Inf2B Learning Summary and Equations

Version 1.0
May 2018
Ben Southall

1 Notation

Generally I have followed the notation used in the slides and the lecture notes. However I have made a few small deviations when I personally have found the lecture notes to cause confusion. In all cases, I follow the convention that when iterating or summing over dimensions, an upper-case letter represents the highest index or a set, and the corresponding lower-case letter represents an index. For example,

$$\sum_{d}^{D}(...) \qquad \text{for d } \epsilon \text{ D}$$

Generally, I use the following conventions for letters used as indices/sizes and sets:

d, D - dimensions of vector-space, normally the dimensions or individual components of a feature vector. When using a matrix, d corresponds to an element representing the dth dimension of a feature-vector, but this could be in either direction of the matrix itself. For document classification, however, these letters represent documents.

n, N - a number of a set, generally the training samples or test samples.

k, K - the classes in classification problems.

i,j - these are used in the notes but I prefer not to use i and j for general indices as I find it is quick to forget exactly what they are indexing. Therefore I will only use i and j when it is obvious that we are indexing over something for which there is no other clear choice of letter to use.

Typically in a summation or product the indices start at 1 and go to the number at the top. If there is any divergence from this it will be noted explicitly. For example, the following are all equivalent:

$$\sum_{n}^{N}(...) = \sum_{n=1}^{N}(...) = \sum_{n\epsilon N}(...)$$

This last one demonstrates my informal use of th capital letter as if it were a set, this does not mean N is itself a set but rather $n\epsilon N$ is a shorthand for $n\epsilon\{1..N\}$

The notes often use an alternative letter when we are considering alternative cases of the same type of thing, for wantt of better phrasing. For example, when considering a class k, the notes use the letter l to sum over all the other classes in K. In this case, I prefer to use k' to refer to a class that is not class k, so that I can remember what we're talking about. For example,

$$\frac{p(\mathbf{x}|C_k)P(C_k)}{\sum_{k'}^{K} p(\mathbf{x}|C_{k'})P(C_{k'})}$$

Here, as we're summing over all classes K but we're considering a class k in the numerator, so we use k' to index the other classes in the denominator.

Vectors will appear in bold face and in an exam I would underline them. Matrices in the notes appear in bold face, I will diverge from the notes' notation at times by using a capital bold letter in these notes for clarity.

I am using this notation because on a personal level I sometimes find I get confused in the lecture notes and have adopted my own personal style that is only slightly different but will help me remember what I am doing more clearly. I hope it makes sense to you too. My strategy in the exam will be to be explicit about what notation I am using and if you did choose to follow my notation that would be a good idea, however if in doubt I can only recommend following the notation used in lectures verbatim. I have not *yet* asked Hiroshi about how strict the notation will be marked.

2 Distance and Similarity Measures

2.1 Distance Measures

There are various ways of measuring how far two items are in D-dimensional vector-space. The *Manhattan* or *City-block* distance measures how far two items are from each other as if walking along a series of square blocks, as the streets are arranged in a modern city. It is called the L_1 *norm* and is calculated thus:

$$r_1(\mathbf{a}, \mathbf{b}) = \sum_{d}^{D} |a_d - b_d|$$

The Euclidean distance is the 'normal' distance measure for basic geometry that we are used to using,

$$r_2(\mathbf{a}, \mathbf{b}) = |a - b| = \sqrt{\sum_{d}^{D} (a_d - b_d)^2} = \sqrt{\mathbf{(a\text{-}b)} \cdot \mathbf{(a\text{-}b)}}$$

This measures the distance between two points, but is highly dependent on the units in question. We can normalise this to a similarity measure between 0 and 1 that is therefore independent of the scale of the numbers involved. For this, we use

$$sim(\mathbf{a}, \mathbf{b}) = \frac{1}{1 + r_2(\mathbf{a}, \mathbf{b})}$$

Thinking about how this works, when x and y are the same point, the Euclidean distance is 0 and the value is 1. When they are infinitely far away, the similarity is 0.

The Hamming distance between two items is simply the number of items thsat differ. The binary strings "01101001" and "10101010" have a Hamming distance of 4.

2.2 Correlation

Correlation is different from distance. Distance measures whether different points are close to each other, whereas correlation measures if they follow a general trend - if when x is large, y is too, and when x is small, y is too.

