1 Regression using CCA

Recall that performing $k$-dimensional Canonical Correlation Analysis means that given data matrices $X \in \mathbb{R}^{n \times p}$ and $Y \in \mathbb{R}^{n \times q}$, we find $k$-dimensional subspaces represented by $U \in \mathbb{R}^{p \times k}$ and $V \in \mathbb{R}^{q \times k}$ such that we maximize the correlation between the projected data points $XU$ and $YV$:

$$\rho(XU, YV) = \frac{\text{cov}(XU, YV)}{\sqrt{\text{var}(XU) \cdot \text{var}(YV)}}$$

Okay, maximizing correlation is great and all, but what’s the point? One application is to use it for regression! Recall that the correlation coefficient attains a greater value when the two sets of data are more linearly correlated. Thus, it makes sense to find the $k \times k$ weight matrix $A$ that linearly relates $XU$ and $YV$. We can do this just with OLS!

Note: to stay consistent with lecture slides, let’s now denote $X_c$ to be the projected $XU$ matrix and $Y_c = YV$. Observe that $X_c$ and $Y_c$ are zero-mean because they are linear transformations of $X$ and $Y$, which are zero-mean. Thus we can fit a linear model relating the two:

$$Y_c \approx X_cA$$

The least-squares solution is given by

$$A = (X_c^\top X_c)^{-1}X_c^\top Y_c = (U^\top X^\top XU)^{-1}U^\top X^\top YV$$

However, since what we really want is an estimate of $Y$ given new (zero-mean) observations $\tilde{X}$ (or vice-versa), it’s useful to have the entire series of transformations that relates the two. The predicted coefficients write

$$\hat{Y}_c = \tilde{X}_cA = \tilde{X}UA = \tilde{X}U(U^\top X^\top XU)^{-1}U^\top X^\top YV$$

Then we use the coefficients to compute the actual values:

$$\hat{Y} = \hat{Y}_c(V^\top V)^{-1}V^\top = \tilde{X}U(U^\top X^\top XU)^{-1}(U^\top X^\top YV)(V^\top V)^{-1}V^\top$$

We can collapse all these terms into a single matrix $A_{eq}$ that gives the prediction $\hat{Y}$ from $\tilde{X}$:

$$A_{eq} = \underbrace{U}_{\text{projection}} \underbrace{(U^\top X^\top XU)^{-1}U^\top X^\top YV}_{\text{whitening}} \underbrace{(V^\top V)^{-1}V^\top}_{\text{projection back}}$$
2 Nonlinear Least Squares

All the models we’ve seen so far are linear in the parameters we’re trying to learn. That is, our prediction \( \hat{y} = f(x; \theta) \) is some linear function of the parameters \( \theta \). For example, in OLS, \( \theta = w \) and the residuals \( r_i \) are computed by \( y_i - w^T x_i \), which is linear in the components of \( w \). In the case of least-squares polynomial regression, the predicted value is not a linear function of the input \( x \), but it is still a linear function of the parameters.

However, we may have need for models which are nonlinear function of their parameters. We consider a motivating example first.

2.1 Noisy Distance Readings

Here’s an example of a model which isn’t linear in \( \theta \): Euclidean distance.

Suppose we want to estimate the 2D position \( \theta = (\theta_1, \theta_2) \) of some entity, for example a robot. The information we have to work with are noisy distance estimates \( Y_i \in \mathbb{R} \) from \( m \) sensors whose positions \( X_i \in \mathbb{R}^2 \) are fixed and known. If we assume i.i.d. Gaussian noise as usual, our statistical model has the form

\[
Y_i = \|X_i - \theta\| + N_i, \quad N_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2), \quad i = 1, \ldots, m
\]

where

\[
\|X_i - \theta\| = \sqrt{(X_{i1} - \theta_1)^2 + (X_{i2} - \theta_2)^2}
\]

Here our prediction is

\[
\hat{y} = f(x; \theta) = \|x - \theta\|
\]

which is clearly not linear in \( \theta \).

2.2 Formulation from MLE

More generally, let us assume a model similar to the one above, but where \( f \) is now some arbitrary\(^1\) function and \( \theta \in \mathbb{R}^d \):

\[
Y_i = f(X_i; \theta) + N_i, \quad N_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2), \quad i = 1, \ldots, m
\]

Note that this implies \( Y_i \sim \mathcal{N}(f(X_i; \theta), \sigma^2) \).

