Introduction to Machine Learning for Computer Graphics

Peter M. Hall*
University of Bath

Abstract

Computer Graphics is increasingly using techniques from Machine Learning. The trend is motivated by several factors, but the difficulties and expense of modelling is a major driving force. Here 'modelling' is used very broadly to include models of reflection (learn the BRDF of a real material), animation (learn the motion of real objects), as well as three-dimensional models (learn to model complex things).

Building around a few examples, we will explore the whys and hows of Machine Learning within Computer Graphics. The course will outline the basics of data-driven modelling, introduce the foundations of probability and statistics, describe some useful distributions, and differentiate between ML and MAP problems. The ideas are illustrated using examples drawn from previous SIGGRAPHs; we'll help non-artists to draw, animate traffic flow from sensor data, and model moving trees from video.

1 Why is Machine Learning of value to Computer Graphics?

Machine Learning is of growing importance to many areas of modern life; 'big-data', security, financial prediction, and web-search are areas that impact lives of almost everyone. Computer Graphics, along with allied fields, like Computer Vision, are also benefiting from ideas and techniques that originate from Machine Learning. Computer Graphics research that pulls on Machine Learning is typified by the inclusion of real-world data.

Users of Computer Graphics – the creative industries in film, broadcast, games, and others – articulate the value of acquiring models (in a broad sense) from real data. Modelling for them is a very expensive business. Consider the effort involved in modelling a landscape populated with trees, rivers, flower meadows, stones on the ground, moss on the stones ... now animate it. Machine Learning offers a feasible way towards less expensive acquisition of models, because it is able to construct models from data.

Looking back at past SIGGRAPHs and Eurographics there is a clear trend towards to use of Machine Learning in Computer Graphics, see Figure 1. The data in the figure are my interpretation of papers presented; others may take a different view. Even so, when compared with 5 or 10 years ago the recent trend is unmistakable.

The influence of Machine Learning reaches across almost all of Computer Graphics, but clusters around subjects such as modelling the reflection properties of surfaces, animation of people and faces, organising databases of models, capturing natural phenomena, interaction, and other areas where data-driven, example-based input offers an advantage.

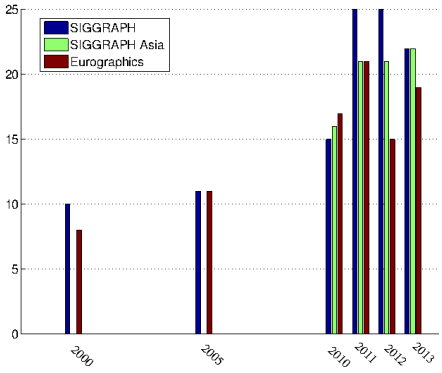


Figure 1: *The percentage of papers relating to Machine Learning in SIGGRAPH, SIGGRAPH Asia, and Eurographics, 2000-2013.*

In this course we will first introduce the basics of Machine Learning. We will first look at three of the generic problems: regression; density estimation; and classification. Next we will study the underlying mathematics used, later illustrated using two examples taken from SIGGRAPH 2013 and one example from SIGGRAPH Asia 2011.

Our first example is a *crowd sourcing* application of Machine Learning to help people draw; the input set comprises raw pen strokes made by a user, the output set comprises refined user strokes, and the function is learned by looking at pen strokes made by many different people.

The second example comes from the realistic animation of traffic flow in a city. In this case reading from sensors that monitor traffic flow make up the input set, the output set is vehicle trajectories. The parameters of a simulation are learned to enable realistic animation.

The final example is modelling moving trees. The input is now a video of a real tree, and the output is a statistical description of the moving tree that enables new individual trees to be spawned. This example shows how learned probabilistic rules can replace hard-coded rules of production.

2 Machine Learning and Data Modelling

Given a model, the aim of Computer Graphics is to render it. In mathematical terms, a model, x, is input to a rendering function f, which outputs an image y. That is $y = f(x)$. This makes Computer Graphics a *forward problem*.

Computer Vision, on the other hand, is an *inverse problem*. It tries to compute a model from the input, $x = f^{-1}(y)$. More often, it uses many images, $x = f^{-1}(y_1, y_2, ..., y_n)$.

Machine Learning differs from both: it aims to learn the function that maps inputs, x to outputs y. We may think of Graphics and Vision as travelling in opposite directions over the same road; but

*e-mail:maspmh@bath.ac.uk

SIGGRAPH 2014, August 10 – 14, 2014, Vancouver, British Columbia, Canada.

ACM 978-1-4503-2962-0/14/08

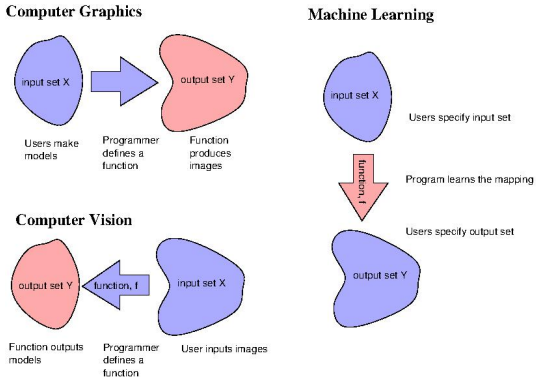


Figure 2: *A schematic view of Computer Graphics, Computer Vision, and Machine Learning. The elements in blue are inputs to some computer program, the element in red is what the program produces.*

Machine Learning is somehow orthogonal to both, as depicted in Figure 2.

Of course, Machine Learning in its 'pure' form is abstract – it does not care what the elements of the input and output sets actually are, although in practice the elements are mathematical objects such as numbers or vectors. Pure Machine Learning does not care about 3D models or 2D images, it talks instead about *events*, and about the *probability* of events. An event can be pretty much anything: the result of flipping a coin, the weather on a particular day, a three dimensional object or a picture. The probability of an event is a measure of the chance it will occur.

The subject of Machine Learning can be said to be *data modelling*. In very broad terms, Machine Learning algorithms assume there exists two sets; a domain,

$$\mathbb{X} = \{x_i\}_{i=1}^N,$$

and a range,

$$\mathbb{Y} = \{y_i\}_{i=1}^N,$$

that these are linked by a function,

$$y = f(x).$$

We can write the task of an Machine Learning algorithm as

$$f = \mathbf{MachLrn}(\{x\}, \{y\}) \tag{1}$$

where the inputs to the algorithm are subsets of the domain and range: $\{x\} \subset \mathbb{X}$, and $\{y\} \subset \mathbb{Y}$. The subsets contain *samples* of the wider *population*, typically samples are observed in some way (by sensors, or by a human). Since the wider population may be an infinite set it is often the case that samples are the best we can hope for. All Machine Learning algorithms need a sampled domain as input; but as we will see, they do not all need a sampled range.

In the rest of this course we will use $\mathbb{X}$ to mean a sampled domain, and $\mathbb{Y}$ to mean sampled range values. We will assume that the values in these sets are not perfect, but have been jittered away from their true values: they are noisy data.

We will now consider three generic Machine Learning problems to make the idea of data modelling more concrete: *regression*, *density estimation*, and *classification*, explaining these in general terms and using the concepts just introduced. After which, we'll provide mathematical underpinning so that we can then study three SIGGRAPH papers from an Machine Learning point of view.

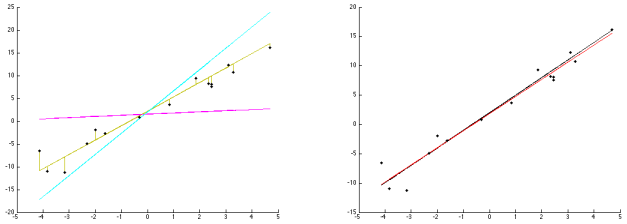


Figure 3: *Left sample points and postulated lines with error vectors. Right, the least-squares line (red) and true line (black).*

2.1 Regression

Regression is the problem of trying to find a function f given both a domain $\mathbb{X}$ and a range $\mathbb{Y}$; in the jargon we must 'fit the function'. This problem is common across all of science and engineering. The functions can be anything. Lines, $y = mx + c$, quadratics $y = a + bx + cx^2$, and general polynomials $y = \sum_i w_i x^i$ are common. More difficult would be functions such as $\exp(-\alpha x^2)\cos(2\pi x\omega)$. The functions can be multi-variate (x is a vector) or multivalued (y is vector), or both.

In all cases we can write $y = f(x, \theta)$, and the task of a regression algorithm is to find the optimal parameters, θ. The data can be written in pairs, $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^N$.

One of the easiest examples is to fit a straight line through data points, as seen in Figure 3. In this case the domain $\mathbb{X} = \{x_i\}_{i=1}^N$ comprises 1D values along the abscissa, and the range $\mathbb{Y} = \{y_i\}_{i=1}^N$ are corresponding 1D values over the ordinate.

The task is to find the values of slope, m, and intercept c for the function $y = mx + c$. If there were no errors in the data then every thing would work out perfectly and $y_i = mx_i + x$ would be true for the training data. In practice the data are noise ridden, so that the value of $mx_i + c$ is never exactly equal to y_i. Even if the slope and intercept were known for sure we would still see an error, which for the pair (x_i, y_i) is

$$e_i = mx_i + c - y_i. \tag{2}$$

A simple but effective approach of determining m and c is to use *least squares fit*. This uses the sum of the square of all errors as an *objective function*:

$$E(m, c) = \sum_{i=1}^{N} (mx_i + c - y_i)^2. \tag{3}$$

The value of the objective function depends on the parameters, $\theta = (m, c)$, we are looking for – vary the parameters (*i.e.* change the line) and the error will change. Parameter values are found which minimise the objective function. Written mathematically the problem is

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, E(\theta). \tag{4}$$

Finding the parameters is an *optimisation* problem. Using least squares to find the parameters of a line, or indeed any polynomial, is relatively easy in that an analytic (closed-form) solution is available. It can be shown that

$$\begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N x_i^2 & \sum_{i=1}^N x_i \\ \sum_{i=1}^N x_i & N \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^N x_i y_i \\ \sum_{i=1}^N y_i \end{bmatrix}. \tag{5}$$

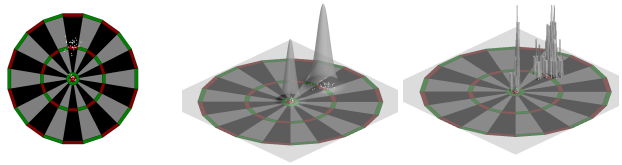


Figure 4: *Density estimation. Left dart throws. Middle, parametric estimation. Right non-parametric estimation.*

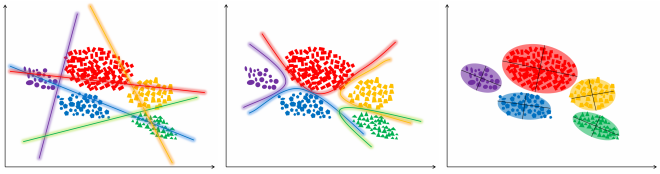


Figure 5: *Classification illustrated using data points plotted in a two-dimension space. The data are coloured according to class, a classifier tries to find the boundaries between classes. Left to right: linear SVM, non-linear SVM, and GMM for shapes (circle, pentagon, square, trapezoid, triangle) – zoom in to see them. Misclassification is a common problem.*

CLosed-form solutions are available for any polynomial. In general though, some kind of search is needed.

Regression is not just for straight lines. The least squares method is very general and can be used to fit a wide range of different types of function. Regression is behind modern model acquisition techniques such as laser scanning. The scanner produces a *point cloud*, in which $\mathbb{X}$ comprises points x_i, each in three dimensions. Typically, subsets of points are selected and planes are fitted through them by assuming a contour, that is $f(x, \theta) = 0$, where θ is a tuple that contains the parameters of the plane (*e.g.* a point on the plane and a unit normal to it).

2.2 Density Estimation

The density of physical matter is defined as the mass per unit volume. If we think about matter as being composed of particles, each of the same mass, then the density is directly proportional to the number of particles in a region.

Similarly, we can think about *data density*. This means we think about elements in the domain, x, as being points in space. It is common to observe more points in some regions of space, and fewer elsewhere. Studying these patterns is a key part of Machine Learning. An analogy would be salt spilled onto a table. The number of grains of salt in a region of the table is the local density, and the pattern of salt could lead us to infer about the manner in which the salt was spilled.

Density estimation requires only a domain, $\mathbf{X}$, as input to the Machine Learning algorithm. The range, $\mathbb{Y}$, is now the local density, hence it is not given as input to a density estimator, rather it is computed. Notice this is equivalent to determining a function, because a function can be defined in general by listing the range elements that correspond to domain elements (as in a look-up table).

The importance of density estimation comes from the fact that the data density is the probability of observing a point sampled at random from the population. For example, suppose you wished to animate someone throwing darts at a board. You might make physical model complete down to the tiniest details including muscle strength, air currents, the shape of the dart's feathers, and so on. It would be much easier to model the probability of where the dart will land. If you could model the density of real dart-holes from a real player, then an avatar could emulate that real player, see Figure 4.