The Pearson Correlation Co-efficient is defined as

$$\rho(a,b) = \frac{1}{N-1}\sum_{n}^{N}\frac{(a_n-\mu_a)}{\sigma_a}\frac{(b_n-\mu_b)}{\sigma_b}$$

where μ is the mean and σ is the variance. Dissecting this, $(a_n - \mu_a)$ is the straight-line distance from a to the mean value of a, and this is normalised by the variance of a. Therefore we have a relative measure of the deviation of a point from the mean. Because we take the product of the left- and right-hand sides, if a is greater than its mean when b is greater than its mean, the result is positive and we have a positive correlation. Conversely, if a is greater than its mean when b is less than its mean, the correlation is negative. This is a bit like how we take the covariance, except the covariance is a measure of how spread-out the data values are, and the correlation is a measure of whether the values follow a similar trend.

When we are using vectors, we can represent this in the following way:

$$\frac{(\mathbf{x}-\bar{\mathbf{x}})\cdot(\mathbf{y}-\bar{\mathbf{y}})}{|x-\bar{x}\|y-\bar{y}|}$$

which is the same as the cosine of the angle between $x - \bar{x}$ and $y - \bar{y}$. This is one if they are both in the same direction (positive correlation), -1 if they are opposite (negative correlation) and 0 or very small if they are perpendicular (no correlation)

We can also express the correlation succintly as

$$\rho(\mathbf{a},\mathbf{b}) = \frac{COV(\mathbf{a},\mathbf{b})}{\sigma_a\sigma_b}$$

2.3 Recommender Systems

For recommender systems, we have a set of known relations between two variables, let's assume this is a matrix $\boldsymbol{M}$ of c against f, using the film example, c is a critic and f is a film, amd $M_{c,f}$ is the rating given by a critic to a film. We want to find a film that a user u will like. The most basic theory is to graph in multiple dimensions. Each critic c_i is a point in F dimensions, the value for each dimension being the value that the particular critic gave to that film. For example, if we have three critics, c_1, c_2, c_3 and c_4 and three films, f_1, f_2, and f_3, then let's say

$$\boldsymbol{M} = \begin{bmatrix} 5 & 4 & 2 \\ 1 & 3 & 3 \\ 4 & 1 & 4 \\ 1 & 2 & 5 \end{bmatrix} \Rightarrow \boldsymbol{f}_1 = \begin{bmatrix} 5 \\ 1 \\ 4 \\ 1 \end{bmatrix}, \boldsymbol{c}_1 = \begin{bmatrix} 5 & 4 & 2 \end{bmatrix}$$

To recap, in this case, we want to consider our *users* as the points, and the *items that are being reviewed* as the dimensions in this vector space. This is because we want to compare the location of a new user to that of the existing users, and use this to estimate what to recommend. We are recommending films to users, not users to films, so the film scores are the dimensions, and the users/critics are the points.

We use the Euclidean distance to work out how far existing critics' ratings are from the user's ratings, for those films the user has reviewed. We then normalise these using the similarity measure:

$$sim(\boldsymbol{c_u}, \boldsymbol{c_i}) = \frac{1}{1+\sqrt{\sum\limits_{f \epsilon F_u \cap F_i} (\boldsymbol{M}_{u,f} - \boldsymbol{M}_{i,f})^2}}$$

where $f \epsilon F_u, \cap F_i$ expresses that we want the Euclidean distance between critic i and the new user, for all those films that both have rated already. We can't have the Euclidean distance in dimensions representing a film that either one or both hasn't rated yet.

We could then recommend to the user the film that the closest critic most highly rated. Or we could use an average to smooth this out a bit and consider all critics. For a film f that the user has not yet rated, we can complete the gaps in the user's critic vector $\boldsymbol{c}_u$ to make a complete, estimated critic vector $\hat{\boldsymbol{c}}_u$:

$$\hat{\boldsymbol{c}}_{uf} = \frac{1}{C} \sum_{c}^{C} M_{c,f}$$

We average over all the critics their entry in the ratings matrix for that film, and assign this value to the gap in the user's entry. These two options aren't a very sophisticated system - using the closest critic's favourite film, if a critic gave a very bizarre rating for one film, this will enter into our system. And by averaging, we smooth it out but lose our sense of picking a close critic. So we combine these two into a method where we average, but weight our average by similarity:

$$\hat{\boldsymbol{c}}_{uf} = \frac{1}{\sum\limits_{c}^{C} sim(\boldsymbol{M}_c, \boldsymbol{M}_u)} \sum_{c}^{C} (sim(\boldsymbol{M}_c, \boldsymbol{M}_u) \boldsymbol{M}_{cf})$$

Or, using vector notation,

$$\hat{\boldsymbol{x}}_{uf} = \frac{1}{\sum\limits_{c}^{C} sim(\boldsymbol{x}_c, \boldsymbol{x}_u)} \sum_{c}^{C} (sim(\boldsymbol{x}_c, \boldsymbol{x}_u) \boldsymbol{x}_{cf})$$

