Applied: Exam Questions

Due on

Contents

2011 Q1

This question concerns fitting Gausian Mixture Models (GMs).

$$p(x) = \sum_{k=1}^{K} w_k N_x(m_k, \Sigma_k)$$

where $0 \leq w_k \leq 1$ and $\sum_{k=1}^{K} w_k = 1$ and

$$N_x(m_k, \Sigma_k) = \frac{1}{\sqrt{det(2\pi\Sigma)}} \exp\left(-\frac{1}{2}(x-m)^T\Sigma^{-1}(x-m)\right)$$

The GM will be trained to fit a set of N data points $X = x^1, \ldots, x^N$.

a. Explain why a GM can be used to cluster the data points.

We can use each element of the mixture to represent a cluster. We can represent the model as a marginalisation over a hidden variable h which can take the values of each of the classes and which has a prior distribution as a categorical variable with parameters equal to the weights. We define $pr[x|h]$ to be the relevant normal. Then

$$pr[x] = \sum_h pr[x,h] = \sum_h pr[x|h=k]pr[h=k] = \sum_h w_k N_x(m_k, \Sigma_k).$$

One the model is fitted, the posterior distribution for h given a data point will represent the probability that a point comes from the various clusters.

b. For independently and identically distributed data,show that the log likelihood is

$$\sum_{n=1}^{N} \log \sum_{k=1}^{K} w_k \mathcal{N}(x^n | m_k, \Sigma_k)$$

$$Likelihood = \prod_{n=1}^{N} \sum_{k=1}^{K} w_k N(x^n | m_k, \Sigma_k)$$

and

$$loglik = \sum_{n=1}^{N} \log \sum_{k=1}^{K} w_k N(x^n | m_k, \Sigma_k)$$

c. Show that by setting a GM mean m_k, to a data point x^n ,the log likelihood becomes infinite when the corresponding covariance matrix $\Sigma_i = 0$. Explain why for maximum likelihood training of the GM, one must place constraints on the covariance matrices in order to find sensible solutions.

We have:

$$loglik = \sum_{n=1}^{N} \log \sum_{k=1}^{K} w_k \frac{1}{\sqrt{det(2\pi\Sigma)}} \exp\left(-\frac{1}{2}(x-m)^T\Sigma^{-1}(x-m)\right)$$

When we set $m_k = x^n$ then for that datapoint and for that mixture, the second part of the loglik becomes 1. It cannot shrink regardless of how low σ^2 becomes. i.e. $\exp\left(-\frac{1}{2}(x-m)^T\Sigma^{-1}(x-m)\right) =$

$\exp\left(-\frac{1}{2}0\Sigma^{-1}0\right) = 1$. So we are just left with $\frac{1}{\sqrt{det(2\pi\Sigma)}}$ which we can make blow up by setting Σ to be very close to zero. Then by simply having a non-zero weight for this part of the mixture, and by letting Σ become very small, we can make the likelihood infinitely large.

This solution can be pretty meaningless - it just requires one point to be its own cluster and will not mean any other meaningful clusters have been found. Hence to stop this happening we should require a minimum size for the Σs.

d. Explain a reasonable initialisation procedure for training the GM on the datapoints X.

No idea. Possibly choose cluster centres at random, somewhere within the range of the data (along each dimension). Mainly though initialise with large variances (so that probable areas covered by the clusters, cover all the data for each cluster.

e. For the case of constrained variance matrices $\Sigma_k = \sigma_k I$ (that is, each covariance matrix is proportional to the identity matrix), show that the M step of the EM algorithm is given by

$$\sigma_i^2 = \frac{1}{D}\sum_{n=1}^{N} p^{old}(n|i)(x^n - m_k)^2$$

where the dimension of each datapoint and $p^{old}(n|i)$ is suitably well defined.

Lots of extra stuff here, just for background.

For EM there are always two terms in the lower bound, entropy and energy. We always have the variational distribution which is the probability of the hidden thing that we do not know (here this is the cluster membership), given the current parameter estimates and the data. The entropy is the expected value of the log of this variational distribution with respect to itself. The energy term is the expected value of the log likelihood over the visible data and the hidden variables, given the distribution parameters (here the means and covariances). In general, if h is the hidden variable with a variational distribution, the lower bound is

$$\sum_{n=1}^{N}\langle\log q(h^n|x^n)\rangle_{q(h^n|x^n)} + \sum_{n=1}^{N}\langle\log p(h^n, x^n|\theta)\rangle_{q(h^n|x^n)}$$

To be consistent with the slides, call the hidden class i. $pr[i]$ is then the probability that the class is class i for a given example and the lower bound is

$$\sum_{n=1}^{N}\langle\log q(i^n|x^n)\rangle_{q(i^n|x^n)} + \sum_{n=1}^{N}\langle\log p(i^n, x^n|\theta)\rangle_{q(i^n|x^n)}$$

The M step maximises the energy (second term) assuming the variational distribution is fixed. It will give us updates for the means and variances of the clusters. The $q(i^n|x^n)$ are an estimate of $pr[i^n|x^n]$ and get updated each iteration so to make this clear write $p^{old}[i^n|x^n]$ instead of $q(i^n|x^n)$. Here we note that

$$\sum_{n=1}^{N}\langle\log p(i^n, x^n|\theta)\rangle_{p^{old}[i^n|x^n]} = \sum_{n=1}^{N}\langle\log p(x^n|i^n,\theta)p(i)\rangle_{p^{old}[i^n|x^n]}$$

As per the slides, plug in the formula for the normal distribution and note that the expected value is over the various classes and note that we can write σ_i rather than Σ_i (with power to the D for the determinant), to get:

$$\sum_{n=1}^{N}\sum_{i} p^{old}[i^n|x^n]\left(-\frac{1}{2\sigma^2}(x^n-m_i)(x^n-m_i)-\frac{1}{2}\log(2\pi)-\frac{D}{2}(\sigma_i^2)+\log p(i)\right)$$

Differentiate with respect to the variance for class i only, to get

$$\sum_{n=1}^{N} p^{old}[i^n|x^n]\left(+\frac{1}{2(\sigma^2)^2}(x^n-m_i)(x^n-m_i)-\frac{D}{2\sigma_i^2}\right)$$

setting to zero and simplifying gives

$$\sigma_i^2=\frac{1}{D}\sum_{n=1}^{N}\frac{pr[i^n|x^n]}{\sum_{n=1}^{N}pr[i^n|n^x]}(x^n-m_i)(x^n-m_i)$$

which is as required if we define $p^{n|i}$ to be the first term in the sum.

f. Explain the relation between the GM under Expectation Maximisation training, and the K-means algorithm.

2011 Q2

2011 Q3

a. Principal Components Analysis (PCA) is a method to form a lower-dimensional representation of data. For data points $x^n mn = 1, \dots, N$, define the matrix

$$X_n[x^1, x^2, \dots, x^N]$$

That is, for data points x with dimension D, then X is $D \times N$ dimensional. The data is such that the mean is zero, that is

$$\sum_{n=1}^{N} x^n = 0.$$

The covariance matrix of the data, S, has elements

$$S_{i,j} = \frac{1}{N} \sum_{n=1}^{N} x_i^n x_j^n$$

i) Explain how to write S in terms of matrix multiplication of X.

S is a $D \times D$ matrix. Form it as XX^T. Note first the dimension is correct since this is $(D \times N) \times (N \times D) = (D \times D)$. And the elements are correct for example the top left element of XX^T is made up of the first row of X which is the first entries of all the $x^(n)$ times the first column of X^T which is also all the first entries of all the x^n.

ii) PCA is based on a linear model of the data

$$x^n \approx My^n.$$

where M is a $D \times H$ dimensional matrix and each y^n is a H dimensional vector, with $H \leq D$. With reference to an orthogonal matrix $R^T R = I$ explain why there is no unique setting for M and y^n. Use also a diagram to explain the geometric meaning of this result.

Given some solution approximation for x^n by H vectors in the columns on M with H weights, y^n. Consider another solution MR^T with weights Ry^n. Then this new solution gives the approximation

$$(MR^T)(Ry^n) = M(R^T R)y^n = MIy^n = My^n$$

which is the same. Hence the original solution can be rotated, the weights adjusted and the result is unchanged. Therefore there is no unique setting for M and y^n, however the approximations $\tilde{x}^n$ are unique.