\(^1\)Actually not completely arbitrary. It had better be differentiable if we want to solve using the method described later.
The maximum likelihood estimator is given by

\[ \hat{\theta}_{\text{MLE}} = \arg\max_\theta \log P(y_1, \ldots, y_m \mid x_1, \ldots, x_m; \theta, \sigma) \]

\[ = \arg\max_\theta \log \prod_{i=1}^m P(y_i \mid x_i; \theta, \sigma) \]

\[ = \arg\max_\theta \sum_{i=1}^m \log P(y_i \mid x_i; \theta, \sigma) \]

\[ = \arg\max_\theta \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(y_i - f(x_i; \theta))^2}{2\sigma^2}\right) \]

\[ = \arg\max_\theta \sum_{i=1}^m \left[-\frac{1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} (y_i - f(x_i; \theta))^2 \right] \]

\[ = \arg\min_\theta \sum_{i=1}^m (y_i - f(x_i; \theta))^2 \]

The last step holds because the first term in the sum is constant w.r.t. the optimization variable \( \theta \), and we flip from max to min because of the negative sign.

Observe that the objective function is a sum of squared residuals as we’ve seen before, even though the function \( f \) is nonlinear in general. For this reason the method is called **nonlinear least squares**. Unfortunately, there is no closed-form solution for \( \hat{\theta}_{\text{MLE}} \) in general. Later we will see an iterative method for computing it.

### 3 Solutions to Nonlinear Least Squares

Motivated by the MLE formulation above, we consider the following optimization problem:

\[ \min_\theta \varepsilon_{\text{LS}}(\theta) = \min_\theta \sum_i (y_i - f(x_i; \theta))^2 \]

One way to minimize a function is to use calculus. We know that the gradient of the objective function at any minimum must be zero, because if it isn’t, we can take a sufficiently small step in the direction of the negative gradient that the objective function’s value will be reduced.

Thus, the **first-order optimality condition** that needs to be satisfied is:

\[ \nabla_\theta \varepsilon_{\text{LS}} = 2 \sum_i (y_i - f(x_i; \theta)) \nabla_\theta f(x_i; \theta) = 0 \]

In compact matrix notation:

\[ J(\theta) \top (Y - F(\theta)) = 0 \]
where

\[
F(\theta) = \begin{bmatrix}
  f(x_1; \theta) \\
  \vdots \\
  f(x_n; \theta)
\end{bmatrix}
\]

\[
J(\theta) = \begin{bmatrix}
  \nabla_\theta f(x_1; \theta)^\top \\
  \vdots \\
  \nabla_\theta f(x_n; \theta)^\top
\end{bmatrix}
= \nabla_\theta F, \text{ the Jacobian of } F
\]

Observe that when \( f \) is linear in \( \theta \) (i.e. \( f(x_i; \theta) = \theta^\top x_i \)), the gradient \( \nabla_\theta \varepsilon_{LS} \) will only have \( \theta \) in one place because the term \( \nabla_\theta f(x_i; \theta) \) will only depend on \( x_i \):

\[
\nabla_\theta \varepsilon_{LS} = 2 \sum_i (y_i - \theta^\top x_i) \nabla_\theta (\theta^\top x_i) = 2 \sum_i (y_i - \theta^\top x_i) x_i
\]

and it is easy to derive a closed-form solution for \( \theta \) in terms of the \( y_i \)'s and \( x_i \)'s. Here’s the derivation in compact matrix notation:

\[
2X^\top (Y - X \theta) = 0 \\
2X^\top Y - 2X^\top X \theta = 0 \\
X^\top Y = X^\top X \theta \\
\theta = (X^\top X)^{-1} X^\top Y
\]

It’s just OLS!

If, however, \( f \) was not linear in \( \theta \), the term \( \nabla_\theta f(x_i; \theta) \) would contain more \( \theta \) terms (since differentiating once wouldn’t be enough to make them go away), and it would not be possible to write out a closed-form solution for \( \theta \).

**Remark:** Without more assumptions on \( f \), the NLS objective is not convex in general. This means that the first-order optimality condition is a necessary but not sufficient condition for a local minimum. That is, it is possible that the derivative is zero for some value of \( \theta \), but that value is not a local minimum. It could be a saddle point, or worse, a local maximum! Even if it is a minimum, it may not be the global minimum.

