MSA220 Statistical Learning for Big Data

Lecture 4

Rebecka Jörnsten

Mathematical Sciences
University of Gothenburg and Chalmers University of Technology

More on Discriminant analysis

More on Discriminant analysis

As mentioned in the previous lecture, one problem with LDA stems from the instability of the estimate $\hat{\Sigma}$ when n is small and/or p is large and/or x-features are correlated.
Let's look at the source of this problem in more detail.

Inverse of Σ

Consider the eigendecomposition of $\hat{\Sigma} = UDU'$, where U are the eigenvectors and $U'U = I$ and D is a diagonal matrix containing the eigenvalues

$$\begin{pmatrix} d_1^2 & 0 & \cdots \\ 0 & d_2^2 & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & 0 & d_p^2 \end{pmatrix}$$

where $d_1^2 > d_2^2 > \cdots$

Inverse of Σ

We can then write the inverse as

$$\hat{\Sigma}^{-1} = UD^{-1}U'$$

and plugging this into the expression for the DA rule

$$\hat{c}(x) = \arg\min_c (x - \hat{\mu}_c)' \hat{\Sigma}^{-1} (x - \hat{\mu}_c) =$$

$$= (x - \hat{\mu}_c)' UD^{-1}U' (x - \hat{\mu}_c) =$$

$$= [U'(x - \hat{\mu}_c)]' D^{-1} [U'(x - \hat{\mu}_c)] =$$

$$= (\tilde{x} - \tilde{\mu})' D^{-1} (\tilde{x} - \tilde{\mu}) =$$

$$= \sum_{j=1}^{p} \frac{(\tilde{x}_j - \tilde{\mu}_j)^2}{d_j^2}$$

which is a weighted euclidean distance between x and μ in the new coordinate system corresponding to the principal component directions U of $\hat{\Sigma}$.

Inverse of Σ

So LDA is really just nearest centroids in the new coordinate system that you get by rotating the data by U and scaling it by D. Writing $\hat{\Sigma} = UDU' = UD^{1/2}D^{1/2}U'$, we have that

$$\hat{\Sigma}^{-1} = UD^{-1/2}D^{-1/2}U' = \hat{\Sigma}^{-1/2}\hat{\Sigma}^{-1/2}$$

where we define $\hat{\Sigma}^{-1/2} = D^{-1/2}U'$ (square root of a diagonal matrix is just the square root of the elements).
Therefore we can write

$$(x - \hat{\mu}_c)'\hat{\Sigma}^{-1}(x - \hat{\mu}_c) = [\hat{\Sigma}^{-1/2}(x - \hat{\mu}_c)]'[\hat{\Sigma}^{-1/2}(x - \hat{\mu}_c)].$$

Inverse of Σ

The operation $\hat{\Sigma}^{-1/2}$ on x is called *sphering* the data. Why?

$$Cov(\hat{\Sigma}^{-1/2}X) = E[\hat{\Sigma}^{-1/2}X(\hat{\Sigma}^{-1/2}X)'] =$$

$$= E[\hat{\Sigma}^{-1/2}XX'\hat{\Sigma}^{-1/2}] = \hat{\Sigma}^{-1/2}E[XX']\hat{\Sigma}^{-1/2} = \hat{\Sigma}^{-1/2}\hat{\Sigma}\hat{\Sigma}^{-1/2} = I$$

I.e., in the new coordinate system Xs are uncorrelated and all features have variance 1.

Inverse of Σ

When $\hat{\Sigma}$ is near singular, $\hat{\Sigma}^{-1}$ behaves poorly (or may not even exist). The estimate is numerically unstable and small changes to the data can lead to big change for the inverse (and thus how you rotate the data before applying nearest centroids $\rightarrow$ poor classification performance.

The source of the problems lie in the direction u_j corresponding to small eigenvalues d_j since d_j appears in the denominator in the weighted euclidean distance computation. Small ds "blows up" the distance computation.

How do we fix this? The solution is to stabilize the inverse by reducing the influence of these small eigenvalues. This is done quite easily by simply adding something to the diagonal of $\hat{\Sigma}$ before you take the inverse.