Diagram should show a plane and two bases in the plane achieving the same result.

iii) PCA is typically described in terms of the eigen-decomposition of S. Explain how this procedure works and also discuss the computational complexity of performing PCA based on directly computing the eigen-decomposition of S.

$$S = \frac{1}{N-1} XX^T$$

- Centre the data (and transform to equal variances if relevant).

- Form the empirical covariance matrix
- Find the eigen-decomposition
- Choose the top K eigenvectors in order of decreasing eigenvalue
- The reconstruction error is $N-1$ times the sum of eigenvalues for eigenvectors not used, divided by the sum of eigenvalues (which is also the empirical covariance). The % of total variance explained is the the sum of eigenvalues for eigenvectors used divided by the sum of all the eigenvalues.

S is a $D \times D$ matrix and so computing the eignevalues is of order $O(D^3)$ (Indirectly takes $O(N^3)$ to find eigenvectors of $X^T X$ and then $O(ND)$ to recover those of S)

iv) Whilst there is no unique setting to the best lower dimensional linear representation, PCA nevertheless provides a unique setting for the lower dimensional representation. Explain why this is and what additional criterion PCA uses over least squares reconstruction error.

If we require that the basis vectors chosen maximise the variance of the projection and that this is done in order (i.e. the first basis vector maximised the variance of all one basis projections), then the required basis is exactly the eigenvectors of S in order of reducing eigenvalues and rotations cannot be freely applied. (Known as Maximal Variance / Principal Directions).

v) An alternative way to perform PCA is based on the SVD of the matrix X. Explain why this is related to the eigen-decomposition of S and explain the computational complexity of this approach to performing PCA compared to directly computing the eigen-decomposition of S.

$$X = UDV^T$$

and

$$(N-1)S = XX^T = (UDV^T)(UDV^T)^T = UDV^TVDU^T = UD^2U^T.$$

So the eigenvectors of S are the same as the columns of the U matrix in the SVD of X.

S is a $D \times D$ matrix. The complexity of finding eigen-values

Complexity of SVD are $O(\min(D^2N, DN^2))$. Assuming $D \gg N$ SVD on X is $O(D^2N)$

S is a square $D \times D$ matrix. And complexity of eigen-decomposition is $O(D^3)$ which can be significantly worse than $O(D^2N)$.

vi) The matrix X, with elements $X_{i,n}$ represents the real-valued rating for a user n given to film i. Only a small fraction of the elements of the matrix X are known since each user has only seen a small number of films. Explain how PCA with missing data can be used to "complete" the missing entries in X, thereby forming a prediction for the rating of a user to the missing entries in X. Explain also a method to implement PCA in this missing data case.

If we can find a set of basis vectors b_j (placed in a column matrix b) and a set of weights y_j^n for each b_j for each example (n), then we can reconstruct each example with By_n. The non missing values can then be taken from the reconstruction.

To implement PCA in this case:

- Chose random starting point for B

- Fix B and find solution for best Y - the error function is a quadratic in Y and the solution can be found in closed form
- Fix Y and find best B - again the error function is a quadratic in Y and the solution can be found in closed form
- Iterate until convergence

b. PCA can be considered a form of matrix factorisation. An alternative matrix factorisation method is probabilistic latent semantic analysis (PLSA) (also called non-negative matrix factorisation). This takes a positive matrix X whose entries all sum to 1:

$$\sum i, jX_{i,j} = 1,\ 0 \leq X_{i,j} \leq 1$$

and forms an approximation based on

$$X_{i,j} =\approx \sum_{k=1}^{H} U_{ik} V_{kj}$$

for positive U and V with $\sum_i Uik = 1$ and $\sum_k V_{kj} = 1$.

i. In the lectures, we compared the application of PCA and PLSA on a set of face images. Explain what are the typical characteristics of the 'eigenfaces' compared with the 'plsa' faces.

The PSLA faces are more localised and one of them, for example, might be clearly emphasising aspects of the chin. The eigenfaces are more general and emphasise broader aspects of the image. (Note though that overall, PLSA must have a higher reconstruction error since it has more constraints (positive bases).)

ii. Describe a way to train PLSA based on an interpretation of X, U and V in terms of probability distributions.

N pairs of two types of objects. First set of objects has $1, \ldots, I$ things in it and second has objects $1, \ldots, J$. Count matrix C has entries C_{ij} the number of times i found in x AND j found in y. Form a frequency matrix

$$p(x = i, y = j) = \frac{C_{ij}}{\sum_{ij} C_{ij}}$$

2011 Q4

2012 Q1

2012 Q2

2012 Q3

a. Explain why PCA is often used as a pre-processing step in machine learning and explain what the geometric meaning of PCA is.

Often, when the data is high dimensional, the key features that we are interested in will lie in a low dimensional subspace. For example when numbers are represented by a 100×100 pixel picture, there will be very many pixel combinations that will never represent a number (one such example is for each alternate pixel to be off). It makes sense then to find the a lower dimensional space which approximates the examples. Besides speeding up learning, it might also make the result more stable. The geometric meaning of the result is a plane which minimises the square distances of the original data to the plane.

b. Explain how to write S in terms of matrix multiplication of X.

$$S = \frac{1}{N-1} X X^T$$

where the N data vectors are the N columns of X

c. PCA is typically described in terms of the eigen-decomposition of S. Explain how this procedure works and also discuss the computational complexity of performing PCA based on directly computing the eigen-decomposition of S.

- Centre
- Adjust to make variance of all dimensions the same
- Find eigen-decomposition (in decreasing order of eigenvalues). Take the first K as required. B say.
- Coordinates are $y_k^n = B^T x^n$.
- Reconstruction is $\tilde{X} = BY = BB^T X$.
- Squared reconstruction error is $(N-1)$ times sum of eigenvalues not used. % reconstruction error is sum eigenvalues not used over total sum of eigenvalues.

Complexity based on eigen-decomposition of S is $O(D^3)$.

d. Consider the situation in which the data points x_n are very sparse - that is, only a few elements of each vector x^n are non-zero, resulting also in a sparse matrix S. Describe a computationally efficient procedure to estimate the principal direction (the largest eigenvector of S) and explain why this is efficient.

Doing S directly is $O(D^3)$.

[Start: From DB One way to compute the principal eigenvector of a matrix A is by repeated multiplication. If we start with a non-zero vector x, then while not converged

- x = S*x
- $x = x./\|x\|$ (this normalises the vector to make it unit length)
- end

This process will converge to the principal eigenvector (eigenvector with the largest eigenvalue). This is particularly efficient for the case that S is sparse since each matrix-vector product will be fast. End: From DB]

Each Sx creates a D dimensional vector and needs dD multiplications for each item in the vector so in total dD^2 (where $d\%$ is the sparsity factor). If d is very small, this will be far smaller than the $O(D^3)$ to calculate all of the eigenvectors using standard non-sparse techniques.

e. Continuing with the sparse data point scenario, can you conceive a technique that would enable one to perform full PCA (not just the principal eigenvector) efficiently?

[AC: I think we could] First find eigenvector of the largest eigenvalue as above (call this b^1). Then deflate the existing data by this vector. ie For each example (x^n), find its coordinate along this direction with $(b^1)^T x^n$ Reconstruct approximation as $(b^1)^T x^n b^1$ and deduct this from x^n to get the remaining unexplained aspects of the example. Now repeat the whole process (ie find S of the deflated data etc).

f. An alternative way to perform PCA is based on the singular value decomposition (SVD) of the matrix X. Explain why this is related to the eigen-decomposition of S and explain the computational complexity of this approach to performing PCA compared to directly computing the eigen-decomposition of S.

Consider SVD of $X = UDV^T$. Then $(N-1)S = (N-1)XX^T = (N-1)UDV^TVDU^T = (N-1)UD^2U^T$.

Hence the eigenvectors of S are the same as the columns of the left matrix of the SVD of X. (Also the eigenvalues of S are the squared singular values of the SVD of X.)