By using correlation, instead of looking for the distance between film ratings, we look for how well film ratings occur in a pattern. For example, the vectors $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ and $\begin{bmatrix} 61 & 62 & 63 \end{bmatrix}$ have a large Euclidian distance measure but their values in each dimension are similar. If a critic ranks all films consistently lower than others, then a way to normalise the score is to take a measure from the mean of that critic's scores.

$$\bar{x}_c = \frac{1}{F} \sum_{f}^{F} x_{cf}$$

$$s_c = \sqrt{\frac{1}{F-1} \sum_{f}^{F} (x_{cm} - \bar{x}_c)^2}$$

If we then define the score for a critic c and a film f as z_{cf}, then this score has a mean of 0 and a (sample) standard deviation of 1:

$$z_{cf} = \frac{x_{cf} - \bar{x}_c}{s_c}$$

We can also use the Pearson Correlation Co-efficient.

$$r_{c_1c_2} = \frac{1}{F-1}\sum_{f}^{F}(z_{c_1f})(z_{c_2f}) = \frac{1}{F-1}\sum_{f}^{F}(\frac{x_{c_1f}-\bar{x}_{c_1}}{s_{c_1}})(\frac{x_{c_2f}-\bar{x}_{c_2}}{s_{c_2}})$$

This has the advantage that $-1 \leq r_{c_1c_2} \leq 1$ and so 1 can be added to give a similarity measure.

3 K-Means Clustering

Cluster analysis is different from classification. In classification, there is some inherent difference between members of different classes, and the learning is a supervised process. With clustering we are simply trying to group points to reduce the amount of data we have to handle. Hierarchical clustering uses a tree-like structure and can be top-down or bottom-up. Partitional clustering divides the input space into non-overlapping clusters. K-means clustering is an example of this.

K-means clustering to group D-dimensional vectors into K clusters:

Pick K random points to serve as cluster centres.
Assign each point to the nearest cluster.
Make each cluster centre the mean of its assigned vectors.
If the cluster centre has moved, repeat from the assignment stage.

We use the mean-squared error function to find the distances of points $\boldsymbol{x}_n$ from the cluster centre $\boldsymbol{m}_k$:

$$E = \frac{1}{N}\sum_{k}^{K}\sum_{n}^{N} z_{nk}|\boldsymbol{x}_n - \boldsymbol{m}_k|^2 \qquad z_{nk} = \begin{cases} 1 & \text{if } n \text{ belongs to class k} \\ 0 & \text{otherwise} \end{cases}$$

The mean-square error function measures how far points are from their cluster means, but not how far clusters are from each other. K-means clustering is guaranteed to converge on a local minimum of the error function, but not encessarily the global minimum error. Where data are split far away and the cluster centre is in the middle, we may end up with one very large cluster rather than two smaller ones. Also, a large cloud of data can gradually pull a small cluster's cluster centre away from a good location for it

$$\min_{z_{kn}} \frac{1}{N}\sum_{k}^{K}\sum_{n}^{N} z_{kn}|\boldsymbol{x}_n - \boldsymbol{m}_k|^2$$

4 Dimensionality Reduction

For dimensionality reduction, we want to take D-dimensional data and reduce it to a (e.g.) 2-dimensional plane that facilitates visualisation of the data. For this, we take the dot product with (two) unit vectors, here called $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$. For optimum viewing, we want the variance along these dimensions to be at maxima. Therefore the aim is the following maximisation problem:

$$\max_{\hat{\boldsymbol{u}},\hat{\boldsymbol{v}}} Var(u) + Var(v), \text{ subject to } \hat{\boldsymbol{u}} \perp \hat{\boldsymbol{v}}$$
$$\text{where } u_n = \hat{\boldsymbol{u}} \cdot \boldsymbol{x}_n, v_n = \hat{\boldsymbol{v}} \cdot \boldsymbol{x}_n,$$

$\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$ are the two eigenvectors with the largest eigenvalues of the covariance matrix, $\boldsymbol{S}$:

$$\boldsymbol{S} = \frac{1}{N-1} \sum_{n}^{N} (\boldsymbol{x_n} - \bar{\boldsymbol{x}})(\boldsymbol{x_n} - \bar{\boldsymbol{x}})^T$$
$$\bar{\boldsymbol{x}} = \frac{1}{N} \sum_{n}^{N} \boldsymbol{x}_n$$

Apparently calculating eigenvectors is beyond the scope of the course.

