Signals, Patterns and Symbols

Dedicated to Charles Babbage

November 18, 2020

1 Data, Data Modeling and Estimation

1.1 Data - Types, Notation, Acquisition and Properties

Types of data: numeric - sequences, lists, symbolic - sequences, lists

- Numeric sequences - Digital Speech 1-D, Digital Images 2-D
- Symbolic Symbols - DNA sequences, words in text
- Numeric and Symbolic Lists - e.g tables with numeric or symbolic data

Patterns - structured collection of elements.
Sampling and Quantisation $\Rightarrow$ Digital representation. Sampling rate $f_s = 1/\tau_s$, where τ_s is a sample size
Nyquist-Shannon Sampling Theorem - "A signal can be reconstructed from samples if the sampling rate is greater than twice the highest frequency component in the signal" $f_s \geq 2f_m \Rightarrow$ No corruption of signal.
Aliasing - Error in reconstructed signal due to sampling rate being too low. Higher frequencies 'foldover' into lower frequency bands.
Quantisation - Analogue signals need to be quantised to digital representation. Quality depends on bits. K bits per sample means 2^K quantisation levels. Examples: speech: 8 bits/sample 8 KHz sampling – audio: 16 bits/sample 44 KHz sampling – colour images: 24 bits/pixel.
(**Histogram plots** - give probabilistic information too)

1.2 Distance

- A measure of 'separation' between data. It is often difficult to give the measure of 'distance' a meaning, e.g. what is the 'distance' between "aim" and "sight"?
- Properties: non-negativity $D(a,b) \geq 0$, reflexivity $D(a,b) = 0$ iff $a = b$, symmetry $D(a,b) = D(b,a)$, triangle inequality $D(a,b) + D(b,c) \geq D(a,c)$
- **Numeric Distance - distance between numeric data**
  - Euclidean distance - $||\mathbf{u} - \mathbf{v}|| = \sqrt{\sum_i (u_i - v_i)^2} \equiv \sqrt{(\mathbf{u} - \mathbf{u})^T \mathbf{u} - \mathbf{v})}$
  - ($\mathbf{u}^T\mathbf{v} = u_1v_1 + u_2v_2 = \mathbf{u} \cdot \mathbf{v}$ - inner or dot product, $||\mathbf{u}|| = \sqrt{\mathbf{u}^T\mathbf{u}} = \sqrt{\mathbf{u} \cdot \mathbf{u}}$) - some useful rules
  - P-norm distances (Minkowski distance, L_knorm) - for d-dimensional vectors. Note for 1-D (i.e. $d = 1$) the Euclidean distance and Manhattan distance are equal (the 2's cancel).
    - P-norm - $L_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{\frac{1}{p}}$
      - $\Rightarrow$ Manhattan Distance (think taxi in a city)- $L_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|$
      - $\Rightarrow$ Euclidean Distance (think Pythagoras)- $L_2(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^2 \right)^{\frac{1}{2}}$

· $\Rightarrow$ Maximum norm (also known as Chebyshev distance) (think King on a chest board) - $L_\infty(\mathbf{x}, \mathbf{y}) = max_i|x_i - y_i|$

