simply run all three and choose the one with the highest probability.

Reguralization Suppose we have a polynomial hypothesis $h(\theta) = \theta_1\, x + \theta_2\, x^2 + \theta_3\, x^3 + \theta_4\, x^4$. And, we modify the cost function so that the higher order coefficients are penalized:

$$J(\theta) = \min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\theta_3 + 1000\theta_4$$

Now, the only way to minimize the cost function is to make θ_3 and θ_4 very small.

A natural extension of this is to regularize the sum of the parameters:

$$J(\theta) = \min_\theta \frac{1}{2m} \left[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

By convention the sum begins at $j = 1$ so that θ_0 is not penalized λ controls the balance between the two objectives of the minimization: fitting the data, and keeping the sum of the parameters small. A $\lambda = 0$ and the problem reverts to the unregularized one; a very large lambda and the regularization term becomes dominant. Geometrically, the sum of coefficients squared is a ball in theta space, and the minimization is limited to the contact between the perimeter of the ball and the cost function ellipse. The larger lambda the smaller the radius, and as lambda approaches infinity then all the coefficients apart from zeroeth one go to zero, and the minimization is limited to the origin.

The normalized version of the normal equations is:

$$\theta = \left(X\, X^T + \Lambda \right)^{-1}\, X^T\, y$$

where Λ is a product of λ and a $n + 1 \times n + 1$ matrix of ones across the diagonal except for the uppermost left corner (0,0).

Similarly for regularized logistic regression:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{n} \left[y^{(i)} \log\left(h_\theta(x^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right) + \lambda \sum_{j=1}^{n} \theta_j^2 \right]$$

A vectorized implementation is:

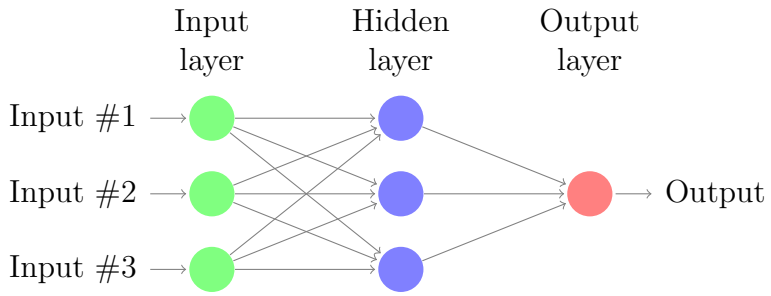
$$h = g(X\theta)$$

$$J(\theta) = -\frac{1}{m} \left(y^T \log(h) + (1 - y^T) \log(1 - h) + \frac{1}{2}\, \lambda \sum_{j=1}^{n} \theta_j^2 \right)$$

and:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m}(X^T\,(h-y) + \Lambda\,\theta);$$

3 Neural Networks:



Neural networks can be thought of a generalization of logistic regression where there is more than one layer of sigmoid transformations (activation functions). The activation units are denoted with the following notation:

$a_i^{(j)}$: activation of unit i in layer j

$\Theta^{(j)}$: Matrix of weights controlling function mapping from layer j to layer $j+1$

Specifically:

$$\begin{aligned}
a_1^{(2)} &= g(z_1^{(2)}) = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) \\
a_2^{(2)} &= g(z_2^{(2)}) = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) \\
a_3^{(2)} &= g(z_3^{(2)}) = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)
\end{aligned}$$

and:

$$h(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right)$$

Each hidden layer gets an extra "bias unit" which inputs the constant value one: $(a_0^{(2)} = 1)$.

More compactly for a single training sample (x, y):

$$a^{(1)} = x \quad (\text{add } x_0) \tag{4}$$
$$z^{(2)} = \Theta^{(1)} a^{(1)} \tag{5}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \tag{6}$$
$$z^{(3)} = \Theta^{(2)} a^{(2)} \tag{7}$$
$$a^{(3)} = g(z^{(3)}) = h_\Theta(x) \tag{8}$$

To summarize, our hypothesis output is the logistic function applied to the sum of the values of our activation nodes, which have been multiplied by yet another parameter matrix $\Theta^{(2)}$ containing the weights for our second layer of nodes. Each layer gets its own wights matrix Θ

The above computation can be vectorized for each layer from $j = 2$ to n where z, a, and Θ are matrices:

$$z^j = \Theta^{j-1} a^{j-1}$$
$$a^j = g(z^j) \quad (\text{add } a_0^{(j)})$$

The action of calculating the values of the hidden nodes, and finally the output nodes, based on the inputs is knows as forward propagation.