5 K-Nearest-Neighbour Classification

In K-NN Classification, we have a set of N classified examples, X, and a set Z of observed vectors to be classified.

for $\boldsymbol{z} \epsilon Z$
  for $\boldsymbol{x} \epsilon X$
    calculate distance $r(\boldsymbol{z}, \boldsymbol{x})$ between $\boldsymbol{z}$ and $\boldsymbol{x}$
  let $U_k(\boldsymbol{z})$ be a subset of X that contains the k nearest samples to $\boldsymbol{z}$
  let the class of $\boldsymbol{z}$, $c(\boldsymbol{z}) = \underset{c' \epsilon C}{argmax} \sum_{\boldsymbol{x}' \epsilon U_k(\boldsymbol{z})} \delta_{c' c(x')}$

$$\text{where } \delta_{ab} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

I.e. for each test point, for all the training data, we find the k closest points and pick the mode of the classes for these k nearest points.

6 Naïve Bayes

Bayes' theorem is a basic observation that we can calculate the probability that we observe a given that b is 'true' from the probability that we observe b given that a is 'true', and the probability of observing a in the first place. For classification, we express the probability that an observed feature vector $\boldsymbol{x}$ belongs to a class C_k from the probability that within class C_k we can find a feature vector $\boldsymbol{x}$ and the probsability that in any class we can find a feature vector $\boldsymbol{x}$.

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_k)P(C_k)}{\sum_{k'}^{K} p(\mathbf{x}|C_{k'})P(C_{k'})}$$

Here, $P(C_k|\boldsymbol{x})$ is the *posterior probability*, the probability that when we see $\boldsymbol{x}$, it belongs to class C_k. We normally assign $\boldsymbol{x}$ to the class with the highest posterior probability. $p(\boldsymbol{x}|C_k)$ is the *likelihood* that within class k we would find a feature vector $\mathbf{x}$, and $P(C_k)$ is the *prior probability*, the probability that

class k appears - this takes into account that some classes are more prevalent than others. If we give a photograph of a fruit to our model that has a height-to-width ratio equally matching the known ratios for a lemon and a dragonfruit, but we're in Scotmid, then it's most likely a lemon. Or, failing that, a bottle of Buckfast.

When we are faced with a classification problem, $p(\boldsymbol{x})$ can usually be omitted. This is because we have already observed x, now we need to decide on the most likely class for x. $p(\boldsymbol{x})$ makes no difference to the calculation of to which class we should assign $\boldsymbol{x}$. However in this case, we need to remember then that if we do omit this, the result is no longer a true probability. The results for all classes won't sum to unity.

For D dimensions, with Naïve Bayes, we assume that each dimension is independent of each other. Therefore,

$$P(\boldsymbol{x}|C_k) = P(x_1, x_2, ..., x_D|C_k)$$

$$= P(x_1|x_2, ..., x_D, C_k)P(x_2|x_3, ..., x_D, C_k)...P(x_D|C_k)$$

$$\simeq P(x_1|C_k)P(x_2|C_k)...P(x_D|C_k)$$

7 Document Classification

For document classification, we have two document models - the Bernoulli Model and the Multinomial Model. They are based on two mathematical distributions of the same names.

7.1 The Bernoulli, Binomial, and Multinomial Distributions

When we are simulating a boolean process, such as tossing a coin, we have two outcomes. If we represent one outcome as being true, then $k = 0$ when the outcome is false and $k = 1$ when the outcome is true; k represents the number of times the outcome is 'true'. When the probability that $k = 1$ is p, then if we have a random variable X, we can model X by the Bernoulli distribution:

$$P(X = 0) = p, P(X = 1) = (1 - p)$$
$$\text{therefore}$$
$$P(X = k) = kp + (1 - k)(1 - p) = p^k(1 - p)^{1-k}$$

We can then simulate the outcomes for a value of k (0 or 1).

When we repeat this process multiple times, we get the binomial distribution:

$$P(X = k) = \binom{n}{k} p^k(1 - p)^{n-k}$$

Here, we drop the restricion that $k\epsilon\{0, 1\}$, rather $k \leq n$ represents the number of times that the outcome is 'true' and p^k represents the chance of picking this outcome k times. The binomial co-efficient takes account of the fact that the order in which we pick the outcomes doesn't matter, as long as there are k of them. So we have repetitions.

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

$n!$ represents the number of ways of picking n items when the order of them matters, and we divide by $r!$ because we don't care about the order of the 'true' outcomes, and by $(n-r)!$ because we don't care about the order of the 'false' outcomes.

When we have more than two outcomes, let us have n items which can belong to one of D classes. I.e. the binomial model represents the special case where D=2, and the Bernoulli distribution represents the special case where D=2 and n=1. For a particular class d, let n_d represent the number of items of class d. We take these as repeats, i.e. the items of class d are indistinguishable from each other. Using the string "hello" as an example, n = 5, D = 4, and the l is repeated so n_l is 2. If we imagine selecting colourful stickers from a bag to spell out the word hello, it doesn't matter in which order we pick the 'l's.