Inverse of Σ

Use $\tilde{\Sigma} = (\hat{\Sigma} + \lambda I)$ and its inverse $\tilde{\Sigma}^{-1} = (\hat{\Sigma} + \lambda I)^{-1}$.
The impact of this is mainly limited to the small eigenvalues as we can see from the following

$$\hat{\Sigma} + \lambda I = UDU' + \lambda I = UDU' + \lambda UU' = U(D + \lambda I)U'$$

For large d_j the contribution λ is negligible.
Using $\tilde{\Sigma}^{-1}$ in your DA rule is called *penalized DA* (or regularized DA). When $\lambda = 0$ PDA is the same as LDA. If you make λ really big it starts to dominate the $d_j, \forall j$ which essentially means you start ignoring the correlation and scale structure in the data (get closer and closer to nearest centroids).

Flexible DA methods

Penalized DA addresses one problem with LDA, poor performance due to unstable estimates of $\hat{\Sigma}^{-1}$ (high variance). We also need to be concerned with potential BIAS, meaning the linear boundaries that LDA implicitly assumes are too simplistic to separate the classes from eachother.
One extension is then to use QDA (quadratic DA) we already looked at. This assumes that each class has its own correlation and scale structure. It leads to quadratic boundaries in x-space and is quite costly in terms of the number of parameters you need to estimate. This can reduce BIAS but lead to a large increase in VARIANCE so the end result is little or no improvement over LDA (or even worse performance if VARIANCE grows quickly as would be the case for very large p).

Mixture DA

A very nice alternative to QDA that generalizes LDA to more flexible boundaries is *mixture discriminant analysis* (MDA), introduced by Hastie and Tibshirani in the mid-90s.
We make the classifier more complex by allowing each class to be made up of many, simple components (as opposed to one complex component as in QDA). By combining many simple shapes we can build up quite complex shapes in x-space! Example: you can build a donut shape in x-space with 5-6 spherical distributions.

Mixture DA

We assume the following model for each class

$$p(x \mid y = c) = \sum_{r=1}^{R_c} \pi_{cr} N(x; \mu_{cr}, \Sigma)$$

Notice

- There are R_c components for class c and this may differ from class to class
- Each component has a different contribution or "weight" in the class distribution, π_{cr}
- Each component within and between the classes have the same shape, Σ.

For large p, the last bullet constitutes a large savings in terms of the number of parameters compared to QDA.

Mixture DA

As we saw before, estimating parameters in DA was relatively easy (just computing means, proportions and covariances). Here, the situation is more complex since we don't actually know which component r within class c that an observation belongs to. The components are artificial constructs that allows to generate complex data distribution shapes, but we don't know anything about them a-priori.

Mixture DA

For models like these we start by working out what we would do if we did know about the component memberships. Then things are just as easy as in standard DA. We would take all the observations in each component and compute the parameter estimates:

- For all classes c, compute for each component r within this class: $\hat{\mu}_{cr} = \sum_{i \in cr} x_i / N_{cr}, \quad N_{cr} = \sum_{i \in cr} 1$
- $\hat{\pi}_{cr} = N_{cr}/N_c, \quad N_c = \sum_{y_i = c} 1$
- $\hat{\Sigma} = \sum_{c=1}^{C} \sum_{r=1}^{R_{cr}} (x_i - \hat{\mu}_{cr})(x_i - \hat{\mu}_{cr})' / (N - C)$

So now we know what we would do if we knew which observations belonged to which component. Let's ask the other hypothetical question: what would we do if we knew all the parameters of the class components? If we knew those we could just apply the maximum posterior principle to classify observations at the *component* level rather than the class level:

$$\arg \max_{r \in \textit{class } c} p(i \in \textit{ component } r \textit{ of class } c \mid y_i = c, \mu_{cr}, \pi_{cr}, \Sigma)$$

To get something we can work with for the posterior we apply Bayes theorem:

$$\arg \max_{r \in \textit{class } c} p(i \in \textit{ component } r \textit{ of class } c \mid y_i = c, \mu_{cr}, \pi_{cr}, \Sigma) =$$

$$= \frac{\pi_{cr} N(x_i; \mu_{cr}, \Sigma)}{\sum_{l=1}^{R_c} \pi_{cl} N(x_i; \mu_{cl}, \Sigma)}$$

We usually denote this posterior by $\eta_{cr}(x_i)$.