Complexity of SVD is $O(\min(DN^2, D^2N))$ so if $D \ll N$ this will be $O(D^2N)$ and will be much less than $O(N^2)$

g. PCA can be considered as representing the i^{th} component of the n^{th} data point using

$$x_i^n \approx \sum_{j=1}^{J} y_j^n b_{ji}$$

where b_{ji} are the elements of the j^{th} basis vector, and y_j^n is the corresponding coefficient. In the case that some of the x_i^n are missing, we cannot find the optimal PCA solution by the standard eigen-approach. In this case define the least squares objective:

$$E(B, Y) = \sum_{n=1}^{N} \sum_{d=1}^{D} \gamma_d^n \left[x_d^n - \sum_j y_j^n b_{ji} \right]^2$$

where γ_d^n is an indicator variable for x_d^n not being missing (it is zero if it is missing).

Derive a procedure for minimising E(B,Y) that is guaranteed to decrease the objective function at each stage of the iteration, and for which each iteration corresponds to the solution of a set of linear equations.

We look to minimise the loss function:

$$E[Y, B] = \sum_{n=1}^{N} \sum_{d=1}^{D} \gamma_d^n \left(x_d^n - \sum_{k=1}^{K} y_k^n b_d^k \right)$$

There is no probabilistic aspect to this problem. Simply fix B and maximise over Y, fix Y and maximise over B and then iterate. First look at fixing the basis vectors and differentiating with respect to the coordinates for an example n (the sum over n disappears since the differential is zero with respect to the sum over other examples):

$$\frac{\partial E[Y]}{y_j^n} = 2\sum_{d=1}^{D} \gamma_d^n \left(x_d^n - \sum_{k=1}^{K} y_k^n b_d^k \right) b_d^j$$

where the b are considered fixed and known.

$$\implies \sum_{d=1}^{D} \gamma_d^n x_d^n b_d^j = \sum_{k=1}^{K} \sum_{d=1}^{D} \gamma_d^n b_d^k b_d^j y_k^n$$

For each example, this is a system of equations of the form $M^n y^n = c^n$ where the k entries of c^n are $\sum_{d=1}^{D} \gamma_d^n x_d^n b_d^j$, the k entries for y^n are the k coordinates for example n i.e. y_k^n and finally the entries for the $K \times K$ matrix M^n are $\sum_{d=1}^{D} \gamma_d^n b_d^{row} b_d^{col}$.

2012 Q4

2013 Q1

1. This question concerns nearest neighbour methods.

i. Explain what is meant by nearest neighbour classification for a dataset of N examples, $\mathcal{D} = \{(x^n, c^n), n = 1, \ldots, N\}$, for D-dimensional inputs x and discrete class labels c. Explain how to classify a new input x.

We classify new examples based on the class of the nearest neighbour (or the K nearest neighbours). In general, classify new example, x, to (I don't remember notation so just make something up) $c^*(x)$ where

$$c^*(x) = \underset{c}{\operatorname{argmax}} \sum_{x^n \in K_{nn}(x)} I_{[c^n = c]}.$$

Using some metric to define distance.

ii. For the Euclidean distance

$$d(x, y) = \sqrt{\sum_{d=i}^{D} (x_d - y_d)^2}$$

and the dataset $\mathcal{D}$ above, describe the computational complexity (both time and storage) of computing the nearest neighbour classifier for a novel input x.

Storage - need to store all the data $O((D+1)N)$ (plus one is for the label). Calculations, need the distance from the novel point to all N examples. $O(ND)$. If want also to consider how to choose nearest, to a comparison to the best distance after each calculation. Then actually it is $O(ND+N)$ and presumably, unless we are efficient, for K-NN, we need $O(ND+NK)$

iii .Explain what is meant by a metric distance and show that the Euclidean distance is a metric.

- $d(x, y) \geq 0$ and $d(x, y) = 0 \iff x = y$
- $d(x, y) = d(y, x)$
- $d(x, y) \leq d(x, a) + d(a, y)$

First two are easy to show. For third

$$\begin{aligned}
\|x - y\|^2 =& \langle x - y, x - y \rangle = \langle x - a + a - y, x - a + a - y \rangle \\
=& \langle x - a, x - a \rangle + \langle a - y, a - y \rangle + 2\langle x - a, a - y \rangle \\
=& \|x - a\|^2 + \|y - a\|^2 + 2\|x - a\|\|y - a\| \cos\theta \\
\leq& \|x - a\|^2 + \|y - a\|^2 + 2\|x - a\|\|y - a\| \\
=& (\|x - a\| + \|y - a\|)^2
\end{aligned}$$

and so, taking square-root of both sides

$$|x - y\| \leq \|x - a\| + \|y - a\|$$

i.e.

$$d(x,y) \leq d(x,a) + d(a,y)$$

b. Orchard's algorithm is a way to speed up the calculation of the nearest neighbour (for a metric distance) of a query q to a set of D-dimensional training vectors $\{x^1, ..., x^N\}$. i. For a metric distance show that if $d(q,xi) \leq \frac{1}{2}d(x_i,x_j)$ then $d(q,x_i) \leq d(q,x_j)$. Draw a picture to describe this mathematical result.

Note that from what we are given we can write $2d(q,xi) \leq d(x_i,x_j)$. This shoud remind us that here we need to start the triangle inequality with $d(x_i,x_j)$. i.e. $d(x_i,x_j) \leq d(q,x_i) + d(q,x_j)$ and so

$$2d(q,x_i) \leq d(x_i,x_j) \leq d(q,x_i) + d(q,x_j)$$

and therefore

$$d(q,x_i) \leq d(q,x_j)$$

Can draw 1-D example. q on a line. x_i somewhere near. Radius centred at q to x_i on one side and continuing to other side. x_j being twice the distance away will fall outside of radius on either side.

ii. Explain in detail how Orchard's algorithm works.

Made up below from memory. Notation prob not very good. Don't rely on it!

- Calculate all pairwise difference between examples
- For each example, create two lists which are the distances (in increasing order) of the other $N-1$ examples and their index form the example.
- Choose a starting point $cand_1$ and calculate its distance to the novel point, q. Call this $d(cand_1, q)$
- Choose the first distance in the list. Call this $d(cand_2, q)$.
- $k = 2$
- Iterate
- If $d(cand_{k-1}, q) \leq \frac{1}{2}d(cand_k, cand_{k-1})$ then $cand_{k-1}$ is the NN and stop.
- Find $d(cand_k, q)$
- If $d(cand_k, q) \leq d(cand_{k-1}, q)$ then: $k = k+1$, choose the next candidate from this list. Call it $cand_k$. Iterate
- If $d(cand_k, q) \geq d(cand_{k-1}, q)$ then: $k = k+1$, Move to beginning of list of $cand_k$. Iterate
- If not otherwise terminated, then terminate when reach end of a list and that is the NN.

iii. Give an example dataset and query for which Orchard's algorithm will not be faster than simply explicitly evaluating the nearest neighbour by computing all distances from the query point q.

As per lecture notes. Points on a line. Equidistant (say). Novel point at one end of line. Start (bad luck!) from other end of line. Will move to new list each time and only reach the end once in the last list (where the triangle inequality will terminate on the first go.

iv. Give an example dataset and query for which Orchard's algorithm will be faster than simply explicitly evaluating the nearest neighbour by computing all distances from the query point q.

A few clusters of points far away from each other. If initialise in correct cluster will never need to evaluate point in further clusters. If initialise in wrong cluster, still chance of jumping to right cluster and missing other bad clusters.

c. Consider a metric distance and a set of data points $x_i, i \in I$ for which $d(q, x_i)$ has already been computed. i. Show that we form the lower bound:

$$d(q, x^j) \geq \max_{i \in I}\{d(q, x^i) - d(x^i, x_j)\} \equiv L_j$$

Note the triangle inequality hiding in the above and start with it

$$d(q, x^i) \leq d(x_i, x_j) + d(q, x^j)$$

and then

$$d(q, x^i) - d(x_i, x_j) \leq d(q, x^j)$$

and since this is true for all i the result follows.

ii. Explain how the Approximating and Elimination Algorithm (AESA) uses the above result to attempt to speed up the computation of the nearest neighbour of a query point q.