### 3.1 The Gauss-Newton algorithm

Since there is no closed-form solution to the nonlinear least squares optimization problem, we resort to an iterative algorithm, the **Gauss-Newton algorithm**\(^2\) to tackle it. At each iteration, this method linearly approximates the objective function about the current iterate and solves a least-squares problem involving the linearized objective in order to compute the next iterate.

---

\(^2\) For some reason this algorithm was called gradient descent in lecture, but it is not really gradient descent. However, like gradient descent, it is an iterative, first-order optimization algorithm. Another popular method for solving nonlinear least squares, the **Levenberg-Marquardt algorithm**, is a sort of interpolation between Gauss-Newton and gradient descent.
Let’s say that we have a “guess” for $\theta$ at iteration $k$, which we denote $\theta^{(k)}$. We can then approximate $F(\theta)$ to first order using a Taylor expansion about $\theta^{(k)}$:

$$F(\theta) \approx \tilde{F}(\theta) = F(\theta^{(k)}) + \nabla_\theta F(\theta^{(k)})(\theta - \theta^{(k)})$$

$$= F(\theta^{(k)}) + J(\theta^{(k)})\Delta \theta$$

where $\Delta \theta := \theta - \theta^{(k)}$.

Now since $\tilde{F}$ is linear in $\Delta \theta$ (the Jacobian and $F$ are just constants: functions evaluated at $\theta^{(k)}$), we can use the closed form solution for $\Delta \theta$ from the optimality condition to update our current guess $\theta^{(k)}$. Applying the first-order optimality condition from earlier to the objective $\tilde{F}$ yields the following equation:

$$0 = J_{\tilde{F}}(\theta)^\top (Y - \tilde{F}(\theta)) = J(\theta^{(k)})^\top \left(Y - \left(F(\theta^{(k)}) + J(\theta^{(k)})\Delta \theta\right)\right)$$

Note that we have used the fact that the Jacobian of the linearized function $\tilde{F}$, evaluated at any $\theta$, is precisely $J(\theta^{(k)})$. This is because $\tilde{F}$ is affine where the linear map is $J(\theta^{(k)})$, so the best linear approximation is just that.

Writing $J = J(\theta^{(k)})$ for brevity, we have

$$J^\top Y = J^\top (F(\theta^{(k)}) + J\Delta \theta)$$

$$J^\top (Y - F(\theta^{(k)})) = J^\top J(\Delta \theta)$$

$$\Delta \theta = (J^\top J)^{-1}J^\top (Y - F(\theta^{(k)}))$$

$$= (J^\top J)^{-1}J^\top \Delta Y$$

where $\Delta Y := Y - F(\theta^{(k)})$.

Recalling that $\Delta \theta = \theta - \theta^{(k)}$, we can improve upon our current guess $\theta^{(k)}$ with the update

$$\theta^{(k+1)} = \theta^{(k)} + \Delta \theta$$

$$= \theta^{(k)} + (J^\top J)^{-1}J^\top \Delta Y$$

Again, we emphasize that the $\theta^{(k+1)}$ computed by this update is the stationary point (i.e. point where the gradient vanishes) of the linearized objective $\tilde{F}$ about the current iterate $\theta^{(k)}$.

Here’s the entire process laid out in steps:

1. Initialize $\theta^{(0)}$ with some guess

2. Repeat until convergence:

   (a) Compute Jacobian with respect to the current iterate, $J = J(\theta^{(k)})$

   (b) Compute $\Delta Y = Y - F(\theta^{(k)})$

   (c) Update: $\theta^{(k+1)} = \theta^{(k)} + (J^\top J)^{-1}J^\top \Delta Y$
Note that the solution found will depend on the initial value $\theta^{(0)}$ in general.

The choice for measuring convergence is up to the practitioner. Some common choices include testing changes in the objective value:

$$\left| \frac{\epsilon^{(k+1)} - \epsilon^{(k)}}{\epsilon^{(k)}} \right| \leq \text{threshold}$$

or in the iterates themselves:

$$\max_j \left| \frac{\Delta \theta_j}{\theta_j^{(k)}} \right| \leq \text{threshold}$$