- **Weighted (Numeric) Distance** - gives components 'weights'
  - weighting matrix $\mathbf{W} = diag[w_1, w_2]$
  - Distance $D_w(\mathbf{u}, \mathbf{v}) = (\mathbf{u} - \mathbf{v})^T\mathbf{W}(\mathbf{u} - \mathbf{v}) = \sum_i w_i(u_i - v_i)^2$(Use matrix vector product to solve this)
  - (Euclidean weighted distance = $\sqrt{D_w(\mathbf{u}, \mathbf{v})}$)
- **Symbolic distance**
  - **Syntactic** - relating to the symbols used
    - Hamming distance - Minimum number of changes needed to get from one word to another if they are both the **same length**.(think how many differences are there between the words/string).
    - Edit distance - Number of 'operations' needed to get from one sequence to another using **substitution, insertion** and **deletion** (think about the minimum amount of changes needed to get from one word to another). Works for any difference in length of the words.
  - **Semantic** - relating to meaning
    - WordNet (basically a lexicon)- uses synsets - inter-changeable synonyms. Hyponymy (hypo means 'under' nymy means 'name') (is-a (comscience terminology) relationship) between a sub-class hyPOnymy (e.g crow,eagle,..) and an upper-class hyPERnymy (birds). Meronymy (mero basically means 'part of') So A (e.g. finger) is (is-part (comscience term) part of B (e.g) means A is a meronymy of B. relationship).
      · WUP distance - Uses the WordNet and gives a numeric "distance" between 0 and 1 as to how connected two words are: WUP(cat,dog) $= \frac{2\times numberofnodesfrommain/largesthypERnymtosharednode}{numberofnodestosharednodefromcat+numberofnodesfromlargesthypernym+2\times numberofnodesfromdogtosharednode}$
- **Similar Sequences** - measuring similarity using correlation, independent of absolute value
  - Correlation coefficient for sequences $u(n)$ and $v(n)$: $C = \frac{1}{N\sigma_u\sigma_v}\sum_n(u(n) - \mu_n)(v(n) - \mu_v)$, where $\mu_{u/v} = \frac{1}{N}\sum_n u/v(n)$ - average, $\sigma^2_{u/v} = \frac{1}{N}\sum_n(u/v(n) - \mu_{u/v})$
  - Or just use Eng maths correlation coefficient formula - or calculator ;) They divide by N here hmmm, so be careful...
  - Cross correlation measure - $r_{uv}(m) = \frac{1}{N\sigma_u\sigma_v}\sum_n(u(n) - \mu_n)(v(n - m) - \mu_v)$, i.e. shift one function and then find correlation.
- **Dynamic Time Warping** - For 'squashed' and 'squeezed' sequences.
  - A distance measure between to sequences that are non-linearly synced.
  - DTW distance - $C_{p*}(x, y)$ where $p* = argmin_pC_p(x, y)$, where $C_p(x, y)$ is the sum of the distances between mapped elements according to warped path p.
  - Triangle inequality doesn't necessarily hold...

1.3 Characterising data

- Basic:
  - Mean 'average': $\mu = \frac{1}{N}\sum_i v_i$
  - Variance 'spread': $\sigma^2 = \frac{1}{N}\sum_i(v_i - \mu)^2$
  - Using histogram (like frequency) then mean is $\mu = \frac{1}{N}\sum_j a_jh(a_j)$ and variance is $\sigma^2 = \frac{1}{N}\sum_j(a_j - \mu)^2h(a_j)$
  - SAMPLE mean is $\bar{x} = \frac{1}{N_s}\sum_i^{N_s} x_i$ (unbiased $\bar{x} \to \mu$ as $N_s \to \infty$) and sample variance is $s^2 = \frac{1}{N_s-1}\sum_{i=1}^{N_s}(x_i - \bar{x})^2$ (unbiased $s^2 \to \sigma^2$ as $N_s \to \infty$)

- Vector form:
  - Mean Vector $\mu = \frac{1}{N}\sum_i \mathbf{v}_i$
  - Covariance Matrix $\mathbf{C} = \frac{1}{N}\sum_i(\mathbf{v}_i - \underline{\mu})(\mathbf{v}_i - \underline{\mu})^T$ (To calculate this in matrix format see lecture 6 and to see patterns see lecture 4 notes). Use this if you are working with 'populations'. Use $\mathbf{C} = \frac{1}{N-1}\sum ...$ if you are working with samples.
- Principle axis and vector form:
  - Solve eigenvectors and eigenvalues from covariance matrix, since $\mathbf{C}\mathbf{u}_i = \lambda \mathbf{u}_i$.
  - Eigenvector tells us about orientation of the data (principle axes)
  - Eigenvalue gives us information about the spread along principle axis. $\sqrt{\lambda}$ is the spread.
  - Eccentricity [0,1] - ratio of the distance of one of the focus points to the centre (c) to to the distance of the end of the major axis to the centre (a) [$e = \frac{c}{a}$, $c < a$] - this shows the degree of correlation between the data. The more eccentric, the more correlated.
- Data outliers
  - Mean and Covariance break down...
  - Could use Median ('middle value') instead, but annoying to work with
  - RANSAC method (Random Sample Consensus)
  - 'kernels' which place a boundary on which data is taken into consideration. Usually variance would be defined as squaring what inside the summation $(u_i - \bar{u})^2$ a kernel can be applied that does something different to it, like nullify outliers $\rho(u_i - \bar{u})$
- Location and Spread
  - Centroid 'Centre of mass', 'Weighted mean' $\bar{n} = \frac{1}{M}\sum_n u(n)n$, where $M = \sum_n u(n)$
  - Moment of Inertia 'Spread about centre of mass' $I = \frac{1}{M}\sum_n u(n)(n - \bar{n})^2$. $\sqrt{I}$ gives spread (I think).
  - For multiple dimensions: centroid - $\bar{\mathbf{x}} = \frac{1}{M}\sum_i u(\mathbf{x_i})\mathbf{x_i}$ and moment of inertia $\mathbf{I} = \frac{1}{M}\sum_i u(\mathbf{x_i})(\mathbf{x_i} - \bar{\mathbf{x}})(\mathbf{x_i} - \bar{\mathbf{x}})^T$
  - Note! Mean is the centroid and variance is the moment of inertia of a histogram.