A candidate is chosen and a distance $d(cand, q)$ is evaluated. The above bounds can then be calculated and are used to speed up the computation in two ways. Firstly any point whose lower bound is higher that the current best distance can be moved to the "checked" list without a computation. Secondly the most promising candidate can then be chosen - ie the one with the smallest lower bound.

d. The KD tree is a hierarchical data-structure that can be used to potentially speed up nearest neighbour search. i. Explain how to form a KD tree, giving an example of a dataset (of two-dimensional data) and the corresponding KD tree.

Split on one dimension at the time. Based on median in that direction. Keep median at the branch and send other point to the left or right depending if $\leq$ or $>$ median. Send single points to leaf. And each branch, keep a note of which dimension we are splitting on. (4,5), (7,3), (-1,2), (0,10), (5,5), (6,6), (2,12) Goes to (4,5) top branch - split on d-1 ≤ 4 or > 4 Right side splits at (5,5) on d-2 ≤ 5 and > 5 sending (7,3) left and (6,6) right. Left side splits at (0,10) on d-2 ≤ 10 and > 10 sending (-1,2) left and (2,12) right.

ii. Consider a query vector q. Let's imagine that we have partitioned the datapoints into those with first dimension x_1 less than a defined value v (to its "left"), and those with a value greater or equal to v (to its' "right")

Let's also say that our current best nearest neighbour candidate is $x_i \in \mathcal{R}$ (and that this point has squared Euclidean distance $\delta^2 = (q - x^i)^2$. from q. Show that $(v - q_i)^2 \geq \delta^2$, then $(x - q)^2 > \delta^2$.

(I think we are meant to assume that q is also on the right and that the candidate point is on the left.) We have that the distance from the candidate point is made up along all dimensions, specifically the one we want to break on and so must be a least as big as the distance in that direction, i.e:

$$(x-q)^2 = \sum_{d=1}^{D}(x_d - q_d)^2 \geq (x_1 - q_1)^2$$

and along this distance since x is on the left and q is on the right the cross term below is made up of two negatives and is therefore positive:

$$(x_1 - q_1)^2 = (x_1 - v + v - q_1)^2 = (x_1 - v)^2 + (v - q_1)^2 + 2(x_1 - v)(v - q_1) \geq (v - q_1)^2$$

So if $(v - q_1)^2 \geq \delta^2$ then the distance to a from the candidate point is greater than the existing distance.

iii. Explain how the above result can be used with the KD tree to search for the nearest neighbour of a query. Use your example KD tree above and an example query point to describe how to find the nearest neighbour using the KD tree.

Send candidate down to leaf and calculate distance from leaf. Move up to next node. If the squared distance from q to the point at which the split occurs (along that dimension) is more than the distance from the leaf, then there is no need to check any point on the other side of the node. This way, it can be possible not to have to check many points. (Still need to continue moving up tree though until all other points or checked or this shortcut kicks in).

For example above, q is (7,2) say. Goes down to (7,3) node. Distance to leaf is 1. Distance to split along dimension 2 is $(5-3)^2$ is 4 so no need to check other side of node. Also distance to (5,5) is more so no point in going up either. Hence search ends.

2013 Q2

a. Consider a dataset of inputs $\{x^n, n = 1, \ldots, N\}$.

i. Show that the transformation

$$z^n = x^n - \mu$$

where

$$\mu = \frac{1}{N}\sum_{n=1}^{N} x^n$$

results in a new dataset that has zero mean, that is

$$\sum_{n=1}^{N} z^n = 0$$

We have:

$$\sum_{n=1}^{N} z^n = \sum_{n=1}^{N}(x^n - \mu) = \sum_{n=1}^{N} x^n - \sum_{n=1}^{N}\mu = \sum_{n=1}^{N} x^n - N\mu = \sum_{n=1}^{N} x^n - \frac{N}{N}\sum_{n=1}^{N} x^n = 0$$

ii. For zero mean data $\sum_{n=1}^{N} x^n = 0$, describe a “scaling” transformation

$$z_i^n = \lambda_i x_i^n$$

that makes each component of z have unit variance, that is

$$\frac{1}{N}\sum_{n=1}^{N}(z_i^n)^2 = 1$$

We have

$$\frac{1}{N}\sum_{n=1}^{N}(z_i^n)^2 = \frac{1}{N}\sum_{n=1}^{N}(\lambda_i x_i^n)^2 = \frac{1}{N}\lambda_i^2\sum_{n=1}^{N}(x_i^n)^2 = 1$$

and so

$$\lambda_i^2 = \frac{N}{\sum_{n=1}^{N}(x_i^n)^2}$$

and

$$\lambda_i = \sqrt{\frac{N}{\sum_{n=1}^{N}(x_i^n)^2}}$$

iii. For zero mean data $\sum_{n=1}^{N} x^n = 0$, describe and explain the computational cost of finding a “whitening” transformation matrix M

$$z^n = Mx^n$$

that makes z have a unit covariance. That is

$$\frac{1}{N}\sum_{n=1}^{N} z^n (z^n)^T = I$$

We have:

$$\frac{1}{N}\sum_{n=1}^{N} z^n (z^n)^T = \frac{1}{N} M \sum_{n=1}^{N} x^n (x^n)^T M^T = M \left(\frac{1}{N}\sum_{n=1}^{N} x^n (x^n)^T \right) M^T = MSM^T$$

where S is the covariance matrix.

Apply an eigen-decomposition to S so $S = U\Lambda U^T$ (say) and then we have

$$MU\Lambda U^T M^T$$

and we simply choose

$$M = \Lambda^{-\frac{1}{2}} U^T$$

and then we have, as required

$$MU\Lambda U^T M^T = \Lambda^{-\frac{1}{2}} U^T U \Lambda U^T U \Lambda^{-\frac{1}{2}} = I$$

The cost of doing it this way is $O(D^3)$. Maybe there is a much better way?

iv. With the help of a diagram, explain the difference between the two above 'scaling' and 'whitening' transformations.

No idea what is being looked for here - and I don't know how to draw a diagram to show something has become not correlated (whilst still remaining possibly dependent). Could show some simply 2-D data which is clearly correlated but not centred at zero and has different x and y scales - could show that after the scaling, it is still clearly correlated.

b. Consider a Gaussian mixture model of D-dimensional data (with $\sigma^2 = 1$)

$$p(x) = \sum_{k=1}^{K} \frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}(x-\mu^k)^2\right)$$

i. On a 32 bit machine, Matlab considers 2^z equal to zero, for $z < -1075$. For a point x, the largest contribution to the mixture model is given by a particular k, namely (setting $\sigma^2 = 1$):

$$\frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}(x-\mu^k)^2\right)$$

Setting $\sigma^2 = 1$, show that for the mixture model to have a non-zero probability in Matlab, we must have (for the natural log)

$$D \leq \frac{2 \times 1075 \times \log 2}{\log(2\pi)}$$

We need

$$\frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}(x-\mu^k)^2\right) \geq 2^{-1075}$$

and we note that $\exp\left(-\frac{1}{2}(x-\mu^k)^2\right)$ is always less than or equal to one, so at the very least we must have

$$\frac{1}{(2\pi)^{D/2}} \geq 2^{-1075}$$

(although this is not necessarily sufficient to ensure non zero results in MATLAB, it is a minimum.) Take natural logs of both sides to get

$$\log_e\left(\frac{1}{(2\pi)^{D/2}}\right) \geq \log_e(2^{-1075}) \implies -\frac{D}{2}log_e(2\pi) \geq -1075\log_e 2$$

and so

$$D \leq \frac{2 \times 1075 \log_e 2}{log_e(2\pi)}$$

Estimate (without a calculator) is whatever you like - say $1075 \times 2 \times 2/3/2 \approx 800$

ii. Explain how the logsumexp trick can be used to compute the log likelihood of the mixture model.