Mixture DA

Let's put the two things together. Given a component classification we know how to estimate the parameters, given the parameters we know how to classify observations into the components. We simply iterate these two steps until the results converge.
How do you start off? Usually by a random selection of center points for each component and a nearest centroid classification. Then you start estimating the parameters etc. You may need to try a couple of different starting points in order to ensure convergence to the best fitting component distribution.

Mixture DA

The above iterative scheme is a variant of the so-called EM (Expectation-Maximization) algorithm which is a very important tool for dealing with models that have this additional complication - a bit of information is missing (component labels). The above algorithm is a variant called *classification* EM since I classify the observations into only one of the components.
Since the components are an artificial construct they are rarely well separated and then one might argue that using observations only for one component is an inefficient use of data if the components overlap and blend into each other.

The remedy for this is to use *weighs* in the parameter estimation and let those weights be the component posterior probabilities for each observation. This way, an observation can contribute to all components within a class, just more to the component it fits best to.
The weights are thus the $\eta_{cr}(x_i)$ we defined above.

Mixture DA

We use the weights to come up with another form for the parameter estimates:

- For all classes c, compute for each component r within this class: $\hat{\mu}_{cr} = \sum_{y_i=c} \eta_{cr}(x_i) x_i / N_{cr}, \quad N_{cr} = \sum_{y_i=c} \eta_{cr}(x_i)$
- $\hat{\pi}_{cr} = N_{cr}/N_c, \quad N_c = \sum_{y_i=c} \sum_{r=1}^{R_c} \eta_{cr}(x_i)$
- $\hat{\Sigma} = \sum_{c=1}^{C} \sum_{y_i=c} \frac{\sum_{r=1}^{R_{cr}} \eta_{cr}(x_i)(x_i - \hat{\mu}_{cr})(x_i - \hat{\mu}_{cr})'}{\sum_{y_i=c} \eta_{cr}(x_i)} / (N - C)$

EM algorithm

The EM-algorithm is something we will come back to when we do clustering.
Here, the E-step is the computation of the posterior probabilities that an observation belongs to a certain component within a class. This produces the weights $\eta_{cr}(x)$.
The M-step is the parameter estimation step where the weights are used to allow for observations within a class to contribute to the estimation of all the class components.

Validation

Both PDA and MDA can be tuned to be more or less flexible/local.
For PDA, you have to choose λ just big enough that the instability of the matrix inverse operation is surpressed (variance reduced) without affecting the rotation of the data in the leading principal component directions (would lead to bias).
For MDA, you have to choose the number of components for each class just big enough so that the class shapes are adapting to the data shape but not so big that you are adapting to random noise in the data or don't have enough observations to train the component parameters on.
As always in statistics - we have to consider the bias-variance trade-off!

Validation

Both PDA and MDA tuning parameter selection can be done via cross-validation.
PDA:

1. Split the data into B parts
2. For $b = 1, \cdots, B$
   For $\lambda = 0, \cdots, \lambda_{max}$
   1. Apply PDA with value λ to all data except the b-th test data
   2. Predict class labels on the b-th test data
   3. Compute the test error rate TE_λ^b
3. Compute the average error rate across folds b for each λ: $TE_\lambda = \sum_b TE_\lambda^b / B$
4. Choose the λ that minimizes this test error: $\lambda^* = \arg\min TE_\lambda$

Validation

For MDA it's slightly more complicated since you have to consider different number of components for each class. Let's denote a set of component numbers by $R = (R_1, R_2, \cdots, R_C)$, e.g. $R = (3, 4)$ if you use 3 components for class 1 and 4 components for class 4. Let's enumerate all sets R to consider as $R^m, m = 1, \cdots, M$

1. Split the data into B parts
2. For $b = 1, \cdots, B$
   For $m = 1, \cdots, M$
   1. Apply MDA, with component set $R^m = (R_1^m, R_2^m, \cdots, R_C^m)$, to all data except the b-th test data
   2. Predict class labels on the b-th test data
   3. Compute the test error rate TE_m^b
3. Compute the average error rate across folds b for each m: $TE_m = \sum_b TE_m^b / B$
4. Choose the m that minimizes this test error: $m^* = \arg\min TE_m$

The space of component sets to consider is quite large, but it's usually a safe strategy to start out small and search forward by adding one component at a time to the class where the improvement is the biggest.