A concrete example demonstrating how NNs work. The following implements an OR gate. Let $x \in \{0, 1\}$ and $\Theta^{(1)} = [-10 \quad 20 \quad 20]$. Then: $h_\Theta(x) = g(-10+20x_1+20x_2)$. This will cause our hypothesis to be positive only if both x_1 and x_2 are 1. (Recall that g is the sigmoid function, so is tends to 0 at $x = -4$ and to 1 at $x = +4$).

Table 1: default

x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(+10) \approx 1$
1	0	$g(+10) \approx 1$
1	1	$g(+30) \approx 1$

Alternative, by setting the intercept to -30 one can implement an AND gate:

$$\Theta^{(1)} = [-30 \quad 20 \quad 20]$$

Or a NOR gate:

$$\Theta^{(1)} = [10 \quad -20 \quad -20]$$

By combining the X1 AND X2 gate, and the NOR gate ((NOT X1) AND (NOT X2)) gate as a hidden unit, and X1 OR X2 gate as output unit, we can build a XNOR gate:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_0 \\ a_1^{(2)} \\ a_2^{(2)} \end{bmatrix} \rightarrow \left[a^{(3)}\right] \rightarrow h_\Theta(x)$$

So:

$$\begin{aligned} a^{(2)} &= g(\Theta^{(1)}\, x) \\ a^{(3)} &= g(\Theta^{(2)}\, a^{(2)}) \\ h_\Theta(x) &= a^{(3)} \end{aligned}$$

For multi-class classification we set the output layer to be a number of nodes, instead of a single node, so that the NN outputs vectors. Accordingly, we change the labels from scalers to vectors. So if an image can be one of four things then the label representing thing four takes the form of:

$$y^{(i)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Training NNs: The first step is forward propagation and calculation of a cost. The cost function is as follows. If the output for each observation is a vector $h_\Theta \in \mathbb{R}^K$, and $h_\Theta(x^{(i)})_k$ is the k^{th} entry of the output, then the cost function is similar to the cost for logistic regression but summed over all k entries of the output vector of each sample, as well as over all samples:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{K} y_k^{(i)} \log h_\Theta(x^{(i)})_k + (1 - y_k^{(i)}) \log\left(1 - h_\Theta(x^{(i)})_k\right)\right] + \mathrm{R}$$

The regularization term R is similar to the regularization term for logistic regression, but now every entry of every matrix $\Theta^{(l)}$ is squared and the whole lot is summed

together across all rows, all columns, and all matrices:

$$R = \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

Here L is the total number of layers in the network and s_l is the number of units (not including bias) in layer L.

Alternatively, define a matrix Y such that:

$$Y = \begin{bmatrix} - - - \left(y^{(1)}\right)^T - - \\ - - - \left(y^{(2)}\right)^T - - \end{bmatrix}$$

and similarly define a matrix H which consists of all the hypotheses $h_\Theta(X)$ for all samples

$$H = \begin{bmatrix} - - - \left(h_\Theta(x^{(1)})\right)^T - - \\ - - - \left(h_\Theta(x^{(2)})\right)^T - - \end{bmatrix}$$

then:

$$J(\Theta) = -\frac{1}{m} \Big[\sum_{column} \sum_{rows} \left(Y \otimes \log H + (1 - Y) \otimes \log(1 - H) \right) \Big] + R$$

where $\otimes$ denotes element-wise multiplication and R is:

$$R = \frac{\lambda}{2m} \left(\sum_{column} \sum_{rows} L\Theta_1^{\otimes 2} + \sum_{column} \sum_{rows} L\Theta_2^{\otimes 2} + \dots \right)$$

where L chooses all columns but the first one (we don't regularize the bias wieghts), and the squaring is performed element-wise.