Therefore, again, we have $n!$ permutations, but for each d, we have n_d permutations that are equivalent. The multinomial co-efficient is therefore:

$$\frac{n!}{\prod_d^D n_d!}$$

To model a probabilistic distribution then, let p_d represent the probability of drawing an item of type d. If n items are drawn at random from a large set, then let $\boldsymbol{x}$ model the distribution with x_d representing the number of times an item of type d is drawn. Therefore,

$$P(\boldsymbol{x}) = \frac{n!}{\prod_d^D n_d!} \prod_d^D p_d^{x_d}$$

Here, $p_d^{x_d}$ representd the probability of drawing an item of type d, the number of times in x_d we want to model. So the left-hand side, the multinomial co-efficient represents the number of equivalent permutations, and the right-hand side represents the probability of one permutation.

7.2 The Multinomial Document Model

For each model we have a vocabulary w consisting of V words or word stems that are of interest to us in classifying documents into different categories. For a document d in a test sample of D documents, using Naïve Bayes,

$$P(C_k|\boldsymbol{d}) \propto P(\boldsymbol{d}|C_k)P(C_k)$$

We estimate the probabilities of individual words:

$$\hat{P}(w_v|C_k) = \frac{\text{Number of documents in class k where } w_v \text{ is observed}}{\text{number of documents in class k}}$$

Class C_k has N_k documents in it, and document $\boldsymbol{D}_d^k$ is document d, belonging to class C_k.

In the Bernoulli model, document $\boldsymbol{D}_d^k$ is a feature vector in V dimensions, with each element, D_{dv}^k representing the number of ocurrences in the document

of word v. $n_k(w_v)$ represents the number of ocurrences of word v in documents observed in class k during training.

We find the probability that a word v occurs in a document of class k using the estimates from the training feature vectors. The the posterior probability that document d belongs to class k is estimated using Naïve Bayes, using a multinomial distribution on the probability of the individual words in the document. The multinomial model uses the probabilities of drawing each word found in the document from the vocabulary, using the calculated probabilities that these words would occur in class k documents. The number of draws is thus the length of the document.

$$\hat{P}(w_v|C_k) = \frac{\sum_{n}^{N_k} D^k_{nv}}{\sum_{v'}^{V} \sum_{n}^{N_k} D^k_{nv'}}$$

Here we are summing over all the documents of class k $\{1..N_k\}$ and counting the ocurrences of word v, and dividing it by the ocurrences of all words in documents of class k. Here, we use $\boldsymbol{D}^k_n$ to represent the set of documents of class k, however the lecture notes use the notation

$$\hat{P}(w_t|C_k) = \frac{\sum_{i}^{N} x_{it} z_{ik}}{\sum_{s}^{|V|} \sum_{i}^{N} x_{it} z_{ik}}$$

where $\boldsymbol{x}_n$ is a document that may or may not be in class k, and so z_{ik} is 1 when document $\boldsymbol{x}_n$ is in class k and 0 otherwise. In either case, they are both equivalent and equivalent to

$$\frac{n_k(w_v)}{\sum_{v'}^{V} n_k(w_{v'})}$$

For our multinomial model then, let $\boldsymbol{x}$ represent the feature vector for document d in class k, D^k_d. Recall that

$$\prod_{v}^{V} P(w_v|C_k)^{x_v}$$

gives the probability of drawing a single, unique string of words from the vocabulary that matches the document we see, and

$$\frac{n!}{\prod_{v}^{V} x_v!}$$

normalises this to take into account that firstly, we're not interested in the order the words come in, and secondly, when we have repeated words, there are several orders of drawing them all of which have the same result. Therefore,

$$P(\boldsymbol{x}|C_k) = \frac{n!}{\prod_v^V x_v!} \prod_v^V P(w_v|C_k)^{x_v}$$

Again, as this is a classification problem, the multinomial co-efficient doesn't depend on the class and so can be omitted. So,

$$P(C_k|\boldsymbol{x}) \propto P(\boldsymbol{x}|C_k)P(C_k) \propto (\prod_v^V P(w_v|C_k)^{x_v})P(C_k)$$

If we need the actual probability, then whilst the above expression is proportional to the probability but not equal to it,

$$\hat{P}(\boldsymbol{x}|C_k) = \frac{P(\boldsymbol{x}|C_k)P(C_k)}{\sum_{k'}^K P(\boldsymbol{x}|C_{k'})P(C_{k'})}$$