This is interesting, especially as we just saw that the problem is not even just caused by the obvious exponent but can be caused simply by the dimensions. So I think we need to get everything into the exponent ie

$$\begin{aligned} p(x) &= \sum_{k=1}^{K} \frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}(x-\mu^k)^2\right) \\ &= \sum_{k=1}^{K} \exp\left(-\frac{D}{2}\log(2\pi) - \frac{1}{2}(x-\mu^k)^2\right) \end{aligned}$$

Now we can take logs

$$log(p(x)) = \log\sum_{k=1}^{K} \exp\left(-\frac{D}{2}\log(2\pi) - \frac{1}{2}(x-\mu^k)^2\right)$$

Consider the terms in the exponent as a set of positive $a_1, \ldots a_k$. Find the minimum of these, $a_{k'}$ and then

$$\begin{aligned} log(p(x)) =& log\left(e^{-a_{k'}}\sum_{k=1}^{K} e^{+a_{k'}}\exp(-a_k)\right) \\ =& \log\left(e^{-a_{k'}}\right) + \log\left(\sum_{k=1}^{K} e^{+a_{k'}}\exp(-a_k)\right) \\ =& -a_{k'} + \log\left(\sum_{k=1}^{K} \exp(+a_{k'} - a_k)\right) \end{aligned}$$

so that for at least one term, the exponent is zero, and for all other terms it is less. Hence the most important term is calculated as accurately as possible.

iii. On a 32 bit machine, Matlab considers 2^z equal to infinity, for $z > 1024$. Show that the maximal dimension D for which the log likelihood is computable in Matlab is

$$D \leq \frac{2^{1025}}{\log(2\pi)}$$

and comment on the difference between this maximal dimension, and the maximal dimension of the raw probability itself.

Similarly to previously, other than this time we are working with log lik we have the need to calculate

$$\log \frac{1}{(2\pi)^{D/2}} \exp\left(-\frac{1}{2}(x-\mu^k)^2\right)$$

This time, given that the second part of the LHS is always less than one, if we ignore it, we come up with a sufficient condition and we have the need to calculate

$$\log \frac{1}{(2\pi)^{D/2}} = -\frac{D}{2}\log(2\pi)$$

Assuming MATLAB first calculates

$$\frac{D}{2}\log(2\pi)$$

and then negates it, then to avoid overflow we need

$$\frac{D}{2}\log(2\pi) \leq 2^{1024} \implies D \leq \frac{2^{1025}}{\log(2\pi)}$$

This limit is not really a problem - the limit based on raw probabilities was very limiting in real life terms.

2013 Q3

2013 Q4

Consider binary classification problems with class $c \in \{0, 1\}$. LogIstic Regression models the probability of the D-dimensional input vector x being in class $c = 1$ as

$$p(c = 1lx, w) = \sigma(w^T x)$$

for input vector x and weight (parameter) vector w, where

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

The dataset is $\mathcal{D} = \{(c^n, x^n), n = 1, \ldots, N \}$

a. Assuming that the data is independently and identically distributed, show that the log likelihood is given by

$$L(w) = \sum_{n=1}^{N} \big(c^n \log \sigma(w^T x^n) + (1 - c^n) \log \sigma(-w^T x^n)\big)$$

Main thing to remember is that the differential of $\sigma(x)$ is given by $\sigma'(x) = \sigma(x)(1-\sigma(x))$ (which is easy enough to show but takes time in an exam).

Also note that

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

Likelihood is

$$\prod_{n=1}^{N} pr(c^n = 1|x, w)^{c^n} \times (1 - pr(c^n = 1|x, w))^{(1-c^n)}$$

Then we have that the log of the likelihood is

$$L(x; w) = \sum_{n=1}^{N} \mathbb{1}_{\text{class=1}} \log pr(c^n = 1|x, w) + \mathbb{1}_{\text{class=0}} \log pr(c^n = 0|x, w)$$

and using c^n and $1 - c^n$ as the indicators, we get

$$L(x; w) = \sum_{n=1}^{N} c^n \log \sigma(w^T x) + (1 - c^n) \log(1 - \sigma(w^T x))$$

and remembering that $(1 - \sigma(w^T x) = \sigma(-w^T x)$ we have

$$L(x; w) = \sum_{n=1}^{N} (c^n \log \sigma(w^T x) + (1 - c^n) \log \sigma(-w^T x))$$

as required.

b. Show that the gradient of the log likelihood is given by

$$g_i \equiv \frac{\partial L}{\partial w_i} = \sum_{n=1}^{N} (c^n - \sigma(w^T x^n)) x_i^n$$

We have (using the chain rule twice)

$$\begin{aligned}
\frac{\partial L}{\partial w_i} =& \frac{\partial}{\partial w_i} \sum_{n=1}^{N} (c^n \log \sigma(w^T x) + (1-c^n) \log \sigma(-w^T x)) \\
=& \sum_{n=1}^{N} c^n \frac{1}{\sigma(w^T x)} \sigma(w^T x)(1-\sigma(w^T x)) x_i^n + (1-c^n) \frac{1}{\sigma(-w^T x)} \sigma(-w^T x)(1-\sigma(-w^T x))(-x_i^n) \\
=& \sum_{n=1}^{N} c^n (1-\sigma(w^T x)) x_i^n + (1-c^n)(1-\sigma(-w^T x))(-x_i^n) \\
=& \sum_{n=1}^{N} \left(c^n(1-\sigma(w^T x)) - (1-c^n)(1-\sigma(-w^T x))\right) x_i^n \\
=& \sum_{n=1}^{N} \left(c^n(1-\sigma(w^T x)) - (1-c^n)\sigma(w^T x)\right) x_i^n \\
=& \sum_{n=1}^{N} \left(c^n - c^n\sigma(w^T x) - \sigma(w^T x) + c^n\sigma(w^T x)\right) x_i^n \\
=& \sum_{n=1}^{N} \left(c^n - \sigma(w^T x)\right) x_i^n
\end{aligned}$$

as required.

c. Show that the Hessian is given by

$$H_{ij} = \frac{\partial^2}{\partial w_i \partial w_j} = -sum_{n=1}^{N} \sigma(w^T x^n)\sigma(-w^T x^n) x_i^n x_j^n$$

Differentiate the gradient with respect to w_j to get

$$\frac{\partial}{\partial w_j} \sum_{n=1}^{N} \left(c^n - \sigma(w^T x)\right) x_i^n = -\sum_{n=1}^{N} \sigma(w^T x)(1-\sigma(w^T x)) x_i^n x_j^n$$

and given that $(1-\sigma(w^T x)) = \sigma(-w^T x)$ the result follows.

d. d. Show that the Hessian is negatlve (semi) definite. That is, for any non-zero vector y

$$y^T H y \leq 0$$

and explain what this means in terms of the geometry of the log likelihood.

We could claim this by recognising that for each example, the function is the equivalent of a feature map and so must be PSD, and that the sum of PSD matrices is PSD. Alternatively

$$\begin{aligned} y^T H y &= (y_1, \ldots y_D)^T H (y_1, \ldots y_D) \\ &= \sum_i \sum_j y_i y_j \left(-\sum_{n=1}^N \sigma(w^T x^n)\sigma(-w^T x^n) x_i^n x_j^n \right) \\ &= -\sum_{n=1}^N \sum_i \sum_j \sigma(w^T x^n)\sigma(-w^T x^n)(x_i^n y_i)(x_j^n y_j) \end{aligned}$$

Note that for each n, $\sigma(w^T x^n)\sigma(-w^T x^n)$ is some positive number, p_n say and it is fixed over i and j so just take it out of that part of the sum and it will not affect the sign of things

$$y^T H y = -\sum_{n=1}^N p_n \sum_i \sum_j (x_i^n y_i)(x_j^n y_j)$$

and we recognise this as

$$-\sum_{n=1}^N p_n \left(x_1^n y_1 + x_2^n y_2 + x_3^n y_3 + \ldots\right)^2$$

which therefore in total is ≤ 0, as required. i.e. the Hessian is psd.

I imagine there is a far easier way of showing this!

This means that the log-lik is a concave function and that batch gradient ASCENT converges provided that the step size is small enough.

e. Explain how Newton's method (using the Hessian) can be used to find the maximum likelihood parameter vector w.

Initialise w_1. Update $w_{k+1} = w_k - \epsilon H^{-1}(w)|_{w_k} \nabla f(w)|_{w_k}$

f. Describe the computational complexity of a single Newton update, both in terms of time and storage cost. For D-dimensional inputs x, comment on the practical usefulness of the Newton method for high dimensional inputs $D \gg 1$.