The next step is back-propagation and calculation of gradients of the cost with respect to the various parameters. The intuition here is that $\delta_j^{(l)}$ is the "error" of node j in layer l.

Example for a four-layer NN:

$$\begin{aligned} \delta^{(4)} &= a^{(4)} - y \\ \delta^{(3)} &= (\Theta^{(3)})^T \delta^{(4)} \otimes g'(z^{(3)}) \\ \delta^{(2)} &= (\Theta^{(2)})^T \delta^{(3)} \otimes g'(z^{(2)}) \end{aligned}$$

where $g'(z^{(l)}) = a^{(l)} \otimes (1 - a^{(l)})$. There is no $\delta^{(1)}$. The values of δ can then be used to calculate the derivatives:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

The backprop algorithm is as follows:

4 Resampling Methods

- k-fold cross validation:

$$\text{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\text{MSE}_i$$

- LOOCV is simply a special case of k-fold CV where k=n.
- With linear least-squares models the following holds:

$$\text{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - h_i}\right)^2$$

  where y_i is the ith value from the original least squares fit and h_i is the leverage (for linear regression):

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{i'=1}^{n}(x_{i'} - \bar{x})^2}$$

- In a classification setting we use the number of missclassified observations:

$$\text{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n} Err_i$$

5 Tree-Based Methods

For classification trees the deviance (as reported in `summary`):

$$d = -2\sum_m\sum_k n_{mk}\log\hat{p}_{mk}$$

where n_{mk} is the number of observations in the mth terminal node that belongs to the kth class. The residual mean deviance is the deviance divided by $n - |T_0|$ where n is the number of observation and T_0 is the number of terminal nodes. Here $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.

The residual mean deviance is reported by dividing the deviance by $n - T_0$, where T_0 is the terminal number of nodes.

$$RMD = \frac{d}{n - |T_0|}$$

6 Hashing Trick

A hash table is a dictionary. It uses a hash function to map keys to indices of buckets which contain values. Many hash tables have imperfect hashing in which different keys have the same index in the table (collisions). On the other hand, hash tables have efficient lookup: the average cost is independent of the number of elements stored in the table. In many situations hash able turn out to be more efficient than search trees or other table lookup structures.

7 Metrics

Confusion matrix: True on the rows and predicted on the columns.

		Predicted	
		T	F
TRUE	T	TP	FN
	F	FP	TN

Precision: aka positive predictive value (PVV). Is correctly identified positives over all predicted positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall: aka sensitivity, true-positive rate (TPR). Is the correctly identified positives over all actual positives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

P-R curve As recall is a measure of false negatives and precision is a measure of false positives there is a tradeoff between them. One could capture all terrorists (reduce false negatives) by accusing the entire population of being terrorists. One could try to reduce false positives (false accusations) by accusing only a small number of people. This would naturally result in a few getting through (false negatives). If

one starts with a low threshold (accusing everybody) and slowly increases it then one moves from high recall to high precision. Ideally even at a low threshold one has high precision. Hence the area under the P-R curve can serve as a metric.

Fallout aka false-positive rate (FPR), is the ratio of false-positives (incorrectly identified negative) to actual negatives (both correctly and incorrectly identified). Gives probability of a false alarm.

$$\text{Fallout} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

ROC curve It is a plot of the recall (sensitivity, TPR) against fallout (FPR) for the different possible thresholds of a diagnostic test (T).

Accuracy: Fraction of instances where predicted score equals actual score.

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Calculated with:

$$\text{accuracy}(\text{y}, \hat{\text{y}}) = \frac{1}{n_{samples}} \sum_{i=0}^{n-1} 1_{\hat{y}_i = y_i}$$

F1-score:

$$\text{F}_1\text{-score} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

8 Combinatorial Analysis

The basic principle of counting: If there are two experiments, one with n outcomes and one with m outcomes, then there are $m\,n$ possible outcomes of the two experiments. More generally, if there are r experiments each with n_i outcomes, then the total number of outcomes is $n_1 \cdot n_2 \cdot n_3 \cdots = \prod_{i=1}^{n} n_i$

Permutations: How many permutations of the letters $a\,b\,c$ are there? First option is for either one of the three, then the remaining two, and then the remaining one. So in total $2 \cdot 2 \cdot 1 = 6$. In general, if there are n different items to permute, then there are $n \cdot (n-1) \cdot \cdots \cdot 2 \cdot 1 = n!$ options.