7.3 The Bernoulli Document Model

In the Bernoulli model, the feature vector $\boldsymbol{x} = \boldsymbol{D}_d^k$ represents whether or not each word is present in the document, and not the frequency of ocurrences. We use the Bernoulli distribution to model the chance of picking out the feature vector representing the document and then use Naïve Bayes as before.

$$P(\boldsymbol{x}|C_k) = \prod_v^V \left[x_v P(w_v|C_k) + (1 - x_v)(1 - P(w_v|C_k))\right]$$

As the model only takes into account presence or absence of a word, the estimated likelihood of the word is easier to calculate:

$$\hat{P}(w_v|C_k) = \frac{n_k(w_t)}{N_k}$$

That's simply the number of documents in which word v ocurrs over the total number of documents, all within class k. So for the Bernoulli model, $\hat{P}(w_v|C_k)$ represents the proportion of documents, whereas in the multinomial model, it represents the frequency of the word itself.

For both models, $P(C_k)$ is the number of documents of class k over the number of documents, $\frac{N_k}{N}$. A difference between the two models is that in the Bernoulli model, if a word does not occur in class k, that effects the probability of drawing document d, whilst in the multinomial model, only the presence of words in the vocabulary effects the probabilities.

8 Gaussian Classifiers

We use Gaussians because they fit a lot of data. When we observe data in many scenarios, they fit a 'bell curve' and this can be modelled by a Gaussian. So when we don't have enough samples to calculate a specific model for the data, we can assume that it fits a Gaussian distribution and this normally gives a satisfactory result. In one dimension, the Gaussian function is as such:

$$p(x|\mu,\sigma^2) = N(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

For multiple dimensions, the formula induces last night's dinner. Let's see it in action:

$$p(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

We estimate the parameters $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ from the sample data like so:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n}^{N} \boldsymbol{x}_n$$
$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N-1}\sum_{n}^{N} (\boldsymbol{x}-\hat{\boldsymbol{\mu}})(\boldsymbol{x}-\hat{\boldsymbol{\mu}})^T$$

For this exam, we will need to be able to find the inverse of a 2x2 matrix:

$$\boldsymbol{A}\begin{bmatrix} a & b \\ c & d \end{bmatrix} \Longrightarrow A^{-1} = \frac{1}{|\boldsymbol{A}|}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \frac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Using the covariance matrix, we can get the correlation co-efficient:

$$\rho(\boldsymbol{x}_i,\boldsymbol{x}_j) = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}\sigma_{jj}}}$$

Using a Gaussian to perform a basic classification,

$$P(C_k|x) \propto P(x|C_k)P(C_k) = N(x;\mu,\sigma^2)$$

Taking the natural logarithm of this gives the log likelihood,

$$LL(x|\mu_k,\sigma_k^2) = ln\ p(x|\mu_k,\sigma_k^2) \quad = \quad \frac{1}{2}(-ln(2\pi) - ln(\sigma_k^2) - \frac{(x-\mu_k)^2}{\sigma_k^2}$$

The log posterior probability is therefore

$$LL(x|C_k) + ln\ P(C_k) + \kappa$$

where κ is some constant

For two-class problems, we can take the ratio of these and base our decision on whether the result is greater than 1 or not, or for multiple classes, we can take the largest value and assign the feature vector to that class.

9 Discriminant Functions

The aim of a discriminant function is to define the boundary between regions associated with different classes. When deciding on a suitable discriminant function, we are trying to minimise the error rate, that is, the rate of misclassification.

$$P(Error) = \sum_{k}^{K} \sum_{k'}^{K} P(\boldsymbol{x} \epsilon \mathcal{R}_{k'} | k \neq k', x \epsilon C_k) P(C_k)$$

We sum the probability that $\boldsymbol{x}$ is misclassified into any region $\mathcal{R}_{k'}$ that is not the correct region, $\mathcal{R}_k$. For continuous probabilities, we would integrate over the region instead. Let's not.

For a discriminant function y_k, we assign $\boldsymbol{x}$ to class C_k if

$$y_k(\boldsymbol{x}) > y_{k'}(\boldsymbol{x}) \quad \forall\, k' \epsilon K, \quad k' \neq k$$

Using the log posterior probability as a discriminant function,

$$y_k(\boldsymbol{x}) = ln\ P(C_k|x) = ln\ p(x|C_k) + ln\ P(C_k) + \kappa$$
$$= -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) - \frac{1}{2} ln|\boldsymbol{\Sigma}_k| + lnP(C_k) + \kappa$$

If the dimensions of our feature vectors are independent from each other, and the covariance is class-independent, then this reduces to a linear function:

$$y_k(\boldsymbol{x}) = \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}_k^{-1} \boldsymbol{x}\ -\ \frac{1}{2} \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\mu}_k\ +\ ln\ P(C_k)$$