There are many ways to estimate data density, but the large-scale categories are *parametric* and *non-parametric*. The *parametric* methods fit a function to the density of data in $\mathbf{X}$. Often a *Gaussian Mixture Model* (GMM) is used, which is a weighted sum of K Gaussians. We'll consider Gaussians in Section 3.3.1, but a GMM is

$$f(x) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(x|\mu_i, C_i) \tag{6}$$

with α_i the weights (which must sum to 1) and each (μ_i, C_i) specifies a particular Gaussian *component*. The parameters α_i, μ_i, C_i for a GMM density estimator can be found by regression, albeit a more complex problem than the straight line – and search must be used. The *Expectation-Maximisation* [Dempster et al. 1977] is a popular choice – but that algorithm is beyond the scope of this course. Once fitted, a parametric density estimator can be very informative and useful, provided the underlying density is well approximated by the chosen functions (such as the Gaussian).

The *non-parametric* alternative gets around the problem of having to decide which function to use to model data density. The *Parzen window* method pretends each data point x is the centre of a fuzzy ball that can be thought of a representing uncertainty in the point's location. This uncertainty is mathematically modelled using a *kernel*, $h(x, x_i)$ that specifies the density at point x due to the data point x_i:

$$f(x) = \frac{1}{N\rho^d} \sum_{i=1}^{N} h\left(\frac{x - x_i}{\rho}\right). \tag{7}$$

Typical kernels include the 'top hat' $k(x) = 1$ only if $|x| < \rho$ for some radius ρ; Gaussian kernels are also used, usually fixed to have a tight spread.

Density estimators allow us to compute important properties of data. They are needed to solve problems such as

$$x^* = \underset{x}{\operatorname{argmax}}\, p(x). \tag{8}$$

Density maxima in a GMM occur at the μ parameters, which are found during fitting. In the non-parametric case an algorithm such as *mean shift* [Comaniciu and Meer 2002] is used to search for peaks.

2.3 Classification

Classification is one of the most important problems in Machine Learning. As humans we classify things as a matter of habit; nouns are names for classes. The idea is that the domain set $\mathbb{X}$ can be partitioned into subsets $\mathbb{X}_1, \mathbb{X}_2, \ldots$, each one being a different *class*. In this case the range $\mathbb{Y}$ contains class labels. As an example, if $\mathbb{X}$ is the set of mammals, then the class subsets could be individual species and elements of $\mathbb{Y}$ would be the name of a species.

Considered as an abstract problem, classification is the problem of finding boundaries between groups of data points in some space. The space is often called a *feature space* because the objects that are to be classified are represented with tuples of values, called *feature vectors* these are the elements x_i of $\mathbb{X}$. This is illustrated in Figure 5 for two-dimensional data, built from real data used to automatically discover shapes in image segmentations [Wu and Hall 2004].

There are many classifiers. Two of the best known are the Gaussian Mixture Model (GMM) and the Support Vector Machine (SVM). The GMM assumes that each component of a GMM covers just one class, the data in any one class is *Gaussian distributed*. SVM algorithms construct surfaces that separate one class from another, as seen in Figure 5. Usually these surfaces are planes (hyper-planes if the data are high-dimensional) but some methods allow more general separating surfaces.

In terms of finding a function, the problem is this: given data $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^N$, find a function such that $y = f(x)$ is the correct class label for any new input x. Constructing such a classifier is called *training*. There are many different ways to train a classifier. In *batch training* the training data is presented all at once, whereas in *incremental training* a stream of incoming data are used to continually update and (hopefully) improve an existing function.

The problem as stated above is more correctly called *supervised training* because each element of the domain $\mathbb{X}$ is labelled with an element from the range $\mathbb{Y}$. The more difficult alternative is *unsupervised training* requires only the domain $\mathbb{X}$. Unsupervised algorithms must 'discover' the classes for themselves; the labels are called *latent variables* – that is variables that exist but which cannot be observed directly. In unsupervised learning it is common to assume that datum from the same class cluster around a local maxima in a density estimate.

3 Underpinning Mathematics

We will now consider the mathematics that underpins Machine Learning. In order to be general, Machine Learning follows the mathematics of statistics and probability. Rather than talk about specific objects like six-sided die, or dogs, or weather, Machine Learning talks about *events*. An 'event' can be pretty much anything we want it to be. The idea is that there is some kind of process that generates events – the roll of a die, the fact a specific dog exists, rainy weather are all events. We get to see specific outcomes, that is we get to *observe* specific events called *samples*, but the events we see are drawn at random from a much wider set (often infinitely large) of possibilities that are hidden from view.

An analogy is drawing coloured marbles from a velvet bag. We dip our hand in to blindly pick a marble, which may be coloured say red, green, or blue. Drawing the marble is an event, and we can observe its colour. If we draw enough marbles we may be able to guess the ratio of red to green to blue marbles in the bag, but we can never open the bag up and look at all marbles at once, so we can never be sure the ratio we guess at is correct.

A **random variate** is a possible observation. In the bag-of-marbles example the variates are red, $green$, $blue$. When we make an observation (draw from the bag) we assign the value of one of these variates to a **random variable**. Of course, the probability of assigning a colour, say, 'blue' to a variable depends on the fractional number of blue marbles in the bag. We write $p(x = blue)$ to mean that the random variable x is assigned the random variate value $blue$ with probability p. In the Machine Learning texts the shorthand $p(x)$ is used to mean that $p(.)$ is a function over the set of variates.

3.1 Statistics, Probability, and Expectations

It can be easy to confuse statistics with probability, not least because the confusion permeates popular language and is evident in newspapers and the media. We should therefore begin by making the difference between these clear.

Statistics are numbers that summarise data. Important examples far one-dimensional data include the average and the variance:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{9}$$

$$\nu = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2. \tag{10}$$

The *standard deviation* is the square root of the variance: $\sigma = \sqrt{\nu}$.

The mean and variance generalise to data in higher dimensions, but the variance is now a matrix called the *covariance*:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{11}$$

$$C = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)(x_i - \mu)^T. \tag{12}$$

As an example, consider $\mathbb{X}$ as a set points in three dimensions. Then μ will also be three dimensional and the covariance will be

$$\begin{bmatrix} \sum_{i=1}^N u_{i1}^2 & \sum_{i=1}^N u_{i1}u_{i2} & \sum_{i=1}^N u_{i1}u_{i3} \\ \sum_{i=1}^N u_{21}u_{i1} & \sum_{i=1}^N u_{21}^2 & \sum_{i=1}^N u_{21}u_{i3} \\ \sum_{i=1}^N u_{31}u_{i1} & \sum_{i=1}^N u_{31}u_{21} & \sum_{i=1}^N u_{i3}^2 \end{bmatrix} \tag{13}$$

in which $u_i = x_i - \mu$.

Probabilities are numbers in the range 0 to 1; they measure the chance of an event. A probability of 0 means the event will never happen, whereas a probability of 1 means the event is certain. Typical examples often given are the probability of rolling a 6 with a fair die, or flipping a heads on a coin. Examples of relevance to the three problems discussed follow.

The probability that a function value $f(x)$ has value y is written as $p(f(x) = y)$, or simply $p(y)$. When the function has been fitted by regression, this probability is a reflection of the uncertainty in the fitting process.

The probability that a variable x appears in the data set $\mathbb{X}$; written as $p(x \in \mathbb{X})$. Notice that this is proportional to the local density. This is easy to understand if we use the dart-throwing example, because now the question "what is the probability x belongs to $\mathbb{X}$?" can be paraphrased as "what is the probability that a dart will land at a particular location?", and that is clearly related to local density.

The probability that a variable x belongs to class $\mathbb{C}$ is an event, $p(x \in \mathbb{C})$. It is often written down at $p(\mathbb{C}|x)$, which can be roughly translated as "the probability we are looking at an element of class $\mathbb{C}$, given we are looking at object x". In some cases this can be very clear cut, if x is an animal, it normally easy to say whether or not it is a dog or cat. Other times it is much less certain. If x is a patch of colour it may be hard to say that patch is grass-green. The probability indicates the level of certainty in the classification.

Note the $p(x|\mathbb{C})$ also appears in the texts. This is roughly translated as "the probability that we'd see a thing that looks like x, assuming it comes from class $\mathbb{C}$". So, the class of dogs exhibits considerable variety, and it is just possible that a dog may be born that hardly looks like a dog at all; it would have a low $p(x|\mathbb{C})$ value.

The *expectation value* of a function is defined as a weighted sum. The weights are probability values:

$$\mathbb{E}[f] = \int_{-\infty}^{\infty} f(x)p(x)dx. \tag{14}$$

The mean of a set numbers is an expectation value, because $1/N \sum x_i = \sum x_i p(x_i)$, with $p(x_i) = 1/N$ for all x_i.

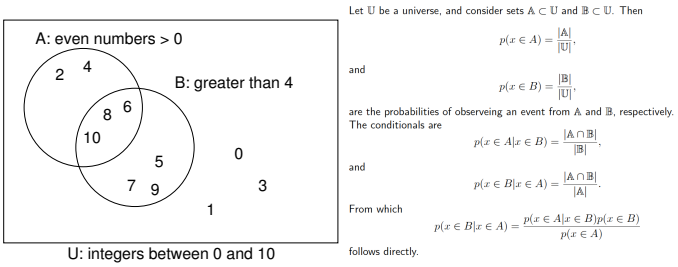


Figure 6: *Bayes Law, proven using sets, left is an example, right is the proof.*

3.2 Axioms of Probability

There are three axioms of probability.

1. All probability values lie between 0 and 1; $\forall x : p(x) \in [0, 1]$
2. The probability of mutually exclusive events is the sum of the individual probabilities. If $\mathbb{X}_i = \{x_{i1}, x_{i2}, ...\}$ is a subset of possible events, then $p(x_{i1}\ or\ x_{i2}\ or\ ...) = \sum_{x_{ij} \in \mathbb{X}_i} p(x_{ij})$ is the probability of observing an event from the subset.
3. The probabilities of all possible mutually exclusive events sum to unity; $1 = \sum_{i=1}^{N} p(x_i)$.

The first law says we can think of a probability as a function that maps events in a set to the real interval $[0, 1]$. The final law is an instruction to take all possible events into account: one of the possibilities will occur for certain. It also means that it is the ratio of probabilities that is of significance. The second law says that no matter which subset of events we consider, the probability of observing an event from the subset is the sum of individual probabilities.

The marble-in-bag analogy is useful to provide an example. Suppose we somehow knew the ratio of red, $green$, and $blue$ marbles is p_r, p_g, p_b, respectively. Then, each of these is a probability between 0 and 1. The probability of picking a marble that is red or one that is green is $p(x = red\ or\ x = green) = p_r + p_g$. And, we know that any marble picked will be one of the three colours, so $p_r + p_g + p_b = 1$.

3.2.1 Bayes' Law

In addition to the axioms just given, there is the very important *Bayes' law*, which can be written down as:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}. \tag{15}$$

A proof of Bayes? law is given in Figure 6, here we explain its terms and develop an intuition for them.

The first term to explain is $p(x|y)$. This is the probability that an event x will be observed given event y is observed. It captures correlations between events, and is an example of a *conditional probability*. Notice that the phrase 'given event y is observed' should not be taken to mean y has actually happened. The conditional probability $p(x|y)$ is the probability of x, if we assume y.

Likewise, the conditional probability $p(y|x)$ is the probability of y, if we assume x has occurred. In general, this will not have the same value as $p(x|y)$, in fact they are linked by Bayes' law. When used in the context of Bayes' law these conditional probabilities are given particular names; $p(x|y)$ is called the *likelihood* and $p(y|x)$ is the *posterior*.

The probability $p(y)$ is called the *prior*. It is the chance that event y is observed, and is therefore a measure of our confidence of the assumption used in the likelihood. The term $p(x)$ is called the *evidence* or *evidence density*.

There is a neat and very human way to understand these terms (which was related as an anecdote to me; I am not the source of this). Suppose you are to go on a blind date. As a Bayesian, at the end of the date you will want to know the probability of the event $y = date\ likes\ me$ given the evidence $x = [good\ conversation,\ smiles]$. That is, you want the posterior $p(y|x)$. Now your prospective date has never met you, but nonetheless there is some prior probability $p(y)$ that they will like you. During the date you will be observing evidence – events such as smiles, good conversation and so. This gives you the likelihood $p(x|y)$ that your date likes you. However your date may simply smile a lot and engage in good conversation anyway, this is the probability of observing the evidence regardless of whether they like you or not, $p(x)$. So Bayes' law allows you to combine the likelihood of the evidence, the chance they like you before they've ever met you, and the possibility they are just friendly anyway, to obtain the post-date chance of being liked.

Before continuing we should note that $p(x, y) = p(x|y)p(y) = p(y|x)p(x)$ is an alternative way to write Bayes' law. The term $p(x, y)$ is called the *joint* probability, which is a measure of the chance the x and y co-occur. In Figure 6 the joint $p(x \in A, x \in B)$ would be $|A \cap B|/|U|$. Also we should note that $p(x) = \int_{-\infty}^{\infty} p(x, y)dx$ is the *marginal*; an appropriate summation is used for discrete events.