Permutation of identical objects: How many arrangements of the letters PEPPER are there? There are 6! possible arrangements of the letters but there are 3! identical permutations of P and 2! identical permutations of E. So in total there are $\frac{6!}{3!\,2!}$ unique permutations. In general, given n objects, of which n_1 are alike, n2 are alike and so forth, then there are

$$\frac{n!}{n_1!\,n_2!\ldots n_r!}$$

Combinations: How many different groups of 3 can be formed from the letters ABCDE? There are 5 ways to choose the first member of the group, 4 of the second, and 3 of the third, so $5 \cdot 4 \cdot 3 = \frac{5!}{2!}$ possible permutations of groups of three, but within group permutations (BAC, CBA ...) do not count so need to divide by 3!. In total $\frac{5!}{(5-3)!\,3!} = 10$ groups can be formed. In general, there are such many ways to form r groups out of n objects:

$$\binom{n}{r} = \frac{n!}{(n-r)!\,r!}$$

This represents the number of different groups of size r that could be selected from a set of n objects when the order of selection is not considered relevant.

Binomial Coefficients: The values $\binom{n}{k}$ are sometimes known as binomial coefficients since they play a role in the binomial theorem:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{r} x^k\, y^{n-k}$$

Multinomial coefficients: A set of n distinct objects is to be divided into r groups of varying sizes $n_1, n_2, \ldots, n_r$ that sum up to n. How many possible partitions are there?

$$\binom{n}{n_1, n_2, \ldots, n_r} = \frac{n!}{n_1!\,n_2!\ldots n_r!}$$

The multinomial coefficients show up in the generalization to the binomial theorem, which is known as the multinomial theorem:

$$(x_1 + x_2 + \cdots + x_r)^n = \sum \binom{n}{n_1, n_2, \ldots, n_r} x^{n_1} {x_2}^{n_2} \ldots {x_r}^{n_r}$$

Summation permutations: A group of n objects is to be decomposed into r non-negative groups of unknown sizes. How many possible partitions are there?

$$\binom{n + r - 1}{r - 1}$$

9 Conditional Probability

Conditional Probability

$$P(E|F) = \frac{P(E \cap F)}{P(F)}$$

The Multiplication Rule

$$P(E_1\, E_2\, \ldots\, E_n) = P(E_1)\, P(E_2|E_1)\, P(E_3|E_1\, E_2) \ldots\, P(E_n|E_1\, E_2 \ldots E_{n-1})$$

Bayes's Formula Conditional times marginal equals conditional times marginal equals joint. Useful for evaluating conditional probabilities.

$$P(F|E)\, P(E) = P(EF) = P(E|F)\, P(F)$$

$$P(F|E) = \frac{P(E|F)\, P(F)}{P(E)}$$

$$\begin{aligned} P(E) &= P(EF) + P(EF^c) \\ &= P(E|F)\, P(F) + P(E|F^c)\, P(F^c) \\ &= P(E|F)\, P(F) + P(E|F^c)[1 - P(F)] \end{aligned}$$

Odds of an event The odds of an event happening are defined as:

$$\frac{P(E)}{P(E^c} = \frac{P(E)}{1 - P(E)}$$

Thus, if hypothesis H is true with probability $P(H)$, how do the odds of the hypothesis change with the introduction of new evidence E?

$$\frac{P(H|E)}{P(H^c|E)} = \frac{P(H)}{P(H^c)} \frac{P(E|H)}{P(E|H^c)}$$

The new odds of the hypothesis being correct are the old odds weighted by the odds of the the likelihood of the evidence given the correctness of the hypothesis.

Bayes's formula for multiple events

$$P(F_j|E) = \frac{P(E\,F_j)}{P(E)} = \frac{P(E|F_j)\,P(F_j)}{\sum_{i=1}^{n} P(E_i|F_i)\,P(F_i)}$$

Independent Events If E and F are independent then:

$$P(EF) = P(E)\,P(F)$$