To create the Hessian,if we store the $\sigma(w^T x^n)$ then "only" costs $O(D^2 + ND)$ (D^2 for the $x_i x_j$ terms and then ND for the N examples with $w^T x$ each time) To invert the Hessian is $O(D^3)$. To calculate the gradient requires $O(ND)$. To multiply the inverse of the Hessian by the gradient costs $O(D^2)$.

Main storage is the Hessian itself which is $O(D^2)$.

Not practical for large D.

g. Conjugate gradients is an alternative way to find the optimal maximum likelihood w. Given a search direction p_k at the k^{th} iteration of the algorithm, conjugate gradients needs to maximise

$$L(w_k + \alpha p_k)$$

with respect to α. Explain whether the optimal α can be found in closed form and, if not, how the optimal α can be obtained.

Unlikely to be available in closed form. Not a quadratic and log-lik highly non linear.

Use line search.

h. For sparse data in which only on average S components of an input vector x^n will be non-zero. i. Explain the computational complexity of computing the gradient vector g. ii. Give an estimate of the number of operations required to find the optimal weight vector using conjugate gradients.

$O(SN)$. Maximum of D updates (?) each costing $O(SN)$. $O(SND)$. No inverse. Assuming line search not costly. (note sometimes the notation in exams is to give a % sparsity being $s\%$ or $d\%$, then you would have $O(sDND)$ above.)

i Describe what is meant by "batch" versus "online" gradient methods.

For batch we calculate the full gradient based on all N examples, before doing an update. In "online" methods we make a move after approximating the gradient with less than N examples. The expected value of the update is the same as the full batch update. In the extreme we can make a move based on the gradient estimated from just one example.

j. Consider maximum likelihood learning for w in logistic regression using gradient ascent. i. Explain if "online" gradient ascent will converge for linearly separable training data. ii. Explain if "online" gradient ascent will converge for training data that is not linearly separable.

Not sure what is wanted here. For separable we have the perceptron type situation. Once the separating hyperplane has been found, the error function will be zero and updates will stop. Otherwise, although there is actually a global minimum to be found, online will not stop, because relative to the subset of points that are chosen each time, there is a chance that the gradient is not zero.

2014 Q1

This question follows the content in the first set of slides (AMLOptimisationPublic).

Consider the linear regression problem for training data with vector input x^n and scalar output $y^n, n = 1, \ldots, N$:

$$w_{opt} = \underset{w}{\operatorname{argmin}} E(w)$$

where we define the regularised total square loss

$$E(w) = \sum_{n=1}^{N} (y^n - w^T x^n)^2 + \lambda w^T w$$

a. Derive an explicit expression for the optimal weight vector w_{opt} in terms of the training data and regularisation constant A. Give also an estimation for the computational complexity required to find w_{opt} using this expression.

Say that dimension of the xs are D. Define the matrix X as a $N \times D$ matrix which has the whole of example x^n in row n. Define y as a $N \times 1$ vector of the y^i. Then we can express the sum of squares in the above expression as

$$(y - Xw)^T (y - Xw).$$

We can multiply this out to give

$$y^T y - y^T X w - w^T X^T y + w^T X^T X w.$$

Each of these terms is a scalar as hence the same as its transpose. In particular we can write $y^T X w = w^T X^T y$ and so the above is equal to

$$y^T y - 2 w^T X^T y + w^T X^T X w.$$

In total we have

$$E(w) = y^T y - 2 w^T X^T y + w^T X^T X w + \lambda w^T w$$

We can differentiate with respect to w to get

$$\frac{\partial E(w)}{\partial w} = -2 X^T y + 2 X^T X w + 2 \lambda w$$

which is zero at a critical point so

$$X^T y = (X^T X + \lambda I_{m \times m}) w_{opt}$$

and finally

$$w_{opt} = (X^T X + \lambda I_{m \times m})^{-1} X^T y.$$

Finding this value directly requires the inversion of a square matrix which has the same dimension as w hence is $(dim(w)^3)$ (or $O(D^3)$).

b. Describe the gradient descent procedure for finding the minimum of $E(w)$, explaining also any potential practical advantages or disadvantages of this approach.

We move in the direction of the steepest descent, i.e. minus the direction of the gradient. The amount that we move in this direction is controlled by a step size parameter.

Start at any point w_0. Update:

$$w_{k+1} = w_k - \epsilon(\nabla E(w))|w_k$$

where the notation means $(\nabla E(w))|w_k$; evaluate the gradient of $E(w)$ at the w value of w_k. ϵ is a parameter to adjust the step size. Note that w_{k+1} and w_k are vectors.

Advantages:

- Easy to calculate updates
- Only need to store gradient vectors - not large
- There is a convergence guarantee (though only of order $O(1/T)$)
- Can overshoot and can end up zig-zagging
- May make slow progress if ϵ is set too low
- Path of least descent depends on dimensions used (not invariant to linear transforms

c) Describe the Newton procedure for finding the minimum of E(w), explaining also any potential practical advantages or disadvantages of this approach. You should give an explicit update formula for the new weight vector in terms of the old weight vector so that this could be implemented directly by a computer programmer.

The Newton updates are

$$w_{k+1} = w_k - \epsilon(H_{E(w)})^{-1}|w_k(\nabla E(w))|w_k$$

Advantages:

- Does not depend on the coordinate system
- Gets the answer in one iteration for quadratic
- More "accurate" direction than gradient descent
- Can be very expensive to calculate the inverse of the Hessian (or to solve the linear system) and indeed to store the Hessian

We need all the entries of the gradient and of the Hessian.

For the gradient, from above:

$$\frac{\partial E(w)}{\partial w} = -2X^T y + 2X^T Xw + 2\lambda w = 2\left((X^T X + \lambda I)w - X^T y\right)$$

and differentiating again to get the Hessian we have:

$$H = 2(X^TX + \lambda I)$$

which does not depend on the current iteration value of w_k. The full update is given by

$$w_{k+1} = w_k - \epsilon(X^TX + \lambda I)^{-1}\left((X^TX + \lambda I)w_k - X^Ty\right)$$

The X and y are as described above and based on user input. ϵ is a parameter provided by the user. Matrix inversion is assumed available.

Unsurprisingly (I hope), since $E(w)$ is a quadratic, this simplifies to

$$w_{k+1} = w_k - \epsilon w_k + (X^TX + \lambda I)^{-1}X^Ty$$

and if $\epsilon = 1$ then this finds the minimum directly in one step (as the notes say is true for Newton and psd quadratics).

d) Describe the conjugate gradients procedure for finding the minimum of E(w), explaining also any potential practical advantages or disadvantages of this approach. You should give an explicit update formula for the new weight vector in terms of the old weight vector so that this could be implemented directly by a computer programmer.

First note that when minimising $E(w)$ (as a function of w) we can ignore y^Ty and we have a standard quadratic

$$f(w) = w^T(X^TX + \lambda I) - w^T(2X^Ty).$$

Let $A = 2X^TX + \lambda I$ and $b = 2X^Ty$ and the above is $\frac{1}{2}w^TAw - x^Tb$.

For conjugate gradients we need to find a series of vectors p_i where each p_i is normal to all the previous vectors in the series. We then move in this direction, i.e.

$$w_{k+1} = w_k + \alpha_k p_k.$$

Note than when doing this, the updated value for the function (excluding the constant we ignored) is

$$f(w_k + \alpha_k p_k) = (w_k + \alpha_k p_k)^TA(w_k + \alpha_k p_k) - (w_k + \alpha_k p_k^T)b$$

which is

$$w_k^TAw_k - w_k^Tb + 2w_k^TA\alpha_k p_k + (\alpha_k p_k)^TA(\alpha_k p_k) - \alpha_k p_k^Tb$$

Since w_k is made up of a linear combination of all the ps up to $k-1$, $2w_k^TA\alpha_k p_k$ is zero and we have

$$f(w_{k+1}) = f(w_k) + \alpha_k^2 p_k^TAp_k - \alpha_k p_k^Tb$$

and we need to find the α_k which minimises the RHS.