3.3 Probability Distributions

In practice we need to consider not just individual probabilities but *distributions*. This means $p(x)$ is a function (over the variates). The function $p(x)$ specifies how the data are distributed (their density). Now, there is a subtlety, which is explained next.

If the random variates form a discrete set, then $p(x)$ is a true probability. The probability of throwing a particular number of a die is an example. The random variates are $\{1, 2, 3, 4, 5, 6\}$ and there is a some chance of each one, which is $1/6$, if the die is fair. It the die is weighted, then the probability of each face will not be uniformly distributed, but will sum to 1.

If the events are drawn from a continuous set we need a different approach. To see why consider having a die with an infinite number of faces – the probability of throwing any one face will always be zero! Another way to think about this is the probability of a dart landing at a location that is specified to infinite precision. The probability of landing on that very exact point is zero. However, we could divide the plane into many tiny regions and count the number of dart throws that land in each one. The density is then the number of darts per unit area; this is formally called the *probability density function* (pdf). By analogy with physical matter, the *probability mass* in a region is the density scaled by size of the region. Some authors distinguish between mass and density, using $P(x)$ for mass and $p(x)$ for density; many others do not.

The art of data modelling lies, in part, in choosing how to model a distribution. There are many standard density functions — too many to consider here. Some density functions deal with data whose value cannot fall below zero (*e.g* the *Gamma* distribution), others with angular data (*e.g.* the *von Mises* distribution), and yet others with objects such a matrices (*e.g.* the *Wishart*). Here, we will concentrate on two of the most important and commonly used distributions.

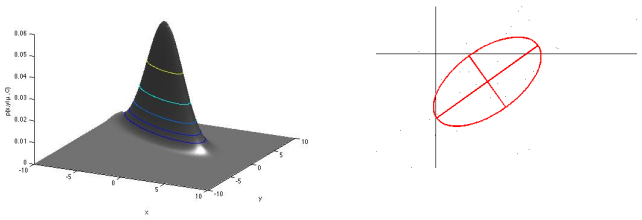


Figure 7: *A Gaussian in two dimensions: left as a surface, right as an elliptical contour showing* eigenaxes.

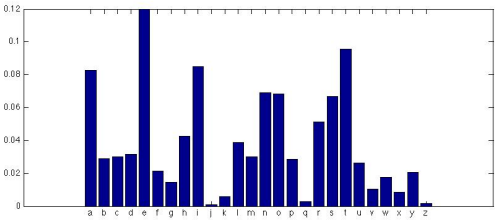


Figure 8: *A Multinomial Distribution.*

3.3.1 Normal, also called Gaussian Distribution

The Gaussian is a ubiquitous function, cropping up in many fields of study for many different reasons. In Machine Learning it is perhaps the most widely used probability density function, where it is called the *Normal* distribution. Defined for continuous data in K dimensional space the Normal distribution is specified by two parameters, a K-dimensional point μ and a $K \times K$ matrix C:

$$p(x|\mu, C) = \frac{1}{(2\pi)^{K/2}|C|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T C^{-1}(x-\mu)\right). \quad (16)$$

This is often written in short-hand form:

$$p(x|\mu, C) = \mathcal{N}(x|\mu, C). \quad (17)$$

The Normal distribution is the classic 'bell curve'. The μ is the point at the peak of the bell, and C is a covariance matrix that controls the shape of the bell. Given a data set, the mean and covariance are easy to compute, as in Equations 11 and 12, respectively. If we are to be exact, we should call these the *sample mean* and *sample covariance* because we have computed them from samples – we have no access to the full population.

The Normal distribution is a density, and the value of $\mathcal{N}(x|\mu, C)$ can exceed 1; this does not break any axoim of probability. This is because we should scale by the size of region dx to get the probability mass, and it is the probability masses that sum to unity $\int_{-\infty}^{\infty} \mathcal{N}(x|\mu, C)dx = 1$.

The covariance, C, is a symmetric matrix, as seen in Equation 13. If C is the identity matrix, then the 'Gaussian bell' (so to speak) is isotropic; contours of equal density make concentric circles about the peak at μ. If C is diagonal, the values on the diagonal act to squash the bell so that the contours become ellipses. If off-diagonal terms are non-zero, these rotate the bell so that contours are not axis aligned. More generally we can always write C as

$$C = ULU^T \quad (18)$$

in which U is an orthonormal matrix so $UU^T = U^TU = I$, the identity, and L is diagonal. The columns of $U = [u_1, u_2, ...]$ are *eigenvectors* that point along the axes of the elliptical contour. For each vector u_i there is a corresponding *eigenvalue* in L, call this λ_{ii}. This value is the variance of the data in the direction of the axis, and λ_{ii} is the standard deviation in that direction. It is the U matrix that turns the distribution, and the L matrix that scales it along its main axes.

The term inside the exponential is $(x-\mu)C^{-1}(x-\mu)$. This is the square of the *Mahalanobis distance*. This is the distance of the point x from centre μ, but measured in terms of the number of standard deviations.

The term outside the exponential $1/((2\pi)^{K/2}|C|^{1/2})$ is a *normalisation factor* used to ensure the distribution integrates to unity. It is this factor that turns the Gaussian function into a Normal distribution. This factor is an indication of the hyper-volume of the particular distribution. In fact $|C|^{1/2} = \prod_{i=1}^{K} \lambda_{ii}^{1/2}$, which is just the volume of a cube whose lengths are standard deviations.

3.3.2 Multinomial

The *multinomial* distribution is used for discrete events. An intuitive and useful way to think about a multinomial distribution is as a histogram that has been scaled so that it sums to unity. Figure 8 shows a multinomial distribution constructed from a histogram of letters in an English document.

The multinomial probability density function gives the probability of observing a histogram, given a vector of mean values. For vectors of length K the probability of observing a histogram with entry values $(m_1, m_2, ...)$, given a vector of means, $(\mu_1, \mu_2, ...)$ is

$$p(m_1, m_2, ...m_K|\mu_1, mu_2..., m_K) = \frac{\sum_{i=1}^{K}(m_i)!}{\prod_{i=1}^{K} m_i!} \prod_{i=1}^{K} \mu_i^{m_i}. \quad (19)$$

The product $\prod_{i=1}^{K} \mu_i^{m_i}$ appears because the probability of a sequence of independent events is just the product of their individual probabilities. The normalising term in front of the product counts the number of ways to partition the $M = \sum_{i-1}^{K} m_i$ objects into K groups; one size m_1, another of size m_2, and so on.

The multinomial distribution is extensively used in areas such as document analysis, where the probability distribution over words can be used to characterise the type of a document as, say 'art', or 'science'. We'll find use for it in modelling trees.

3.4 ML and MAP solutions

We have already seen three problem areas for Machine Learning: regression, density estimation, and classification. Each of these problems can be addressed by solving a Maximum Likelihood (ML) or Maximum A-Posterior (MAP) problem, we will not consider the 'fully Bayes' approaches.

We will illustrate the difference between ML and MAP using the regression of a polynomial as an example, so $\mathbb{D} = \{(x_i, y)\}_{i=1}^{N}$. Before continuing we note that a polynomial

$$y = w_0x^0 + w_1x^1 + w_2x^2 + ... \quad (20)$$

can be written as $\mathbf{w}^T\mathbf{x}$, in which $\mathbf{w} = [w_0, w_1, ..., w_M]$ and $\mathbf{x} = [x^0, x_1, ..., x^M]$; a notational convenience we will make use of.

3.4.1 Maxium Likelihood

Maximum likelihood (ML) solutions solve the problem

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(\mathbb{D}|\theta). \tag{21}$$

This means the program must find the parameter θ that maximises the likelihood of the data, $p(\mathbb{D}|\theta)$. Within the context of regression the probability of the data is

$$p(\mathbb{D}|\theta) = \prod_{i=1}^{N} p(x_i|\theta). \tag{22}$$

By assumning errors in the observed data are Gaussian distributed around the true line (justified by the *central limit theorem*) a functional form for the likelihood is fixed as a Gaussian. So that

$$p(\mathbb{D}|\theta) \quad = \quad \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{(\mathbf{w}^T\mathbf{x}_i - y_i)^2}{\sigma^2}\right). \tag{23}$$

the parameter θ is used to represent the polynomial coefficients $\mathbf{w}$. Taking logarithms we get

$$\ln p(\mathbb{D}|\theta) = N \ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2. \tag{24}$$

The maximum-likelihood solution is therefore found by minimising

$$E = \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2, \tag{25}$$

which is the classic 'least-squares' problem.

3.4.2 Maximum A-Posterior

Maximum-A-Posterion (MAP) solutions solve the problem

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(\theta|\mathbb{D}). \tag{26}$$

This differs from the maximum likelihood solution because it requires a prior over the parameters, $p(\theta)$ so that the posterior $p(\theta|\mathbb{D})$ can be maximised. Following Bayes' law we have

$$p(\theta|\mathbb{D}) \quad \propto \quad p(\mathbb{D}|\theta)p(\theta) \tag{27}$$

$$\quad = \quad p(\theta)\prod_{i=1}^{N} p(x_i|\theta). \tag{28}$$

Taking logarithms we obtain

$$\ln p(\theta) + \sum_{i=1}^{N} \ln p(x_i|\theta). \tag{29}$$

Now returning to the case of regression, we can immediately substitute the least squares term $\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$ for $\sum_{i=1}^{N} \ln p(x_i|\theta)$. We can choose the prior, $p(\theta)$, as we please – except that it cannot depend in any way on the data in $\mathbb{X}$. A typical way to set up the prior is

$$p(\mathbf{w}) = \frac{\kappa}{\sqrt{2\pi}} \exp(-\frac{1}{2}\lambda|\mathbf{w}|^2). \tag{30}$$

By taking logrithms substituting into Equation 29, and considering only those terms that effect the MAP problem we get

$$E = \sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda|\mathbf{w}|^2 \tag{31}$$

as the objective function to minimise, it includes a regularisation term. The value of regularisation becomes apparent when fitting a polynomial of unknown order: regularisation prevents over-fitting; Bishop [2006] explains in greater detail.

4 Helping Non-Artists Draw

Limpaecher *et al.* [2013] describe how to construct an app for mobile phones that helps users trace over photographs of a friend or a celebrity. The idea is to correct a user's stroke as it is drawn (via the touch screen). Correction is possible because the app learns from the strokes of many different people; this is an example of *crowd-sourcing*.

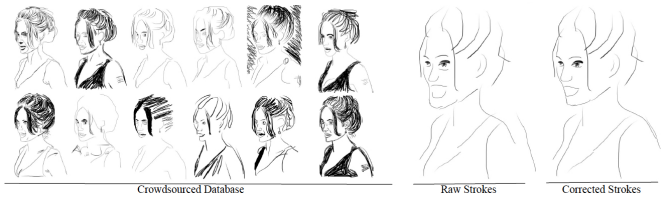


Figure 9: *Many drawings (left) create a repository of strokes so that a new user's input can be corrected. Used by Permission.*

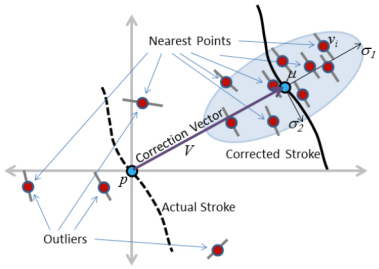


Figure 10: *Finding the nearest crowd-sourced points. Used by Permission.*

The app has as input a user's stroke and a database of strokes collected by crowd-sourcing. The crowd-sourced strokes are assumed to be drawn over the same photograph that the user is currently tracing. Every stroke comprises a set of points. The algorithm corrects a user's stroke in two steps: (i) for each point on the user's stroke, suggest a corrected point μ; (ii) correct the stroke as a whole, using all the suggested points.

We'll consider suggesting a correction point first. Let z be a point on the user's stroke for which a suggestion is to be made. Let x_i be a crowd-sourced point. To make the mathematics a little easier, we will follow Limpaecher *et al.* by shifting all the points so that z is at the origin, then $x_i \leftarrow x_i - z$. Figure 9 depicts the sitation after this shift – the point z is at the origin.

The suggested point is an expectation value:

$$\mu \quad = \quad \sum_{i=1}^{N} p(x_i)x_i, \tag{32}$$

$$p(x_i) \quad = \quad \frac{w_i}{\sum_{j=1}^{N} w_j}. \tag{33}$$

The key to making a good suggestion is to chose the weights w_i properly. The weights are computed on an iterative basis. They are

initialised as the likelihoods of an isotropic Gaussian:

$$w_i = \mathcal{N}(x_i|0, sI), \tag{34}$$

in which $s = 1/N \sum_{i=1}^{N} ||x_i||$ is the mean distance of the x_i points from the origin and I is the identity matrix. Since the covariance C is just a (positive) scalar multiple of the identity it has a circular cross section.