We can summarise this as follows:

$$\text{Let } \boldsymbol{w}_k^T = \boldsymbol{\mu}_k^T \boldsymbol{\Sigma}_k^{-1} \text{ and } w_{k0} = -\frac{1}{2} \boldsymbol{w}_k^T \boldsymbol{\mu}_k\ +\ ln\ P(C_k)$$

then

$$y_k(\boldsymbol{x}) = \boldsymbol{w}_k^T \boldsymbol{x} + w_{k0}$$

$\boldsymbol{w}_k^T$ is known as the weight vector, and w_{k0} as the bias for the class.

For a spherical Gaussian,

$$\Sigma^{-1} = \frac{1}{\sigma^2} \boldsymbol{I}$$

so

$$y_k(\boldsymbol{x}) = \frac{|\boldsymbol{x} - \boldsymbol{\mu}_k|^2}{2\sigma^2} + lnP(C_k)$$

$$= \frac{1}{2\sigma^2} [(\boldsymbol{x} - \boldsymbol{\mu}_k)^T (\boldsymbol{x} - \boldsymbol{\mu}_k)] + lnP(C_k)$$

When expanded, as $\boldsymbol{x^T x}$ is class-independent,

$$y_k(\boldsymbol{x}) = \frac{1}{2\sigma^2} (|\boldsymbol{\mu}_k|^2 - (\boldsymbol{\mu_k^T x} - \boldsymbol{x^T \mu_k})) + lnP(C_k)$$

10 Neural Networks

Right, get ready for this, kids.

A neural network can be used to estimate the weights for a discriminant function without needing to have an underlying mathematical model, such as the Gaussian distribution. Single-layer networks can only output linear discriminant functions, that is, hyperplanes through the D-dimensional vector space. But multi-layer networks can be non-linear and so can produce arbitrarily complex decision boundaries.

A neural network node can produce exactly one decision boundary, that is, it can only separate two things. It takes a feature vector $\boldsymbol{x}$ as input, multiplies it against a weight matrix and produces a scalar output, y. The lecture notes use a matrix for this and make it more complicated, this matrix simply represents multiple discriminant functions for multiple classes. We will just consider the single-class function. Using the discriminant functions from before,

$$y_k = w_{k0} + \boldsymbol{w_k^T x}$$

We modify $\boldsymbol{w}_k$ to add w_0, and modify $\boldsymbol{x}$ to add 1 do the above reduces to

$$y_k = \boldsymbol{w_k^T x}$$

There are different types of neural networks. For linear networks with the step activation function, y_k is 0 or 1, depending on which of the two classes the node assigns $\boldsymbol{x}$ to. In this case, we train the network with a series of training vectors (probably as a matrix), and a vector $\boldsymbol{t}$ where t_n is 0 if y_n should be 0 or 1 if y_n should be 1. Our aim is to choose the weights in w_k so as to minimise the number of misclasifications. For N samples in the training set, we use the sum-of-squares error function,

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_n^N |y_n - t_n|^2$$

$$= \frac{1}{2}\sum_n^N \left|\boldsymbol{w_k^T x_n} - t_n\right|^2$$

Because the output of the nodes is binary, we can't train the network based on this output, however we can train it using the error function as this is a smooth, continuous function. We want to find the gradient of the function,

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}_k) = (\frac{\partial E}{\partial w_{k0}} \ ... \ \frac{\partial E}{\partial w_{kd}})$$

The gradient is a vector in D dimensions representing the direction in which the error function increases most rapidly. Therefore, subtracting a small amount of this from the weights updates them to

For an iteration τ,

$$w_{ki}^{\tau+1} = w_{ki}^{\tau} - \eta\frac{\partial E}{\partial w_{ki}}$$

For a single node,

$$E(\boldsymbol{w}_k) = \frac{1}{2}\sum_n^N \sum_d^D (w_{kd}x_{nd} - t_n)$$

$$\frac{\partial E}{\partial w_{kd}} = \sum_n^N \sum_{d'}^D (w_{kd'}x_{nd'} - t_n)x_{nd}$$

$$= \sum_n^N (y_n - t_n)x_{nd}$$

$$= \sum_n^N (\delta_n)x_{nd} \text{ where } \delta_n = y_n - t_n$$

Thus if the output from the node matches the expected output for that sample, the value is 0 and the weight is not adjusted.