We recognise the RHS as a quadratic in α with a minimum at $\frac{p_k^Tb}{p_k^TAp_k}$

Therefore programmer should

- set $w_0 = 0$
- Iteratively find the conjugate gradients p_k for $k = 1, \ldots, m$

- For each gradient, find the weight as per the above formula
- Once all the weights are found, $w_{opt} = \sum_{k=1}^{m} \alpha_k p_k$

I have not memorised the iterations for finding conjugate gradients, but from the notes this is:

- Set first p_1 to $-g_1$ (minus the gradient)
- Use p_1 to find w_2
- Then set β_1 to norm of new gradient over norm of old gradient
- Set p_2 to -minus new gradient + $\beta_1 \times p_1$ Iterate

e) Consider the case that each training vector x is sparse, with only $0 \leq s \leq 1$ of the elements of the vector being non-zero. Give estimates for the computational complexity of finding w_{opt} based on the above procedures, namely (batch) gradient descent, Newton's method and conjugate gradients.

Batch gradient descent. $O(sDN)$ (note gradient $= -2\sum n = 1^N(y^n - w^T x^n)x_j + 2w_j)$. But then needs as many iterations as required. (Gradient needs X^TXw Do not form X^TX since is not necessarily sparse. Form Xw needs $O(sDN)$)

Newton. Once off cost $O(D^3)$ (inverse of non sparse matrix)- not affected by sparsity. No iterations.

Conjugate gradients. I think this is $O(NsD)$. This is because need p^TAp type of update (where A is based on X^TX). Can do this with Xp and then square it. Xp costs $O(NsD)$

f) Explain what is meant by an Auto-Regressive (AR) model for a time-series $y_1, \dots, y_T$ and derive an explicit expression for the optimal AR coefficients in the least squares sense.

An AR model is one where the mean of the current value for y^t is a linear combination of the previous l values for y. We also assume here that it is normally distributed with a scalar error term that has constant variance over time.

$$pr[y_t|y_{t-1}, /dots, y_{t-l}] = \mathcal{N}_{y_t}(\text{mean} = a^T\bar{y}_{t-1}, \text{variance} = \sigma^2)$$

where $\bar{y}_{t-1}$ is a vector of the last l observations before time t.

Using the standard form for the univariate normal distribution we have:

$$\log pr[y_t|y_{t-1}, /dots, y_{t-l}] = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y_t - a^T\bar{y}_{t-1})^t(y_t - a^T\bar{y}_{t-1})$$

So for all of the data from time 1 to time T (and not worrying too much about the y for negative times existing) the log lik over all the data is

$$\sum_{t=1}^{T} -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y_t - a^T\bar{y}_{t-1})^t(y_t - a^T\bar{y}_{t-1})$$

Differentiating with respect to the whole of the vector of a's gives:

$$\sum_{t=1}^{T}(y_t - a^T\bar{y}_{t-1})\bar{y}_{t-1} = 0$$

where we have dropped coefficients like the σ^2. Hence

$$\sum_{t=1}^{T}(y_t\bar{y}_{t-1}) = \sum_{t=1}^{T}(a^T\bar{y}_{t-1}\bar{y}_{t-1})$$

This is actually confusing because there are two of the $\bar{y}_{t-1}$ vectors next to each other, which is not nice. But since the first is multiplied by a^T and therefore is just a number, we can write this number AFTER the second $\bar{y}_{t-1}$ i.e.

$$\sum_{t=1}^{T}(y_t\bar{y}_{t-1}) = \sum_{t=1}^{T}(a^T\bar{y}_{t-1}\bar{y}_{t-1}) = \sum_{t=1}^{T}\bar{y}_{t-1}\bar{y}_{t-1}^T a$$

and then we have a matrix that we can invert and so we get:

$$a = (\sum_{t=1}^{T}\bar{y}_{t-1}\bar{y}_{t-1}^T)^{-1}\sum_{t=1}^{T}(y_t\bar{y}_{t-1})$$

as needed

2014 Q2

a. For the squared loss objective function

$$E[\mathcal{W}] = \sum_{n=1}^{N} (y^n - f(x^n|\mathcal{W}))^2$$

derive an efficient recursive procedure to compute the gradient of $E(\mathcal{W})$ with respect to all the parameters $\mathcal{W}$.

[See AC notes for a description of notation and diagram]

- $n = E$. $t_E = 1$. No children
- $n = h_i^L = \sigma_L(a_i^L)$. One child, $c = E$.

$$t_n = \sum_c \frac{\partial f_c}{\partial f_n} t_c = \frac{\partial E}{\partial h_i^L} = -2\sum_{k=1}^{d^L}(y_k^n - h_k^L)$$

- $n = a_i^L$. One child $c = h_i^L = \sigma_L(a_i^L)$. t_c is as directly above

$$\begin{aligned} t_n =& \sum_c \frac{\partial f_c}{\partial f_n} t_c \\ =& \frac{\partial h_i^L}{\partial a_i^L} t_c \\ =& \frac{\partial h_i^L}{\partial a_i^L}\left(-2\sum_{k=1}^{d^L}(y_k^n - h_k^L)\right) \\ =& \sigma_L^{'}(a_i^L)\left(-2\sum_{k=1}^{d^L}(y_k^n - h_k^L)\right) \\ =& A_i^L \end{aligned}$$

- The weights with respect to this is the differential of the input of this layer, wrt the weight, w_{ij}^L which is the output of the previous layer that feeds this weight i.e. h_j^{l-1}, times the result from above. i.e.

$$\frac{\partial E}{\partial w_{ij}^L} = h_j^{l-1} A_i^L.$$

- $n = h_j^{L-1} = \sigma_L(a_j^{L-1})$. Many children, all input nodes to next layer, $c = a_k^L$. The differential of each child with respect to the parent is the weight which joints them,

w^L_{kj}

$$\begin{aligned} t_n &= \sum_c \frac{\partial f_c}{\partial f_n} t_c \\ &= \sum_{k=1}^{d^L} \frac{\partial a^L_k}{\partial h^{L-1}_j} A^L_k \\ &= \sum_{k=1}^{d^L} w^L_{kj} A^L_k \end{aligned}$$

- Now output of this layer with respect to signal input. This is, as previously, $\sigma'_{L-1}(a^{L-1}_j)$ and so

$$\begin{aligned} \frac{\partial E}{\partial a^{L-1}_j} &= sigma'_{L-1}(a^{L-1}_j) \sum_{k=1}^{d^L} w^L_{kj} A^L_k \\ &= A^{L-1}_j \end{aligned}$$

- As before, can now differentiate signal input with respect to weight to get

$$\frac{\partial E}{\partial w^{L-1}_{ij}} = h^{L-1}_j A^{L-1}_i$$

- The above is the complete backprop recursion. (In general replace $L-1$ with $l-1$.)

b. Explain what is meant by an autoencoder neural network.

An autoencoder is an NN which aims to learn a lower dimensional representation of the inputs. It does this by setting the inputs to be the same as the outputs and having a bottleneck hidden layer with a small number of nodes. The part of the NN upto the bottleneck is an encoder, and from the bottleneck to the outputs is a decoder.

c. Consider an autoencoder with a single hidden layer ($L = 2$). When the transfer function at the output layer is the identity, $\sigma^L(x) = x$, derive an expression for the optimal weight matrices W^2, W^1 and relate this to Principal Components Analysis. What would be the optimal weights for an autoencoder with a larger number of layers, $L > 2$ but with the identity transfer function on the output layer?

Given transfer function in output layer is the identity. Given each y^n is D dimensional.

Consider the weights from the D dimensional input of a y into the K dimensional hidden code layer. Represent these weights as W^1, where row 1 contains the D weights from the inputs to hidden layer node 1. If Y represents the $D \times 1$ vector input then the activation for the hidden (code) later is $W^1 Y$. The output from the hidden layer is $\sigma(W^1 Y)$. We have a matrix of weights from the hidden to output layer with the top row being the K weights that take hidden layer outputs to the activation for the output later and activations for output layer are $W^2 h_1 = W^2 \sigma_1(W^1 Y)$. If hidden layer output is linear, then $\tilde{Y} = W^2 \sigma_1(W^1 Y)$.