After initialising weights, a putative suggested point is computed, as in Equation 32. The current suggestion and current weights are used to estimate the parameters for a new Gaussian, this time non-isotropic with an important constraint placed on its orientation. Limpaecher *et al.* argue that (i) the direction of the line at x_i is orthogonal to x_i, and (ii) the closest points will have broadly similar directions. Consequently the closest points of interest can be expected to be distributed around a line passing through the origin.

This argument could be used to place a prior over the parameters of a Gaussian, which would allow a MAP fitting problem, that is solve $p(\mu, C|\mathbb{D}) \propto p(\mathbb{D}|\mu, C)p(\mu, C)$. In this case the prior decomposes, $p(\mu, C) = p(\mu|C)p(C)$; $p(\mu|C)$ is taken to be Gaussian and a *Wishart* distribution [1928] is placed over the covariance C.

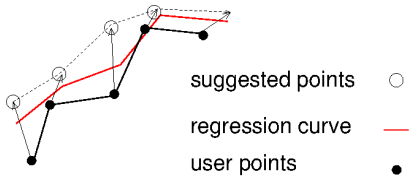


Figure 11: *Using regression to correct a user-stroke as a whole.*

Instead, Limpaecher *et al.* [2013] constrain the Gaussian explicitly. In particular, one of its axes, u_1 points directly to the origin. Since μ is the centre of the Gaussian we have $u_1 = \mu/||\mu||$. The vector u_2 is at right-angles to u_2. The standard deviation over these axes is specified to be

$$\sigma_1 = \sqrt{\sum_{i=1}^{N} p(x_i)(u_1^T(x_i - \mu))^2} \tag{35}$$

$$\sigma_2 = \sqrt{\sum_{i=1}^{N} p(x_i)(u_2^T x_i)^2} \tag{36}$$

The term $u_1^T(x_i - \mu)$ projects shifted crowd-sourced points onto the unit vector u_1; $u_2^T x_i$ is also a projection, but under the assumptions that have been made no shift is required. Again, these are expectation values, and since they are not estimated by a posterior distribution they are maximum likelihood estimates.

We now have enough information to re-compute the weights using the new Gaussian

$$w_i = \mathcal{N}(x_i|\mu, C) \tag{37}$$

in which

$$C = [u_1 u_2] \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} [u_1 u_2]^T \tag{38}$$

is the covariance. The weights are modified: $w_i \leftarrow w_i/(w_i+0.05)$. The iteration continues until the mean stops moving.

Limpaecher *et al.* describe the algorithm for computing a suggested location as a "modified mean shift". The mean shift algorithm [Comaniciu and Meer 2002] comes from non-parametric density estimation. It is a form of search known as *hill climbing* in the sense that a given point in space is continually shifted towards its local density maximum, until it reach the maximum where shifting stop (in practice, when the shift steps are small enough).

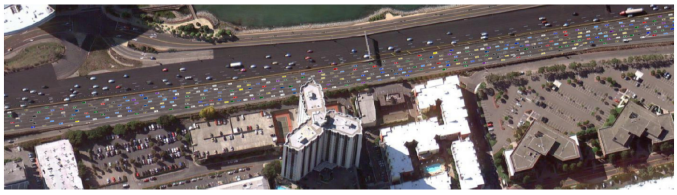


Figure 12: *Traffic flowing on a road. Used by Permission.*

Recalling the aim of the app is to correct a user's stroke, it may seem that all that remains is to compute a suggested new point μ_i for each point z_i on the user's stroke. Simply shifting the points, though, runs the risk of losing the wiggles and turns that make a user's stroke individual. Limpaecher *et al.* overcome this by regression of the whole line at once. The objective function is

$$E = \sum_{i=1}^{N}(x_i' - (x_i + \mu_i)) + \sum_{i=2}^{N}((x_i' - x_{i-1}') - (x_i - x_{i-1})), \tag{39}$$

which is minimised by finding the set of new locations x_i'; depicted in Figure 11. When compared with regression above we see that this is a maximum likelihood solution, because there is no prior placed on the location of the new locations.

Machine Learning is important to Limpaecher *et al.* but is by no measure the only component in their app. The full paper describes how to create a vector field that corrects strokes so efficiently that a real time app is possible, and the whole solution is embedded within a game that encourages people to use the app. As a whole, this example is an excellent demonstration of how to build something that is fun, useful, and innovative using a few tools from Machine Learning.

5 Data Driven Animation of Traffic

We turn now to animation. Games and films have benefited from human motion capture in recent years. The paper we study here, by Wilkie *et al.* [2013], uses Machine Learning to include real-world data to drive a traffic flow simulation that is convincing. These factors make for a complicated simulation with many free parameters – which makes it difficult to control.

There are three parts to their system, as indicated in Figure 13. The first is the sub-system that makes raw velocity and density measurements of real traffic flow. Following the authors, we will assume these measures exist and are accurate. Part three is a traffic-flow simulator that requires many parameters to be set correctly, if it is to be convincing. Part two links the raw data and the simulator; it is the Machine Learning part, called *state estimation*, and so is the part we focus on.

Ensemble Kalman Smoothing, or EnKs [2007], lies at the heart of the learning algorithm used by Wilkie *et al*. EnKS is *state estimator* that extends the *Kalman filter* [1960] to many bodies. We'll start our explanation with the Kalman filter.

The Kalman filter is a well known method used to control the state of a system. It is used in many different engineering fields, temperature control is one example. It can be regarded as a predictor-corrector integrator; a simulation makes a prediction as the next state of some system, which is corrected given an observation. The visual image is of a particle moving along a trajectory, its next step

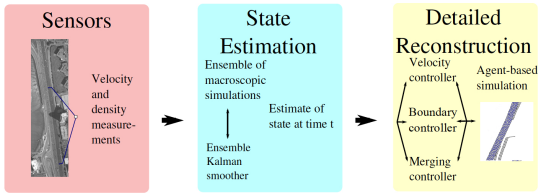


Figure 13: *Three main components to the system of Wilkie* et al. *[2013]. Used by Permission.*

is first predicted from previous steps and then corrected by comparing the prediction with observations.

We will develop our intuition of the Kalman filter by briefly considering its application in Computer Vision, where it is used to track objects in video. The idea is that an object (such as a vehicle or a person) being tracked has a *state*, here denoted x. For tracking, state is normally a position and velocity but sometimes includes acceleration. The state (or at least some of it) is often not directly observable, for example the velocity and acceleration of an object must be inferred from video from an *observation*, here denoted z. The observation and state are related; $z = H(x)$, for some function H.

If the state (of a real system) were known exactly, then the state at the next time instant could be predicted exactly. Newton's laws of motion are often used to make a prediction, but any suitable model will suffice. However, the state is usually not known exactly, so the prediction will be uncertain. In fact the state is assumed to be Gaussian distributed. This means that in principle the system could be in any state, but some are more likely than others and the likelihoods are given by a Gaussian, so $p(x) = \mathcal{N}(x|\mu, C)$, for some μ and C. This is *process noise*, which written in the mathematics of Machine Learning is

$$x_t = f(x_{t-1}, m_t), m_t \sim \mathcal{N}(0, I). \tag{40}$$

This is read to mean that the new state , x_t, is a function, f of the old state, x_{t-1} and noise, m_t; and that the value of noise is drawn at random from an origin-centred Gaussian with isotropic unit covariance.

There is a second source of uncertainty, which arises from measurements, that is from attempts to locate the object in the video frame. This error is also assumed to be Gaussian distributed and is called the *measurement noise*. This is written

$$z_t = h(x_t, n_t), n_t \sim \mathcal{N}(0, I), \tag{41}$$

and just as Equation 40 captures the uncertainty in a single state, this captures the uncertainty in a single observation. The Kalman filter combines the uncertainties to correct the prediction,as illustrated in Figure 14. The corrected prediction is then used to update the state, and the iteration advances by one step.

Ensemble Kalman Smoothing is a generalisation of the Kalman filter that is suitable for multiple objects. Wilkie *et al.* [2013] use a state defined by dividing each traffic lane into N parts. State is then a vector:

$$x = (\rho_1, y_1, \rho_2, y_2, ..., \rho_N, y_N, \gamma, \nu_{max}). \tag{42}$$

Each ρ, y pair refers to the density and velocity of traffic in one of the N cells along a traffic lane; γ is a parameter that relates velocity to density, and ν_{max} is the maximum velocity. The state describes the traffic flow in a particular lane, at a particular instance in time, as depicted in Figure 15. To model traffic flow over several lanes we

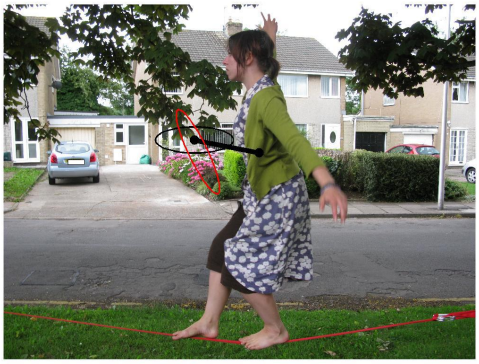


Figure 14: *The Kalman Filter predicts an object's motion (black) and corrects it using observation data (red); both are uncertain. Simulated data is used in this example.*

need several states; so x_t^i is used to denote the state of lane i at time t. An ensemble for a road of M lanes is then $\mathcal{X} = \{x_t^1, x_t^2, ...x_t^M\}$.

Kalman filtering predicts the next state of a single object; EnKS predicts the next state of an ensemble. The predictor can be any function that suits the application. In their paper Wilkie *et al.* use a traffic simulator designed by Aw and Rascale [Aw and Rascale 2000] and Zhang [Zhang 2002]; it is called 'ARZ'. This particular simulator was selected as it is a good model of traffic flow, but the EnKS and therefore the method of Wilkie *et al.* do not depend upon it. We only need to know that the simulator is a function call that predicts the next state given the current state, and the prediction is corrected by Ensemble Kalman Smoothing. The EnKS algorithm is given in Figure 16

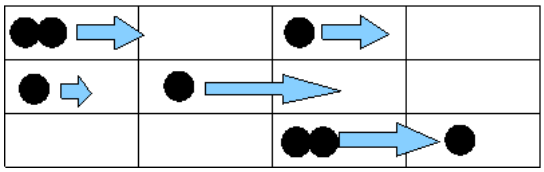


Figure 15: *State of traffic flow for a three lane road. Each lane divides into cells, and the density of traffic and velocity of traffic in each cell comprise the state.*

Looking at the algorithm we see it is a predictor-corrector. At each time step it first uses the simulator to predict both the new state, x_t^i and a simulated observation z_t^i in each cell ("simulated observation" is our term). Once all cells have been updated the algorithm switches from prediction to correction.

For correction to be meaningful it must relate the real measurements to the simulated state x via the simulated observations, z. Clearly, the simulation should be adjusted so that deviation between real and simulated observations are minimised. More exactly, the distribution of simulated observations (which is a consequence of the distribution of states) should match the distribution of real observations.

Correction begins by computing the mean simulated observation,$\hat{z}_t$, over all lanes at the given time instant. Notice that this mean is a vector. The covariance Σ_t over simulated observations is computed at the same instant. These computations assume that simulated observations are Gaussian distributed; that is $p(z) = \mathcal{N}(z|\hat{z}_t, \Sigma_t)$. Since $\hat{z}_t$ and Σ_t are all that is needed to characterise the distribution they are called *sufficient statistics*. We can now imagine the

Algorithm 1: EnKS for Traffic State Estimation

Input: Traffic sensor measurements $z_1 ... z_{t_n}$, ARZ simulator in f, observation model in h, initialized ensemble $\mathbf{x}_0^0 ... \mathbf{x}_0^{m-1}$, noise vectors $\mathbf{m}_t^0 ... \mathbf{m}_t^{m-1}$ and $\mathbf{n}_t^0 ... \mathbf{n}_t^{m-1}, \forall t \in 1...t_n$

Output: Traffic state estimates $\hat{\mathbf{x}}_t, \forall t \in 1...t_n$

for $t \in 1...t_n$ **do**
  for $i \in 0...M-1$ **do**
    `// Motion model`
    $\mathbf{x}_t^i \leftarrow f(\mathbf{x}_{t-1}^i, \mathbf{m}_t^i)$;
    `// Observation model`
    $\mathbf{z}_t^i \leftarrow h(\mathbf{x}_t^i, \mathbf{n}_t^i)$;
  `// Analysis`
  $\hat{\mathbf{z}}_t \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{z}_t^i$;
  $\Sigma_t \leftarrow \frac{1}{M-1} \sum_{i=0}^{M-1} (\mathbf{z}_t^i - \hat{\mathbf{z}}_t)(\mathbf{z}_t^i - \hat{\mathbf{z}}_t)^T$;
  for $j \in 1...t$ **do**
    $\hat{\mathbf{x}}_j \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{x}_j^i$;
    $\Gamma_j \leftarrow \frac{1}{M-1} \sum_{i=0}^{M-1} (\mathbf{x}_j^i - \hat{\mathbf{x}}_j)(\mathbf{z}_t^i - \hat{\mathbf{z}}_t)^T$;
    $K_j \leftarrow \Gamma_j \Sigma_t^{-1}$;
    for $i \in 0...M-1$ **do**
      $\mathbf{x}_j^i \leftarrow \mathbf{x}_j^i + K_j(\mathbf{z}_t - \mathbf{z}_t^i)$,;
`// Final output`
for $t \in 1...t_n$ **do**
  $\hat{\mathbf{x}}_t \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{x}_t^i$;