1.4 Data Modelling - Deterministic

- Models allow us to design 'optimal' algorithms, quantify performance and compare/contrast methods. All models have an 'underlying' model.
- Training data - data collected to 'train'/'tune' the model
- Beware of over-fitting - simple models are often more useful.
- Deterministic models - assumes that relationships within the data are fixed and can be predicted over time (e.g. differential equations). Does not account for randomness or uncertainty in data (e.g Markov chains).
- **Least Squares Method** - minimised squares of vertical offset between best fit and data. For a straight line $R(a,b) = \sum_i (y_i - (a + bx_i))^2$. $\boxed{\mathbf{y} = \mathbf{X}\mathbf{a}}$. In Matrix form the 'residual' $R(a,b) = ||\mathbf{y} - \mathbf{X}\mathbf{a}||^2$, where $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$, $\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}$, and $\mathbf{a} = \begin{bmatrix} a \\ b \end{bmatrix}$
  - Non-matrix solution to this minimisation are $a_{LS} = \bar{y} - b\bar{x}$ and $b_{LS} = \frac{\sum_i x_i y_i - N\bar{x}\bar{y}}{\sum_i x_i^2 - N\bar{x}^2}$, where $\bar{x}, \bar{y}$ are the mean of the data points and N is the number of data points (I think).
  - **Matrix solution** is $\mathbf{a}_{LS} = [c, m]^T = \boxed{(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}}$
  - Residual = $\sum(y_i - (mx_i + c))^2$
  - ERROR FOLLOWS NORMAL DISTRIBUTION - hence why we can use probabilistic modelling too

1.5 Probabilistic Modelling

- Marginal distributions - $P_1(x_1) = \int_{x_1} P(\mathbf{x})dx_2$ and vice versa for $P_2(x_2)$. If the elements/observations are independent then $P(\mathbf{x}) = P_1(x_1)P_2(x_2)$
- Conditional Probability: $P(A|B) = P(A,B)/P(B)$ the $, = \cap$
- Maximum Likelihood estimation (**ML**) recipe (assume $y = \theta x + \epsilon$ and data D.) [aim: $argmax_\theta p(D|\theta)$]
  - Determine θ, D and likelihood expression $p(D|\theta)$, which might be something like $\prod N(\mu, \sigma^2)$
    - The key thing here is that the error (residuals) follows a normal distribution of e.g. $N(\mu, \sigma^2), sop(\epsilon) \sim e^{(\epsilon-\mu)^2/2\sigma^2}$, but that does not mean the pdf $p(D|\theta)$ follows that distribution. Instead substitute $\epsilon = z_i - ax_i$ to find its distribution.
  - Take log of likelihood (since argmax $a_{ML} = p(D|\theta) = argmax lnp(D|\theta) = argmin - lnp(D|\theta)$
  - Take derivative or partial derivatives and set equal to zero and solve for θ (for normal distribution random error ϵ, θ is going to be something like $\theta = \frac{\sum_i y_i x_i}{\sum_i x_i}$ (identical to LSQ method)
- Variability ==> $V(a_{ML}) = E[(a_{ML} - \bar{a_{ML}}] = \frac{\sigma^2}{\sum_i x_i}$ (shows variance decreases if data is further from the origin)
- **MAP** estimation [aim: $argmax_\theta p(D|\theta)p(\theta)$]
  - Determine $p(D|\theta)p(\theta)$
  - Take logarithm
  - Take derivative and set to zero and solve for θ. Note, MAP is equivalent to adding a data point to the set of data.