The following algorithm summarises this (sorry about the abhorrent spacing):

while not converged
    $\Delta w_{kd} = 0 \; \forall \; k, d$
    for $n \epsilon N$
        for $k \epsilon K$
            $y_{nk} = \sum_d^D w_{kd}x_{nd}$
            $\delta_n k = y_{nk} - t_n$
            for $d \epsilon D$
                $\Delta w_{kd} = \Delta w_{kd} + \delta_{nk}x_{nd}$
    for $k \epsilon K$
        $w_k = w_k - \eta w_k$

We can also estimate w_0 from the input data:

$$\bar{\boldsymbol{x}} = \frac{1}{N}\sum_n^N \boldsymbol{x}_n$$

$$\bar{t}_k = \frac{1}{N}\sum_n^N t_{nk}$$

$$w_{k0} = \bar{t}_k - \sum_d^D w_{kd}\bar{x}_d$$

This is the difference between the mean number of samples has class k and the network outputs of class k, over all the training samples.

Perceptrons are linear neural networks that produce a binary output. After applying the weights to the input, the summation phase, the output is passed into a function that returns 1 if the output is over a critical value or 0 if it is below that value. With the step function, no number of layers of the network can make any decision boundary that is not linear, but if we use a different function after the summation phase, we can make non-linear decisions and so can make arbitrarily complex decision boundaries.

What happens if we would like a neural network whose output estimates the posterior probability of the class? Then we would like an *activation function* that looks like the probability distribution for a two-class Gaussian. It turns out that the logistic sigmoid looks roughly like this, and also is differentiable over all input values, so it can be used in gradient descent training.

Let a be the activation value, then $a = \boldsymbol{w}^T \boldsymbol{x}$, then

$$y(\boldsymbol{x}) = g(a) = g(\boldsymbol{w}_k^T \boldsymbol{x}) = \frac{1}{1+e^{(\boldsymbol{w}_k^T \boldsymbol{x})}}$$

Note thet here, we are using w_k to be the vector including the bias w_0, and x to have the additional one prepended for this. For two classes, if

$$\boldsymbol{w}_k^T \boldsymbol{x} = ln \frac{P(\boldsymbol{x}|C_k)(PC_k)}{P(\boldsymbol{x}|C'_k)(PC'_k)}$$

then

$$P(C_k|\boldsymbol{x}) = y(\boldsymbol{x}) = g(a) = \frac{1}{1+e^{-a}}$$

This means for two-class problems, when using the logistic sigmoid, if the weights produced a log probability ratio, the output would actually be a posterior probability. It tuns out if we train our network well, using the sigmoid, the output of the neural network looks enough like a probability that we can use it as such.

We can train the network using gradient descent too, however the error function is different, so the derivative will be too.

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_n^N \sum_k^K (\ g(\boldsymbol{w}_k^T \boldsymbol{x}_n) - t_{nk}\ )^2$$

$$\frac{\partial E_n}{\partial W_{kd}} = \frac{\partial E_n}{\partial y_{nk}} \frac{\partial y_{nk}}{\partial a_n k} \frac{\partial a_{nk}}{\partial w_{kd}}$$

$$= \delta_{nk}\ g'(a_{nk})\ x_{nd}$$

$$\Rightarrow \frac{\partial E}{\partial w_{kd}} = \sum_n^N g(a_{nk})(1 - g(a_{nk}))\ \delta_{nk}\ \ x_{nd}$$

$$\Rightarrow w_{kd} = w_{kd} - \eta[g(a_{nk})(1 - g(a_{nk}))\ (y_{nk} - t_{nk})\ \ x_{nd}]$$

For multiple classes, we use the Softmax activation function instead.

$$P(C_k|\boldsymbol{x}) = y_k(\boldsymbol{x}) = g(a_k) = \frac{e^{(a_k)}}{\sum\limits_{k'}^{K} e^{(a_{k'})}} = \frac{e^{(\boldsymbol{w}_k^T \boldsymbol{x})}}{\sum\limits_{k'}^{K} e^{(\boldsymbol{w}_{k'}^T \boldsymbol{x})}}$$

For both of these, the output addds to 1 and each output is the posterior probability of class k.

For Softmax,

ust

$$\frac{\partial E}{\partial W_{kd}} = \sum_{k'}^{K} \delta_{nk'}\ y_{k'} I_{kk'} - y_{k'} y_k$$

where $I_{ab} = 1$ iff $a = b$ and 0 otherwise.

11 Appendix: JSome of the Lyrics to Cry for You by September

I never had to say goodbye
You must have known I wouldn't stay
While you were talking about our life
You killed the beauty of today
Forever and ever
Life is now or never
Forever never comes around
People love and let go
Forever and ever
Life is now or never
Forever's gonna slow you down
You'll never see me again
So now who's gonna cry for you
You'll never see me again
No matter what you do