Figure 16: *An algorithm for Enemble Kalman Smoothing. Used by Permission.*

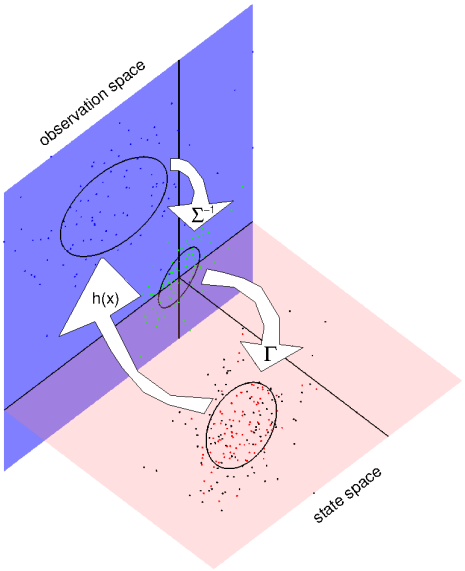


Figure 17: *Mappings inside the EnKS algorithm, schematically illustrated. State vectors (red points) are distributed in 'state space' (pink) under a Gaussian (ellipse). Observations are distributed in 'observation space' (blue), also under a Gaussian. A whitening transform, Σ^{-1} carries observations to a sphere at the origin (green points), and a covariance, Γ, carries whitened observations to the state distribution (black points). The Kalman Gain, K, maps points in observation space directly into state space.*

simulated observations as a kind of squashed fuzzy ball in high-dimensional space.

Next the mapping between the distribution of states and the distribution of simulated observations is computed. This is done at every time step from the start to the current time (which is the smoothing part of EnKS). First the mean state $\hat{x}_j$ is computed at time step $j \leq t$, notice this is over all lanes and so is a vector. Next the covariance, Γ_j, that links the distribution of simulated observations with the distribution of states is computed. The covariance Γ_j is a matrix, and we can imagine it as a transform that maps the space of simulated observations into the space of states. Finally, the *Kalman Gain* matrix $K_j = \Gamma_j \Sigma_t^{-1}$ is computed. The Σ_t^{-1} transforms the squashed fuzzy ball into a spherical fuzzy ball (in high dimensions), this is an example of a *whitening transform*. The Γ_j then transforms this hyper-sphere into state space. These mappings are schematically illustrated in Figure 17.

The Kalman gain, K_j, is used to correct the state of the ensemble using real data. The term $K_j(z_t - z_t^i)$ transforms real observations $z_1, z_2, ...z_{t_n}$ into state space, assuming that the centre of the real distributions is at the simulated observation point z_t^i. The transformed vector is a displacement in state space that is added to the state x_j^i.

In this way the state of the traffic flow simulation is updated using real data. Of course, the 'real' data need not be real – it could be supplied by an artist, for example. The rest of Wilkie *et al.* describes the traffic simulator, and how to use the output of the simulator to make Computer Graphics (broadly, put models of vehicles in the lane cells, and render them). Since we have now covered the Machine Learning elements of the paper, we'll move on to our next example.

6 Modelling Trees

Figure 18: *From video to many trees, via Machine Learning.*

Our final example shows how create models of many moving trees using video as source material. It comes from Li *et al* [Li et al. 2011]. The motivation in this case is that trees are very hard to model, and even harder to animate. Li *et al.* propose a method to capture 3D trees that move in 3D, and which can be used to spawn new individual trees of the same type. The only input to their system is a single video stream, and the only user input is a rough sketch around the outline of the tree in frame one. Figure 18 shows a group of trees that has been produced from two input video.

There is a significant amount of Computer Vision required that segmented the tree from its background and tracks the leaves of the tree as they are blown by wind. Further processing is required to use the tracking information to infer a branching structure for the tree, which is obscured by the leaves being tracked. (If the tree is bare of leaves a method to track the branches would replace this pre-processing). We pick up the paper at the point of converting a 2D skeleton that moves with the 2D tree as seen in the video into a 3D tree.

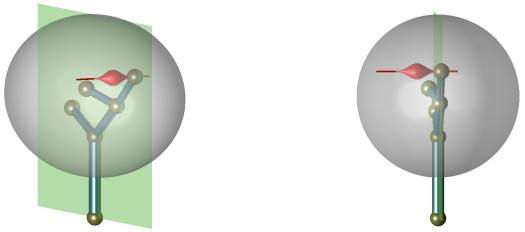


Figure 19: *Pushing a node, probably. A node is pushed out of the plane (green) into a user defined volume (white). The location of the node is probabilistically selected based on a distribution over the line of push (red).*

Figure 20: *Probabilistic pushing (right) is used to balance the dual constrains maintaining branching angles within reasonable limits (left), while optimally filling the volume (middle).*

The idea underlying the conversion from 2D to 3D is simple enough. It comes down to 'pushing' tree nodes away from the plane into a volume, as in Figure 19. In doing so there are two competing demands to be met. The first is to optimally fill the volume into which the tree must 'grow', the second is to prevent tree branches from being bent into implausible shapes, as in Figure 20. Li *et al.* balance these demands using Bayes' law, as in Equation 15. The specific form used in this tree modelling example is

$$\begin{aligned} p(x_i|X_{i-1},\Omega) &= \frac{p(\Omega|X_{i-1},x_i)}{p(X_{i-1},\Omega)}p(X_{i-1},x), \\ &= \frac{p(\Omega|X_{i-1},x_1)}{p(X_{i-1},\Omega)}p(X_{i-1}|x_i)p(x_i). \quad (43) \end{aligned}$$

We want x_i, the location of a branch node in 3D. The X_{i-1} represents all of the tree below that particular node, and Ω is the volume the tree is growing into. A greedy algorithm is used: the x_i with the largest probability value $p(x_i|X_{i-1},\Omega)$ is selected as the 'push' position – this is a MAP solution.

To see how this probabilistic approach is useful in tree growing, notice that on any given 'push' we can safely ignore any term that does not contain x_i. Therefore we need only consider $p(\Omega|X_{i-1},x_i)$ and $p(X_{i-1}|x_i)$. The first term governs the growth of the tree into the volume Ω. There are points placed on the surface of the volume that x_i is attracted to – it is attracted to the nearest points most strongly. The term $p(X_{i-1}|x_i)$ is used to restrict the push of x_i away from the branches below it so that the resulting branch is not bent out of shape. The reader is directed towards the original paper for details.

Motion in 3D is reconstructed by taking advantage of foreshortening. A branch that sways towards or away from the camera plane is foreshortened by a predictable amount that depends on the angle of sway. In practice the noise in measurements makes a deterministic approach unreliable, but the prediction is the basis for a probabilistic approach to motion reconstruction. This probabilistic approach follows exactly the same form as growing except that push distance x_i is replaced with a 3D rotation, and the rotation of the tree below the current node is taken into account too. Again the reader is referred to the original paper.

Rather than provide further details of reconstruction, we will turn to consider the use of Machine Learning methods to make new trees from old. To do this we return to the 2D skeleton that has been extracted from a video by Computer Vision methods, and focus on individual *bifurcations* – locations where one branch splits into two. At the node of a bifurcation there are always three branches, and hence three angles separating them. These three angles can be regarded as an example of the multinomial distribution we looked at in Section 3.3.2. In particular, the angles must sum to 2π. So, if we know the angles $[\phi_1,\phi_2,\phi_3]$, then scaling this histogram by $1/(2\pi)$ yields a multinomial distribution.

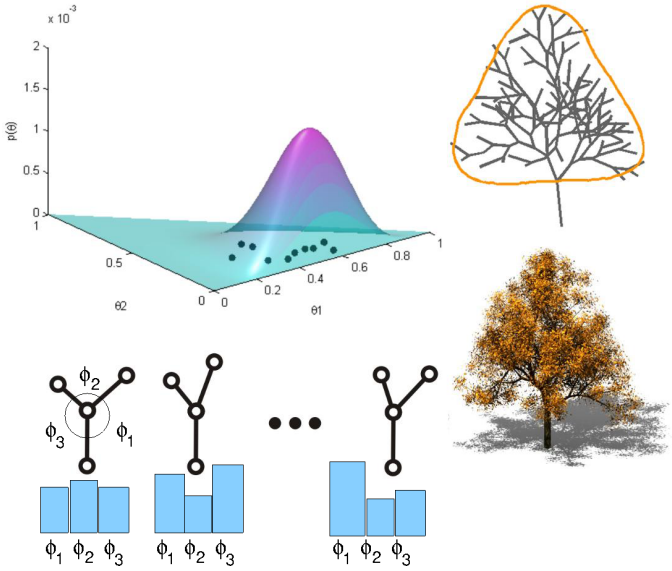


Figure 21: *Creating new trees using probabilistic rules. Angles bifurcations (middle) sum to 2π, so can be represented as multinomials. The multinomials (bottom) are considered as random samples from a Dirichlet distribution (top). Sampling from the Dirichlet allows a 2D skeleton to be grown, which is converted into 3D (right, top and bottom).*

Now, there is a multinomial for every bifurcation in the tree. Li *et al.* assume that for a tree of a given type these multinomials will be similar. For example, in a tall thin tree the angle opposite the local root will usually be narrow, whereas in a broad flat tree the same angle will usually be wide. The problem is to characterise the distribution of these multinomials for a given tree. To do this Li *et al.* use regression to fit a *Dirichlet* distribution to them [Minka 2000]. The Dirichlet distribution is defined by

$$p(\mu_1,\mu_2,...|\alpha_1,\alpha_2,...) = \frac{\Gamma(\sum_{i=1}^K \alpha_i)}{\prod_{i=1}^K \Gamma(\alpha_i)} \prod_{i=1}^K \mu^{\alpha_i - 1}, \quad (44)$$

in which $[\alpha_1,\alpha_2,...]$ are the distribution parameters, and $[\mu_1,\mu_2,...]$ is a multinomial distribution. The Gamma function Γ is akin to the factorial in the multinomial, in fact $\Gamma(x+1) = x!$.

In addition to multinomials of angles, Li *et al.* also make multinomials over branch length, in this case forcing the lengths of the

three branches at a bifurcation to sum to 1. A second Dirichlet distribution is fitted to the multinomials for branch length.

Clearly, the Dirichlet distribution is closely related to the multinomial distribution, in fact a draw from a Dirichlet distribution is a multinomial that belongs to a particular class (in this case, type of tree). This exactly what is needed here – it provides Li *et al.* with a *probabilistic generative model* that can be used to make new individual examples belonging to the same class as the original data.

The generative model for bifurcations is put to work to make new individual trees. Starting from the trunk, bifurcations are added one by one, using exactly the same Bayesian approach as for pushing a point into 3D. At each step, a bifurcation belonging to the family for the tree is created using the pair of Dirichlet distributions. It is scaled to a size that depends on its depth in the tree (which may also be learned), rotated, and placed amongst the existing branches. Tree motion is also generated using a Dirichlet distribution. In this case the multinomials are over the energy distributions in the resonant frequencies of angular velocity.

The use of a probabilistic generative model replaces the need for hand-crafted rules (as in L-systems), does not require users to draw a tree (although the system could learn from drawings), nor does it require a new photograph or video for each new tree. Instead the learning of probabilistic rules is completely automatic and hidden from the user.

7 Where to Learn More

There is plenty of material to support the Computer Graphics researcher wanting to use Machine Learning in their book. Chris Bishop's book [2006] is an excellent general reference for Pattern Recognition and Machine Learning, as Duda, Hart, and Stork[2012]; both are regarded as classic texts.

Allied fields, notably Computer Vision, use Machine Learning extensively. Papers in top conferences such as CVPR, ICCV, and ECCV are rich in its use. The Machine Learning community publish at conferences such as ICML and NIPS.

Acknowledgements

Thanks to Qi Wu for producing Figure 5, and to Dmitry Kit for having the patience to proof-read the script. Thanks too to all of the authors of the SIGGRAPH papers used to illustrate this course. And last but not least, thanks to the EPSRC funding agency who support much of my work.

References

AW, A., AND RASCALE, M. 2000. Ressurection of "second order" models of traffic flow. *SIAM Journal on Applied Mathematics*, 916–938.

BISHOP, C. M., ET AL. 2006. *Pattern recognition and machine learning*, vol. 1. springer New York.

COMANICIU, D., AND MEER, P. 2002. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 24*, 5, 603–619.

DEMPSTER, A. P., LAIRD, N. M., RUBIN, D. B., ET AL. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal statistical Society 39*, 1, 1–38.

DUDA, R. O., HART, P. E., AND STORK, D. G. 2012. *Pattern classification*. John Wiley & Sons.