Output is $\tilde{Y}$, squared loss is $\|Y - \tilde{Y}\|^2$. If general, if we wish to approximate a matrix with a matrix times something, the solution to minimise square loss is to take the largest

K singular values (and related vectors) from $Y = USV^T$. This can easily be achieved by setting W^2, the weights to the output layer, to be U_K (the first K columns of U). This leaves us needing $\sigma_1(W^1Y) = S_K V_K^T$. But $Y = USV^T$ so if we set the outputs to linear and set $W^1 = U_K^T$ this will work, since the limited K vectors we have in U_K are orthogonal to those in U and the zeros will propagate, leaving $S_K V_K^T$ as needed.

The resulting weights are the SVD of Y which is the same as PCA.

For a larger number of layers, but with a linear output layer, the result is the same. The number of hidden layers will be irrelevant and the product of their weights needs to be U.

2014 Q3

i) Explain what is meant by nearest neighbour classification for a dataset of N examples, $\mathcal{D} = \{(x^n, c^n), n = 1, \ldots, N\}$, for D-dimensional inputs x and discrete class labels c. Explain how to classify a new input x.

This means classification of a data point according to the class which is most frequent amongst k-nearest neighbours. Nearest means as measured by some distance metric. Ties can be broken at random or by increasing k.

ii) For the Euclidean distance

$$d(x,y) = \sqrt{\sum_{i=1}^{D}(x_i - y_i)^2}$$

and the dataset $\mathcal{D}$ above, describe the computational complexity (both time and storage) of computing the nearest neighbour classifier for a novel input x.

Storage $O((D+1)N)$ need to store all the data - ie D dimensions plus the class, for each example. Calculation need to find the distance between new input and each example $O(ND)$.

iii) Explain what is meant by a metric distance and show that the Euclidean distance is a metric.

A metric has three properties.

- Symmetric $d(x,y) = d(y,x)$ - obvious.
- $d(x,y) \geq 0$ and $d(x,y) = 0 \iff x = y$ - also obvious ($d(x,x) = 0$, and also given $d(x,y) = 0$, square the RHS)
- $d(x,y) \leq d(x,a)d(a,y)$.

Need to prove the last with the triangle inequality.

$$\begin{aligned}\langle x-y, x-y\rangle^2 &= \langle (x-a)+(a-y), (x-a)+(a-y)\rangle^2 \\ &= \langle x-a, x-a\rangle + \langle a-y, a-y\rangle + 2\langle x-a, a-y\rangle^2\end{aligned}$$

Since $\langle x-a, a-y\rangle^2 = \|x-a\|\|y-a\|cos\theta$ and $cos\theta \leq 1$ the above leads to

$$\langle x-y, x-y\rangle^2 \leq \|x-a\|^2 + \|y-a\|^2 + 2\|a\|\|b\| = (\|x-a\| + \|y-a\|)^2$$

and so

$$\|x-y\| \leq (\|x-a\| + \|y-a\|)$$

i.e.

$$d(x,y) \leq d(x,a) + d(x,b)$$

b) Orchard's algorithm is a way to speed up the calculation of the nearest neighbour (for a metric distance) of a query q to a set of D-dimensional training vectors $x^1, ..., X^N$.

i) For a metric distance show that if $d(q,x^i) \leq \frac{1}{2}d(x^i,x^j)$ then $d(q,x^i) \leq d(q,x^j)$. Draw a picture to describe this mathematical result.

Diagram with x^i in centre. Start with $d(x^i,x^j) \leq \dots$ so that we will be able to replace this with $2 \times d(q,x^i)$ etc.

i.e.

$$d(x^i,x^j) \leq d(q,x^i) + d(q,x^j)$$

and given

$$d(q,x^i) \leq \frac{1}{2}d(x^i,x^j) \implies 2d(q,x^i) \leq d(x^i,x^j)$$

so

$$2d(q,x^i) \leq d(x^i,x^j) \leq d(q,x^i) + d(q,x^j) \implies d(q,x^i) \leq d(q,x^j)$$

ii) Explain in detail how Orchard's algorithm works.

Orchard:

- Calculate the distances of every example from every other example
- For each $x^n, n \in 1,\dots,N$ form an ordered list $L_n(1,\dots,N-1)$ which indexes the examples in order of increasing distance from x^n
- Get new data point, q say.
- Chose one of the examples at random. x^j say.
- Fetch $d(q,x^j)$ from distances
- Fetch $d(q,x^{L_j(1)})$. If this is more than $2 \times d(q,x^j)$ then every other point is guaranteed to be further from q than x^j and algorithm terminates.
- Otherwise if it is less than $d(q,x^j)$, choose it and move to its list and start iterating down its list.
- Otherwise if it is more than $d(q,x^j)$, then stick to same list and move up to next point.
- If not otherwise terminated, terminate when reach the end of a list and the nearest point is the point at the end of that list.

iii) Give an example dataset and query for which Orchard's algorithm will not be faster than simply explicitly evaluating the nearest neighbour by computing all distances from the query point q.

From slides. List of 5 points on the line. And we happen to choose first. And novel point is on the other end of the line. Then each time we move up one, we move to the new list - so we never benefit from being half the distance.

iv) Give an example dataset and query for which Orchard's algorithm will be faster than simply explicitly evaluating the nearest neighbour by computing all distances from the query point q.

Same again, with point in the middle of the line (this will always be faster, regardless of where we start, I think)

c) Consider a metric distance and a set of data-points x_i, $i \in I$ for which $d(q, x^i)$ has already been computed.

i. Show that we may form the lower bound

$$d(q, x^j) \geq \max_{i \in I}\{d(q, x^i) - d(x^i, x^j)\}$$

Using $d(q, x^i)$ we can form a triangle inequality as usual

$$d(q, x^i) \leq d(q, x^j) + d(x^i, x^j)$$

and then can convert this into a lower bound for $d(q, x^j)$ i.e.

$$d(q, x^j) \geq d(q, x^i) - d(x^i, x^j).$$

Since this is true for all $i \in I$ we must have

$$d(q, x^j) \geq max_{i \in I}\{d(q, x^i) - d(x^i, x^j)\}.$$

ii) Explain how the Approximating and Elimination Search Algorithm (AESA) uses the above result to attempt to speed up the computation of the nearest neighbour of a query point q.

It uses this result in two ways. Firstly any example with lower bound greater than the current best, does not need to be checked and can be moved from "not examined" to "examined" without doing a distance calculation.

Also when choosing the next point to examine, the one with the lowest bound in chosen.

d.) The KD tree is a hierarchical data-structure that can be used to potentially speed up nearest neighbour search.

i. Explain how to form a KD tree, giving an example of a dataset (of two-dimensional data) and the corresponding KD tree.

Iteratively split left and right along alternate axes etc. As per slides

ii. Let's also say that our current best nearest neighbour candidate is x^i and that this point has squared Euclidean distance $\delta^2 = (q - x^i)^2$ from q. Show that if $q_j \geq v$ and $(v - q_j)^2 > \delta^2$, then $(x - q)^2 \geq \delta^2$.

So along the first dimension, q is to the right of v. We must be talking about other candidates x which are to the left of v.

$$(x - q)^2 = \sum_{d=1}^{D}(x_d - q_d)^2 = (x_1 - q_1)^2 + \text{ other non negative terms} \geq (x_1 - q_1)^2.$$

Also note that

$$(x_1 - q_1)^2 = (x_1 - v + v - q_1)^2 = (x_1 - v)^2 + (v - q_1)^2 + 2(x_1 - v)(v - q_1)$$

Since x is to left of v, $(v - q_1)$ is negative and similarly because q_1 is to the right of v, $(v - q_1)$ is also negative. So the cross product term is positive. Hence the whole of the RHS $\geq (v - q_1)^2$, so

$$(x - q)^2 \geq (x_1 - q_1)^2 \geq (v - q_1)^2 \geq \delta^2$$

as needed.

iii. Explain how the above result can be used with the KD tree to search for the nearest neighbour of a query. Use your example KD tree above and an example query point to describe how to find the nearest neighbour using the KD tree.

Basically, once the distance from the best candidate so far to the query is so small that any point on the other side of the latest split cannot be closer, we can terminate the search. Hence move up to parent and check for this. If it is not the case we need to go down the other side and iteratively search it. Otherwise go up one and repeat. Keep on going, always replacing candidate with anything better and stop when epsilon ball does not go over boundary.