2 Classification and Clustering

2.1 Classification

- Decision Rules and how to classify
  - Decision boundary = points of class change = points of class uncertainty
  - Coming up with a decision rule: 1) Describe data (e.g. feature: lightness, classes: salmon/sea-bass) 2) Select classification model (e.g feature normally distributed, classes equally likely) 3) Estimate parameters (e.g mean, variance) - might use likelihood estimation 4) Form decision rule (e.g. if $p(lightness|seabass) \geq p(lightness|salmon)$ then sea bass else salmon [Max Likelihood Classifier] or e.g if $p(lightness|seabass)P(seabass) \geq p(lightness|salmon)P(salmon)$then sea bass else salmon [Maximum a Posteriori classifier] 5) Classify
- Independence
  - If data are independent, we can assume covariance matrix to be 1 and decision boundaries straight lines. If data is dependent then there will be curved (hyperbolic) decision boundaries.
- BAYESIAN CLASSIFICATION: Naive-Bayes classifiers
  - There are two types of classifier - the standard one, which assumes no prior knowledge (ML classifier) and the other is the MAP classifier which includes prior probabilities.
  - Both classifiers are naive because they **assume** features are independent (even within the classes - class-conditional independence)
    - let $C_1, C_2, .., C_k$ be the classes and $\mathbf{x} = x_1, x_2, ..., x_n$
    - ML classifier: $P(C_k|\mathbf{x}) = \prod_i^n P(x_i|C_k) \rightarrow$ Decision boundary could be that stated above for ML, or simply $\hat{y} = \underset{k \in 1...k}{argmax} \prod_i^n P(x_i|C_k)$, where $\hat{y} = C_k$ is the class label.

    * MAP classifier: $P(C_k|\mathbf{x}) = P(C_k)\left(\prod_i^n P(x_i|C_k)\right) \rightarrow$ Decision boundary could be that stated above for MAP, or simply $\hat{y} = \underset{k\in 1\ldots k}{argmax} P(C_k)\left(\prod_i^n P(x_i|C_k)\right)$, where $\hat{y} = C_k$ is the class label.
  - Laplace correction

- DECISION TREE CLASSIFICATION: GrowTree algorithm
  - Decision trees are hard to describe but simple to use. The basic idea is to create a decision tree based on paths with the most information gain. Start by calculating the entropy/impurity of the classes ($E(class) = \sum_i^n -p_i log_2(p_i)$) - this corresponds to your Imp(parent) (entropy of parent node). Next you have to calculate the entropies of the classes given a certain feature ($E(class, feature) = \sum_i^n -\frac{n_i}{N} p_i log_2(p_i) = \sum_i^n -\frac{n_i}{N} Imp(child)$ where n_i is the number of times an attribute of a feature (e.g feature = shape; attributes = triangle, square, circle) occurs in that feature. Then finally you calculate the information gain of using a particular feature $I.G. = E(class) - E(class, feature)$ for all features and chose the one with the highest information gain. This then becomes the first node of the 'tree' and the 'parent', and the other features become 'children'. You repeat this process until you only have 'pure' children. It makes sense when you do it...
  - Pruning - decomplexifies the trees and makes classifying better and less overfitted. For algorithm see lecture notes.
  - There is a link to MAP and Naives-Bayes - when you find it, add it here...
  - Applications: Direct Marketing, Fraud detection, Sky survey Cataloguing

- (K)-Nearest Neighbour classification
  - Assign a test instance to the majority class among its k nearest neighbours. Basically take your test data point and classify it to the class that has the majority from the e.g. k = 5 nearest neighbours.
  - Weighted distances can also be considered using a 'Gaussian kernel' $K(d) = e^{||d||^2}$

2.2 Clustering

- K-means Clustering
  - This sums it up pretty nicely: **function** KMeans(Instances,K) randomly initialise K vectors 1 ... K; **repeat** assign each xInstances to the nearest j; recompute each j as the mean of the instances assigned to it; **until** no change in 1 ... K; **return** 1 ... K;
    * Does not necessarily find global optimum $\Rightarrow$ Use domain-dependent initialization schemes to be more certain that one is closer to global optimum.