EVENSEN, G. 2007. *Data assimilation*. Springer.

KALMAN, R. E. 1960. A new approach to linear filtering and prediction problems. *Journal of basic Engineering 82*, 1, 35–45.

LI, C., DEUSSEN, O., SONG, Y.-Z., WILLIS, P., AND HALL, P. 2011. Modeling and generating moving trees from video. In *SIGGRAPH Asia*.

LIMPAECHER, A., FELTMAN, N., TREUILLE, A., AND COHEN, M. 2013. Real-time drawing assistance through crowdsourcing. *ACM Transactions on Graphics (TOG) 32*, 4, 54.

MINKA, T., 2000. Estimating a dirichlet distribution.

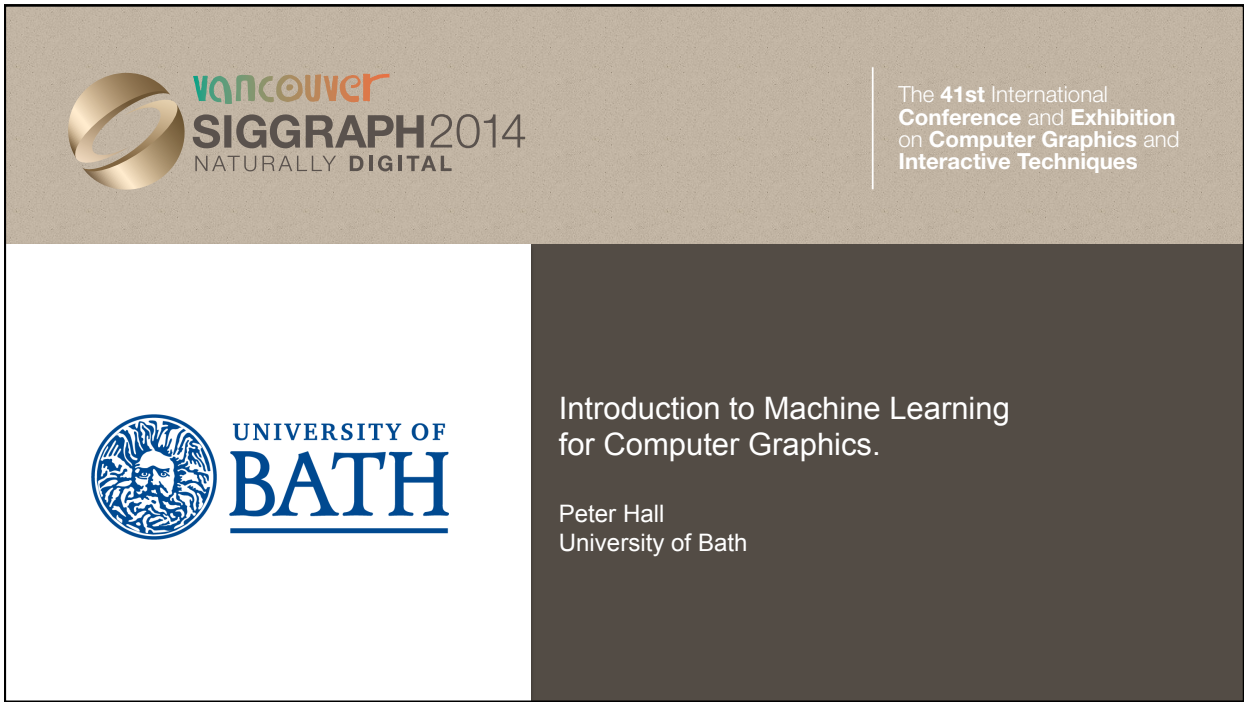
WILKIE, D., SEWALL, J., AND LIN, M. 2013. Flow reconstruction for data-driven traffic animation. *ACM Transactions on Graphics (TOG) 32*, 4, 89.

WISHART, J. 1928. The generalised product moment distribution in samples from a normal multivariate population. *Biometrika*, 32–52.

WU, Q., AND HALL, P. 2004. Prime shapes in natural images. *Computing 22*, 10, 761–767.

ZHANG, H. 2002. A non-equilibrium traffic model devoid of gas-like behaviour. *Transport Research Part B: Methodoligical 36*, 3, 275–290.

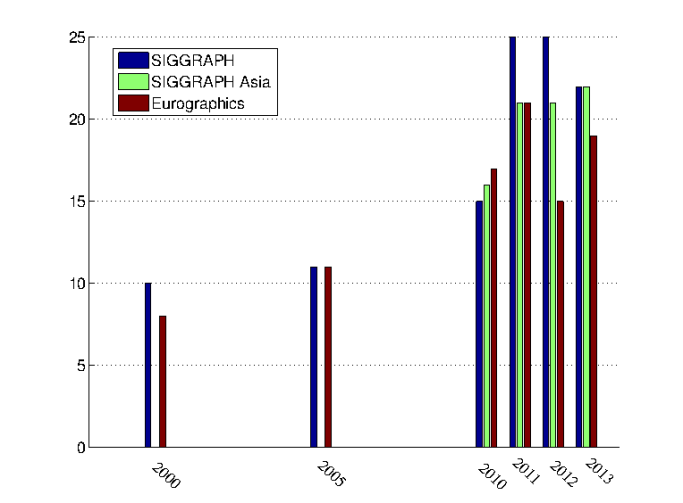

SIGGRAPH2014
NATURALLY DIGITAL
vancouver


vancouver
SIGGRAPH2014
NATURALLY DIGITAL
The 41st International Conference and Exhibition on Computer Graphics and Interactive Techniques
UNIVERSITY OF BATH
Introduction to Machine Learning for Computer Graphics.
Peter Hall
University of Bath

Machine Learning in Computer Graphics

- Why?
  - complexity of models
- Where?
  - ubiquitous
- How?
  - this course!

The ML trend in CG



Examples From SIGGRAPH 2014

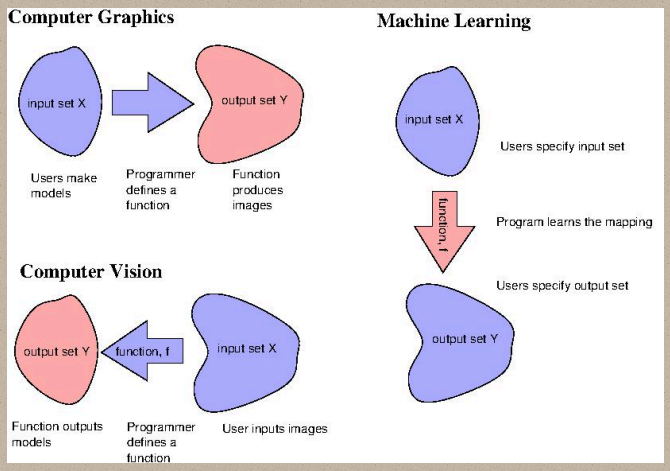
Shape Collection
Controlling Character
Typography and Illustration

Geometry Processing
Light Transport
Reflectance: modeling, capturing, rendering
Image Tricks
Shady Images
Fluids

Case Studies in this Course

- Helping Non-Artists to Draw
  - Limpacher *et al*., Real-time drawing Assistance Through Crowd Sourcing. SIGGRAPH 2013
- Animating Traffic Flow
  - Wilkie *et al*., Flow Reconstruction for Data-Driven Traffic Animation. SIGGRAPH 2013
- Automated Production Rules for Moving Trees
  - Li *et al*., Modeling and Generating Moving Trees from Video. SIGGRAPH Asia 2011

ML and Data Driven Modelling



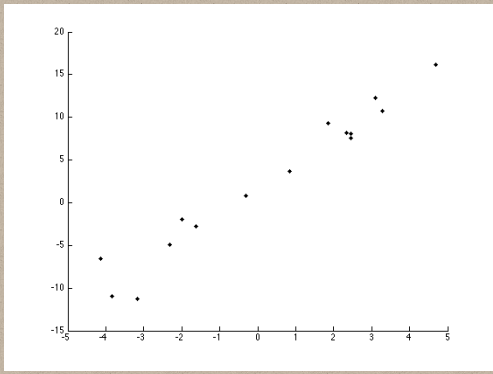
Given
- domain X
- range Y

Compute
- function f

Such that
- y = f(x)

Regression

Find parameters, θ, for a function, y = f(x,θ)

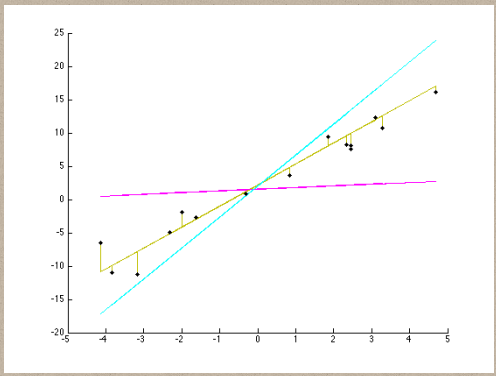


standard example: a straight line; parmeters θ = (m,c).

$$y = mx + c$$

Regression

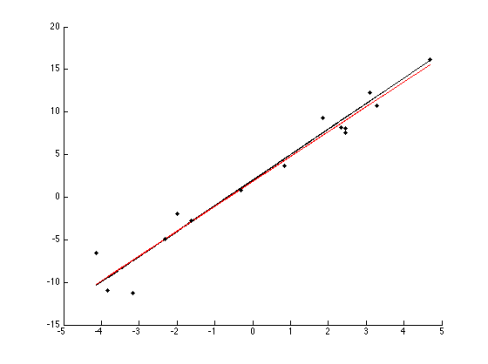
Find parameters for a function y = f(x,θ)



which line?

Regression

Find parameters for a function y = f(x,θ)



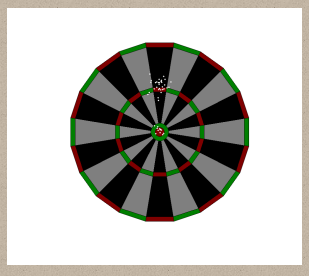
$$E(m,c) = \sum_{i=1}^{N} (mx_i + c - y_i)^2$$

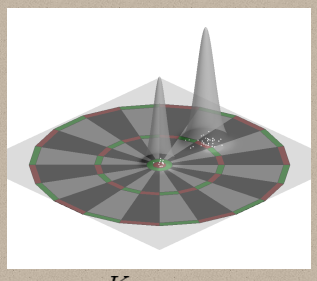
Using Regression

Collomosse and Hall, Motion Analysis in Video: Dolls, Dynamic Cues, and Modern Art. Graphical Models 2006

Density Estimation

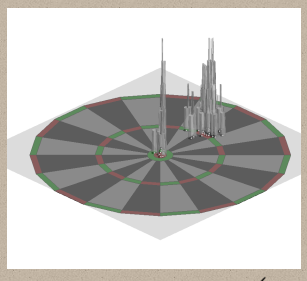
How many data points at (close to) x? y = p(x)





$p(x) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(x|\mu_i, C_i)$

parametric



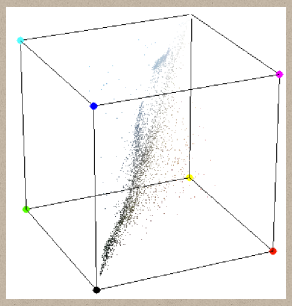
$f(x) = \frac{1}{N\rho^d} \sum_{i=1}^{N} h\left(\frac{x-x_i}{\rho}\right)$

non-parametric

Use of Density Estimation

Mean Shift Algorithm:

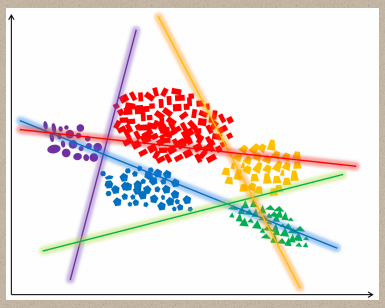
pick a point
follow local density uphill;
same summit; same label



Classification

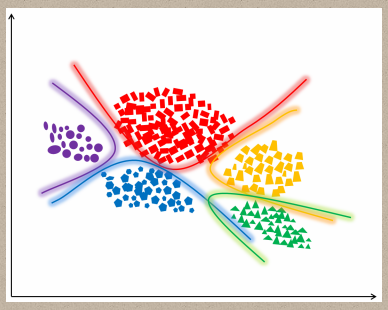
What thing is x? y = class(x)

Classification

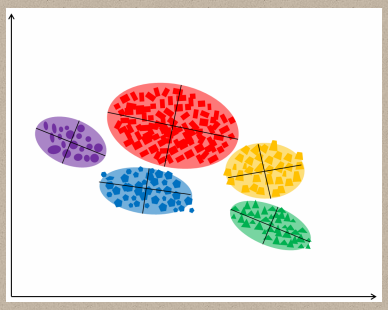


Support Vector Machine

$$\mathbf{w}^T\mathbf{x} - b = \begin{cases} +1 \\ -1 \end{cases}$$



Non-Linear SVM



Gaussian Mixture Model

$$p(x) = \sum_{i=1}^{K} \alpha_i \mathcal{N}(x|\mu_i, C_i)$$

Wu and Hall, Prime Shapes in Natural Images, BMVC 2012

Use of Classification

Shape Classification used in the service of NPR

Song *et al*. Abstract Art by Shape Classification, TVCG 2013

Underpinning Mathematics

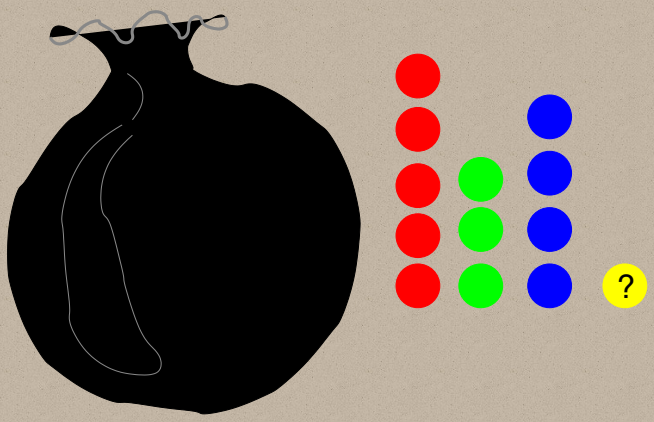
- *Random Variate*
  - set of possible values that can be assigned at random

$$\mathbb{X} = \{X_1, X_2, ..., X_M\}$$

- *Random Variable*
  - the variable assigned a value at random

$$x = X_i$$

Variates and Variables



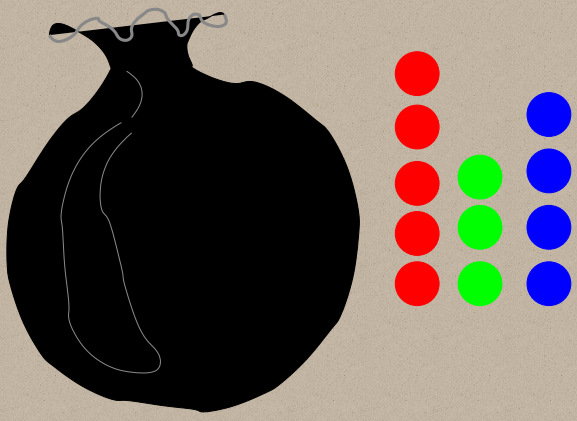
Variates: red, green, blue.

Variables: x_1, x_2, x_3, …

$p(x_3 = \text{red})$
$p(x_1 = \text{blue and } x_4 = \text{green})$

Statistics, Probability, Expectations

- Probability, p(x)



$$p(red) = \frac{5}{12}$$

$$p(green) = \frac{3}{12}$$

$$p(blue) = \frac{4}{12}$$

Statistics, Probability, Expectations

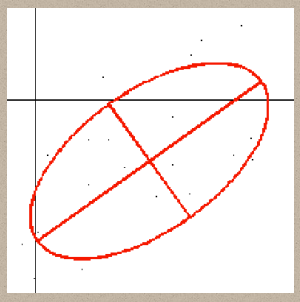
- Statistics

mean

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

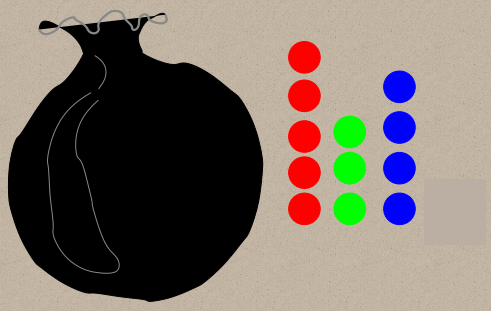
covariance

$$C = \frac{1}{N} \sum_{i=1}^{N} (x - \bar{x})(x - \bar{x})^T$$



Statistics, Probability, Expectations

- Expectation $E[f] = \int_{-\infty}^{\infty} f(x)p(x)dx$



$$E[colour] = \frac{5}{12}\begin{bmatrix}1\\0\\0\end{bmatrix} + \frac{4}{12}\begin{bmatrix}0\\1\\0\end{bmatrix} + \frac{3}{12}\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$= \frac{1}{12}\begin{bmatrix}5\\3\\4\end{bmatrix}$$

Axioms of Probability

1: all probabilities lie between 0 and 1

$$\forall x : p(x) \in [0, 1]$$

2: the probabilities of mutually exclusive events sum

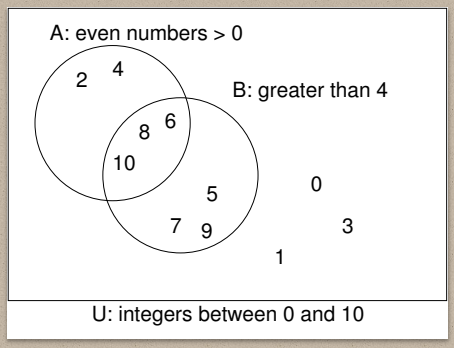
$$p(X_1 \cup X_2 ...) = \sum_{i=1}^{M} p(X_i)$$

3: the sum of all mutually exclusive events is unity

$$\cup_i X_i = \mathbb{X} : p(\cup_i X_i) = 1$$

Bayes' Law

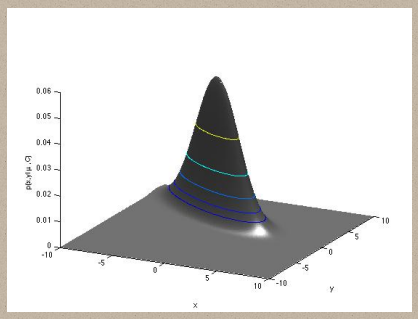
- Connects a ***posterior*** with a ***prior***,
  via an observed ***likelihood*** and the chance of ***evidence.***



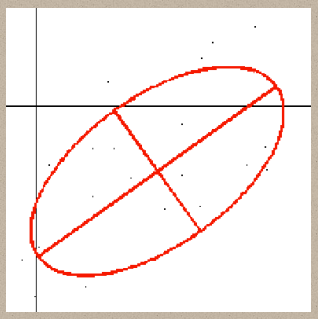
$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

The Gaussian (Normal) Distribution

$$p(x|\mu, C) = \frac{1}{(2\pi)^{d/2}|C|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T C^{-1}(x-\mu)\right)$$



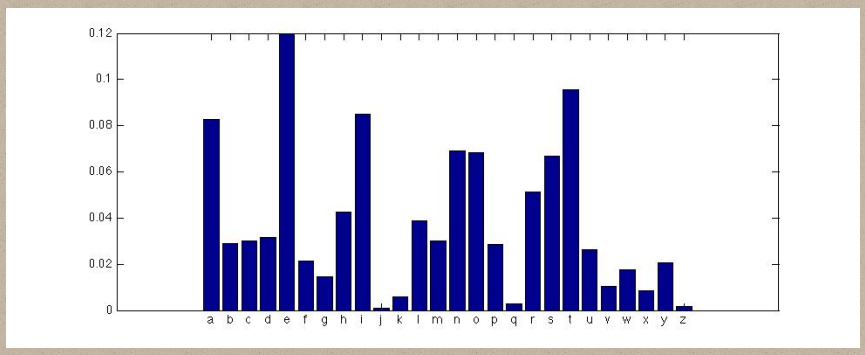
$$p(x|\mu, C) = \mathcal{N}(x|\mu, C)$$



$$C = ULU^T$$

The Multinomial Distribution

$$p(m_1, m_2, ..|\mu_1, \mu_2, ...) = \frac{(\sum_{k=1}^{K} m_k)!}{\prod_{k=1}^{K}(m_k!)} \prod_{k=1}^{K} \mu_k^{m_k}$$



ML and MAP

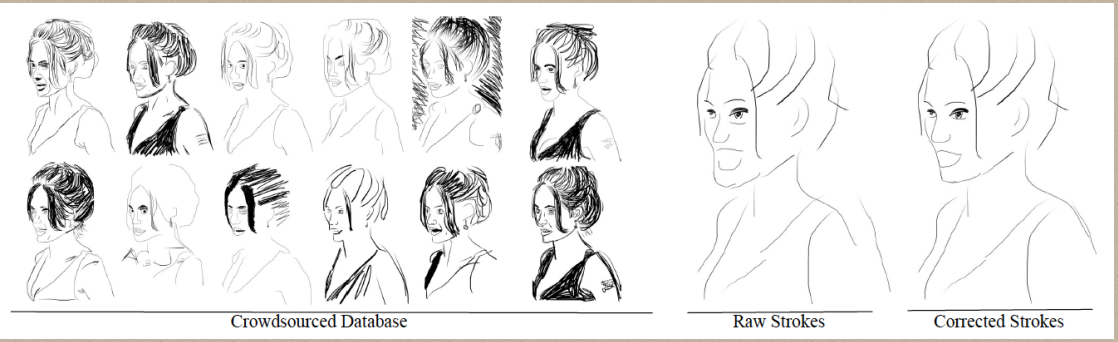
- Maximum Likelihood

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(x \mid \theta)$$

- Maximum A-Posterior

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(\theta \mid x)$$

Helping Non-Artists to Draw



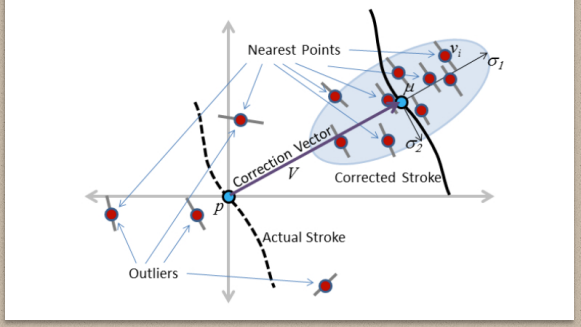
Limpaecher et al. Real-time drawing Assistance Through Crowd Sourcing, SIGGRAPH 2013. Used by Permission

correct a user's stroke using crowd sourced data

1. collect many tracings over a picture
2. New user makes a tracing, for each new stroke:
   i) use crow-sourced data to *suggest* new points for the input stroke,
   ii) correct the stroke *as a whole*.

suggest new points using crowd sourced points, x_i

Limpaecher et al. Real-time drawing Assistance Through Crowd Sourcing, SIGGRAPH 2013. Used by Permission



at *each point* on the stroke, suggest a new point:

$$\mu = E[x_i] = \sum_{i=1}^{N} x_i p(x_i)$$

use a Gaussian to fix p(x)

$$w_i = \mathcal{N}(x|0, C_i)$$

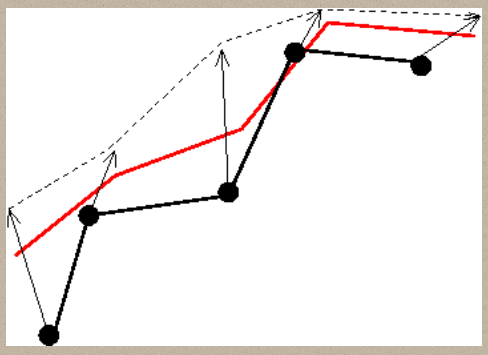
$$p(x) = \frac{w_i}{\sum_{j=1}^{N} w_j}$$

$$\sigma_1 = (\sum_{i=1}^{N} p(x) u_1^T (x-\mu))^{1/2}$$
$$\sigma_2 = (\sum_{i=1}^{N} p(x) u_2^T x)^{1/2}$$

$$C = [u_1 u_2] \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} [u_1 u_2]^T$$

correct stoke *as a whole* using regression

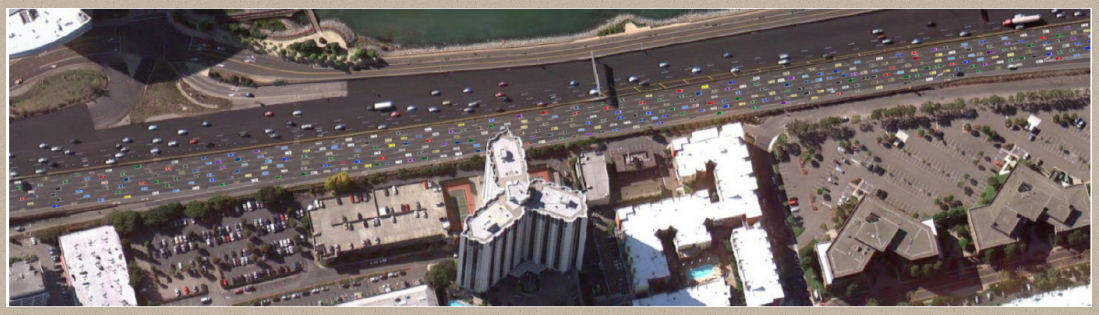
$$E(q_1, q_2, ...) = \sum_{j=1}^{N} ||q_j - \mu_j||^2 + \sum_{j=2}^{N} ||(q_j - q_{j-1}) - (x_j - x_{j-1})||^2$$