- Agglomerative hierarchical clustering
  - It iteratively clusters the data by merging the nearest data/clusters (once two points or a cluster and point merge they become a cluster)
  - Advantage - you can chose the number of clusters you like (from Dendrogram) Disadvantage - its slow to compute $O(n^3)$
  - **Linkage** (how the distance is determined between the clusters)
    * Single linkage - distance of nearest two points of the two clusters
    * Complete linkage - distance of furthest two points of the two clusters
    * Average linkage - average distance between the clusters
    * Centroid linkage - distance between the centroids of the clusters
  - Dendrogram - visual representation of the algorithm. Go from down up, i.e. start with all the data points and then merge according to linkage to form an upside down try of a sort. Note complete linkage does not mean take the furthest distance - it means look at the furthest distances but still chose the smallest one when clustering.

  - Silhouettes - a plot of $s(x) = (b(x) - a(x))/(max(a(x), b(x)))$, where a(x) is the average distance of point to all the other points in its cluster and b(x) is the average distance of that same point to all other points in the nearest lying cluster.

- Expectation and Maximisation (EM algorithm) - iterative method for solving for parameters given some missing information. It has two steps: don't fully understand - edit after doing a few questions
  - Gaussian Mixture Methods - example of EM algorithm
  - E-step: 'assign expected value to hidden variables' - basically you chose random means and variances and work out the likelihood probability that a data point fits the Gaussian distributions resulting from the means and variances you have chosen $P(\bar{x}|parameters)$
  - M-step: 're-estimate parameters' - now kind of do the opposite to above, namely shift the initial means and variances to new ones that fit the data better by working out the posterior probability $P(parameters|\bar{x})$. It's just like k-means, except it using probabilities instead of centroids.
  - Applications: Market segmentation, Document clustering

2.3 Evaluation

[DSP = Digital Signal Processing = manipulation of signals using digital techniques]

- ROC

3 Representation, Transformation and Feature Extraction

3.1 Signals and Fourier Analysis

- Test if function is odd by substituting $(-x)$ into x.
- Any function $f(x) \neq 0$ can be expressed as a sum of even and off functions $f(x) = f_e(x) + f_o(x)$

3.2 Fourier Series

- Fourier Series = $f(t) \approx \frac{a_0}{2} + \sum_{i}^{\infty}(a_n cos(n\omega t) + b_n sin(n\omega t))$ where $\omega = \frac{2\pi}{T}$
- a_n, b_n are Fourier coefficients and cos, sin are the basis functions

3.3 1D Fourier Transform

- Continuous Fourier transform pair: F.T: $F[f(t)] = F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt$, I.F.T $F^{-1}[f(t)] = f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}dt$ Note, He uses a slightly different formula - without 1/2pi and w
- Discrete Fourier transform pair: F.T: $F(\omega) = \frac{1}{N}\sum_{i=0}^{N-1} f(t)e^{\frac{-j2\pi\omega t}{N}}$, I.F.T: $f(t) = \frac{1}{N}\sum_{i=0}^{N-1} F(\omega)e^{\frac{j2\pi\omega t}{N}}$, for $x = u = 0, 1, ..., N-1$
- Facts and properties
  - $F(\omega) = R(\omega) + jI(\omega)$ i.e. $F(\omega)$ is complex.
  - Magnitude or spectrum: $|F(\omega)| = \sqrt{R(\omega)^2 + I(\omega)^2}$ Distribution of this gives frequency spectrum.
  - Phase spectrum: $\phi(\omega) = tan^{-1}\frac{I(\omega)}{R(\omega)} = \frac{complexpart}{realpart}$
  - $F(\omega)$ in polars is $F(\omega) = |F(\omega)|e^{j\phi(\omega)}$
- Observations:

- Slow changing signals are dominated by low frequencies so frequency spectrum has mainly low frequencies (vice versa for fast changing signals)
- Simple application of speech recognition - transform speech into frequency domain then calculate euclidean distance $d_E = (\sum_i |X(\omega) - Y(\omega)|)^{frac12}$ instead of calculating a distance in time domain.

3.4 2D Fourier Transform

- Defined in terms of **spacial** frequency (because 2D I presume)...
- Particularly useful in image processing because it characterises rate of change of **intensity** along each dimension.
- Observations
  - Recall that for 1D a slow changing signal produces a low frequency spectrum, likewise in 2D if the colour (rate of intensity) doesn't change abruptly in the x or y direction then this corresponds to low frequencies.
  - For $u, v = 0$ i.e. the frequency in x and y are zero is the slowest possible varying frequency level - this corresponds to averyage gray level in an image.
- Continuous Fourier transform pair - F.T.: $F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-2\pi j(ux+vy)}dxdy$, I.T.F: $f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{2\pi j(ux+vy)}dudv$
- Discrete Fourier Transform pair - F.T $F(u,v) = \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{\frac{-2\pi j(ux+vy)}{N}}$, I.F.T $f(x,y) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} F(u,v)e^{\frac{2\pi j(ux+vy)}{N}}$ for $x, y, u, v = 0, 1, ..., N-1$
- Properties are identical to 1D properties.

3.5 Frequency Spectrum

- Often plotted using $log(abs(F(u,v)) + 1)$ - spreads the spectrum out and makes it clearer to see
- Think of ripples in a pond where the frequency increases radially - each pixel of the 2D frequency plot a 2D image of sign waves at a certain angle. Thus the brightness of that pixel shows 'how much' of the 2D frequency is present in the image.
- Properties:
  - Symmetric - You only need half of the Fourier spectrum to gain useful information - $F(u,v) = F^*(-u,-v)$ and $|F(u,v)| = |F^*(-u,-v)|$
  - Separable - principle behind the fast Fourier transform (fft2) - you solve transform in the x direction first keeping y constant and then solve for y keeping x constant $f(x,y) \rightarrow F(u,y) \rightarrow F(u,v)$ (or vice versa)
  - Rotation - rotating the image rotates the fourier space image. $(x = rcos\theta, y = rsin\theta, u = \omega cos\phi, v = \omega sin\phi \rightarrow f(r, \theta + \theta_0) = F(\omega, \phi + \theta_0)$
- Observations:
  - Low frequencies $\rightarrow$ contrast
  - High frequencies $\rightarrow$ definition
- Filtering:
  - $f(x,y) \underset{F.T.}{\rightarrow} F(u,v) \underset{Applyfilter H(u,v)}{\rightarrow} G(u,v) = F(u,v)H(u,v) \underset{I.F.T}{\rightarrow} h(x,y)$
  - Butterworth's low-pass filtering:

    - $H(u,v) = \frac{1}{1+[r(u,v)/r_0]^{2n}}$, where $r(u,v) = \sqrt{u^2+v^2}$, r_0 filter radius
  - Butterworth's high-pass filter
    - $H(u,v) = \frac{1}{1+[r_0/r(u,v)]^{2n}}$, where $r(u,v) = \sqrt{u^2+v^2}$, r_0 filter radius
  - Butterworth makes the filter smoother. A basic filter would be $H(u,v) = \begin{cases} 1 & r(u,v) \leq r_0 \\ 0 & r(u,v) > r_0 \end{cases}$