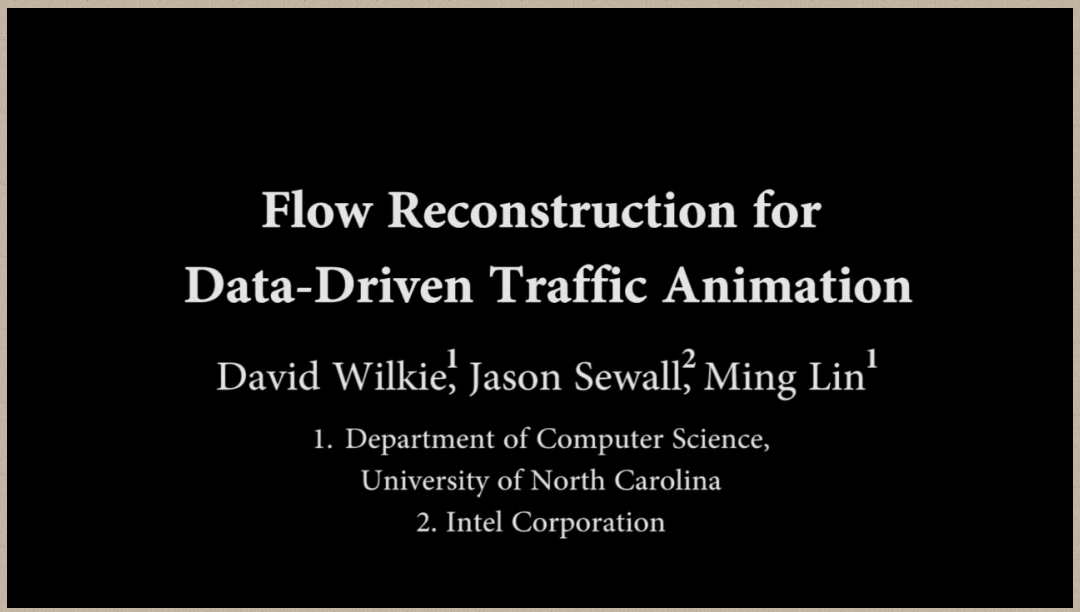
Regression determines a new set of points.

It take into account:
a) The distance from each regressed point to the corresponding suggested point,
b) the change in length of line segments

Data-Driven Animation of Traffic Flow

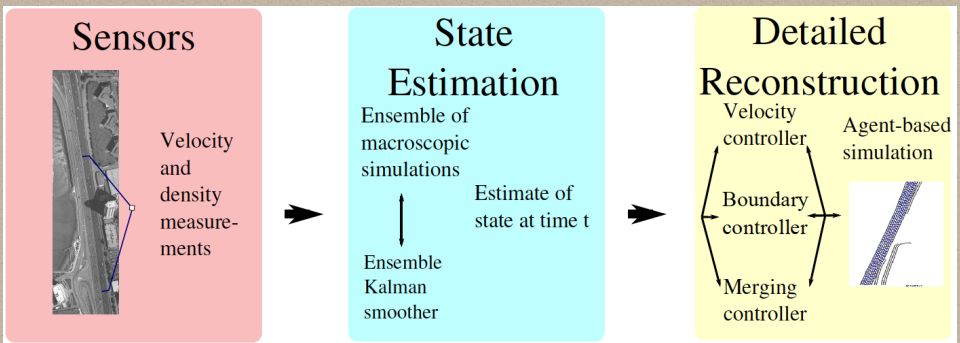


Wilke et al. Flow Reconstruction for Data-Driven Traffic Animation. SIGGRAPH 2013
Used by Permission



Used by Permission

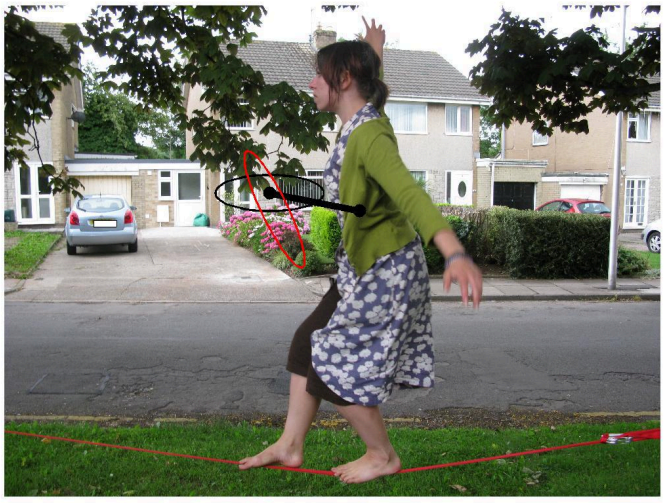
Three step algorithm



Wilke et al. Flow Reconstruction for Data-Driven Traffic Animation. SIGGRAPH 2013
Used by Permission

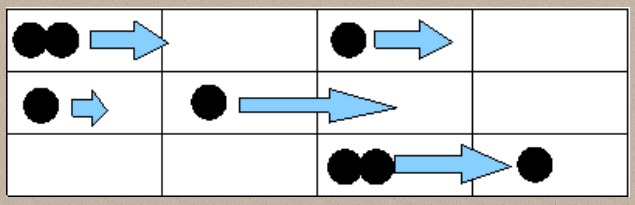
State Estimation: Kalman Filter

predict, correct

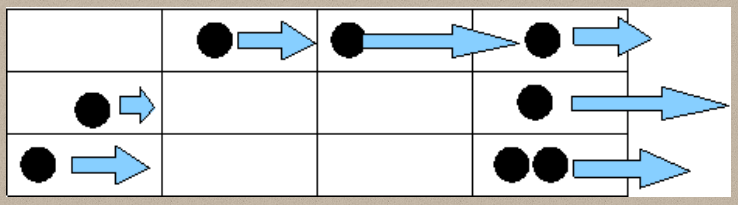


State Estimation: EnKS

predict, correct



Ensemble Kalman Smoothing



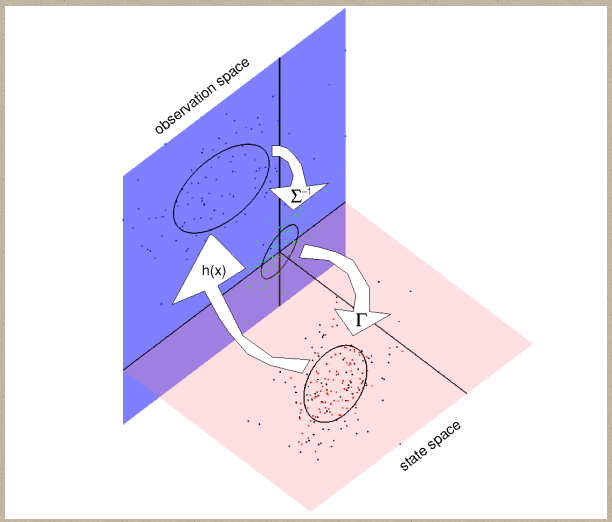
Ensemble Kalman Smoothing

Algorithm 1: EnKS for Traffic State Estimation

Input: Traffic sensor measurements $z_1 \ldots z_{t_n}$, ARZ simulator in f, observation model in h, initialized ensemble $\mathbf{x}_0^0 \ldots \mathbf{x}_0^{m-1}$, noise vectors $\mathbf{m}_t^0 \ldots \mathbf{m}_t^{m-1}$ and $\mathbf{n}_t^0 \ldots \mathbf{n}_t^{m-1}, \forall t \in 1 \ldots t_n$

Output: Traffic state estimates $\hat{\mathbf{x}}_t, \forall t \in 1 \ldots t_n$

for $t \in 1 \ldots t_n$ **do**
  for $i \in 0 \ldots M-1$ **do**
    // Motion model
    $\mathbf{x}_t^i \leftarrow f(\mathbf{x}_{t-1}^i, \mathbf{m}_t^i)$;
    // Observation model
    $\mathbf{z}_t^i \leftarrow h(\mathbf{x}_t^i, \mathbf{n}_t^i)$;
  // Analysis
  $\hat{\mathbf{z}}_t \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{z}_t^i$;
  $\Sigma_t \leftarrow \frac{1}{M-1} \sum_{i=0}^{M-1} (\mathbf{z}_t^i - \hat{\mathbf{z}}_t)(\mathbf{z}_t^i - \hat{\mathbf{z}}_t)^T$;
  for $j \in 1 \ldots t$ **do**
    $\hat{\mathbf{x}}_j \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{x}_j^i$;
    $\Gamma_j \leftarrow \frac{1}{M-1} \sum_{i=0}^{M-1} (\mathbf{x}_j^i - \hat{\mathbf{x}}_j)(\mathbf{z}_t^i - \hat{\mathbf{z}}_t)^T$;
    $K_j \leftarrow \Gamma_j \Sigma_t^{-1}$;
    for $i \in 0 \ldots M-1$ **do**
      $\mathbf{x}_j^i \leftarrow \mathbf{x}_j^i + K_j(\mathbf{z}_t - \mathbf{z}_t^i)$,;
// Final output
for $t \in 1 \ldots t_n$ **do**
  $\hat{\mathbf{x}}_t \leftarrow \frac{1}{M} \sum_{i=0}^{M-1} \mathbf{x}_t^i$;

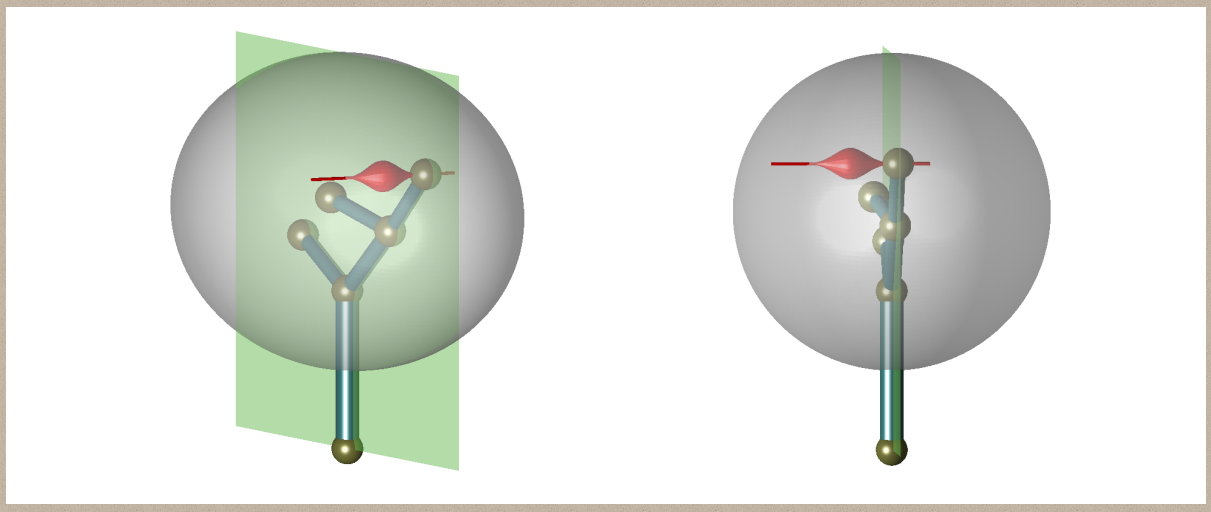


Wilke et al. Flow Reconstruction for Data-Driven Traffic Animation. SIGGRAPH 2013
Used by Permission

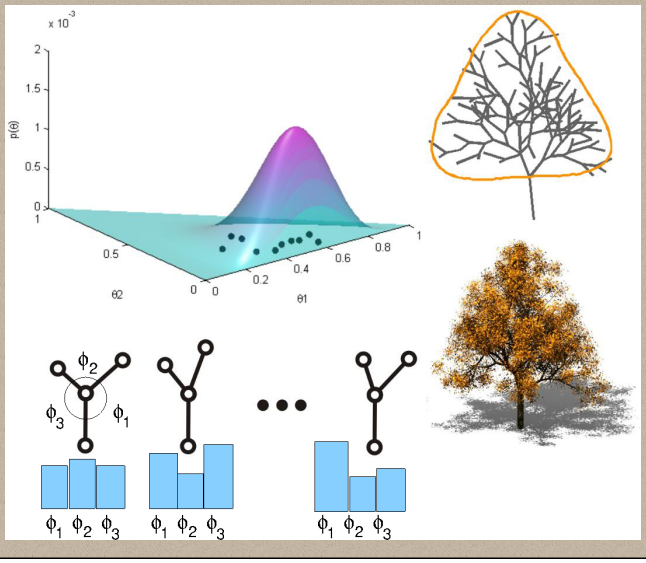
Modelling Moving Trees from Video

Li *et al*. Modeling and Generating Moving Trees from Video. SIGGRAPH Asia 2011

Push 2D into 3D



Generate New Skeletons



Learn More

- Books
  - Bishop. Pattern Recognition and Machine Learning
  - Duda, Hart, Stroke.
- Conferences
  - CVPR, ICCV, ECCV
  - NIPS, ICML