3.6 Features

- How to choose relevant features:
  - Feature Selection: Select a subset of features without a transformation.
    - E.g. given too many features which cause your computer to overload you need to reduce the number of features - say from 25 to 10 features. This means there are 25C10 = 3268760 possible combinations... Therefore we have to use a **cost/objective function** that maximises a certain condition, e.g P(correct classification)
    - Heuristic techniques:
      - Step-wise feature selection -start bottom-up selecting best feature and then next best feature given first feature and so on...
      - Step-wise elimination - Start with full set of features and then progressively eliminate the worst ones.
  - Feature Extraction: Transform features into smaller dimensional space (subset)
    - Aim: Given feature space R^N, with feature vectors $\mathbf{m}$ find transforming mapping $\mathbf{x} = \phi(\mathbf{m}) : R^N \to R^d$ such that the transformed feature vector $\mathbf{x} \in R^d$ conserves most of the information of R^N.
- In Fourier space power $(= \sum_u \sum_v |F(u,v)|^2$ is often used to extract features?...
- Spacial filtering
  - 1D - Convolution is defined as $(f*g)(t) = \int_0^t f(t-\tau)g(\tau)d\tau = \int_0^t f(\tau)g(t-\tau)d\tau = F(\omega)G(\omega)$
  - In practise you can do it without any integration. Put a pixel kernel (filter) over the true values and multiply each corresponding pixel value and then normalise by the total value of the pixel kernel.
  - 1-D Discrete Convolution $\sum_{-s}^{m=-s} f(x-m)h(m)$, where f(x) is the signal, h(m) is the filter/kernel and s is the number of neighbours. **Take kernel and flip it horizontally/vertically depending on orientation**
  - 2D- Discrete convolution $g(x,y) = f*g = \sum_{m=-s}^{s}\sum_{n=-s}^{s} f(x-m,y-n)h(m,n)$ **Flip Kernel horizontally, then vertically then mask it over the 2D input to get the output.**
  - 2D Correlation - $g(x,y) = f*g = \sum_{m=-s}^{s}\sum_{n=-s}^{s} f(x+m,y+n)h(m,n)$ give the masking - looks 2D correlation is what the youtube things are saying is convolution?
    - Correlation = Convolution if kernel is symmetric under 180 degrees rotation.
  - Convolution in the time domain $(f*g)(t)$ is equivalent to multiplication in the frequency domain $F(w)G(w)$ AND VISE VERSA $fg \leftrightarrow F(w)*G(w)$
- Edge Features:
  - Edges occur when there is a change in the intensity function $f(x,y)$ in an image i.e. pixels go from one colour to another.
  - For edge detection we are interested in the places where there is the greatest change so we need the gradient $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$
    - Gradient direction: $\theta = tan^{-1}\left(\frac{\frac{\partial f}{\partial x}}{\frac{\partial f}{\partial y}}\right)$
    - Edge direction (naturally a 90 degree rotation): $\phi = \theta - \frac{\pi}{2}$

* Magnitude of edge: $||\nabla f|| = \sqrt{\frac{\partial f}{\partial x}^2 + \frac{\partial f}{\partial y}^2}$
* Kernel with zeros in vertical middle corresponds $\frac{\partial f}{\partial y} = 0$ and kernel with zeros along horizontal correspond to $\frac{\partial f}{\partial x} = 0$
* Sobel edge detection uses this to form clear edged images: $\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \rightarrow \frac{\partial f}{\partial x} = 0$, $\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \rightarrow \frac{\partial f}{\partial y} = 0$

– You can then plot histograms of the gradient magnitudes relative to angle ⇒ if there are lots of buildings in the image then we expect high gradients at 90 degrees.

3.7 PCA

- Step 1: Subtract the mean of the n-dimensional data $\mathbf{m}$ from the n-dimensional data set. $\mathbf{v} - \mathbf{m}$.
- Step 2: Find the covariance matrix of $\mathbf{v} - \mathbf{m}$.
- Step 3: Find the eigenvalues and normalised eigenvectors.
- Step 4: Organise eigenvalues into a column vector from largest (top) to smallest (bottom) λ and organise eigenvalues from largest to smallest in a matrix, where each eigenvector is a column in the matrix $\mathbf{u}$.
- Step 5: Generate new representation of data $\mathbf{a}$ by multiplying $\mathbf{u}^T(\mathbf{v} - \mathbf{m}) = \mathbf{a}$
- Step 6: Finally get the"old" data back by using this equation $\mathbf{v} = \mathbf{m} + \sum_{i=1}^{d} \mathbf{a}_i \mathbf{u}_i$, where d is number of principle dimensions you want to consider. This reproduces the data but only on the principle axis you decide on.
- PCA's power lies in its ability to retain the variance of the data (and thereby one of its key bits of information, (mean is known too)). To test how much information (in variance) the data holds, use the fact that the sum of the variances = sum of the eigenvectors $\sum_{i=1}^{p} = 100\%$ of variance in original step (trace property I think) (p is the dimension of all the features of the given data). Thus, to check how much information your principle components entail, use the fraction $\frac{\sum_{i=1}^{d}}{\sum_{i=1}^{p}}$.