Linear Discriminant Analysis; K-Nearest Neighbors

Week	THURS. Week 3
Assignment Due	HW 1 Due — (Dimensionality Reduction)
Assignment Done	☑
Due Date	@January 27, 2023 11:59 PM
Notes Done	☑

Presentations

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/74d333a6-352a-4367-89b8-49814f79eb03/L2-LDA.pdf

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/d485e77a-020e-459d-903a-f6021236f59a/L3-KNN.pdf

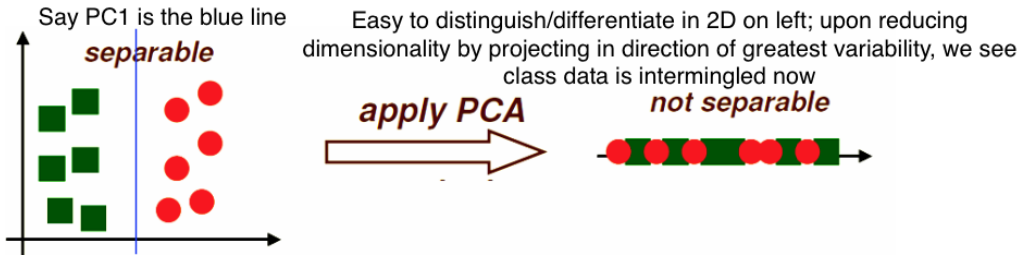
Class Notes

We'll start getting into first two ML algorithms regarding classification:

- We'll be covering **LDA** first, as its closely related to PCA
  - Simple, but powerful algorithm
- In general, with supervised learning, we're trying to find a model that does computations, such that it gets to our target value.

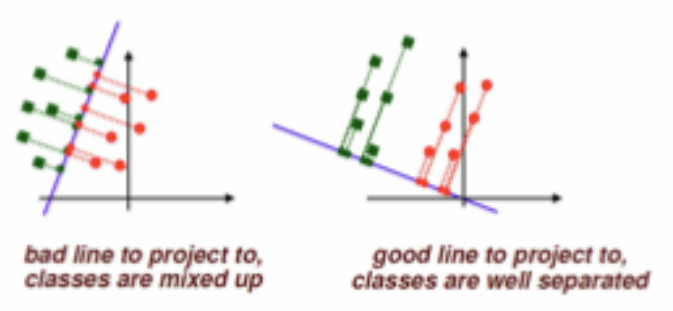
- Hypothesis: Given infinite pure data about an observation, there is an underlying function $y = f(\mathbf{z})$ that can do perfect matching.
    - However, in reality, we only have a subset of information and noise embedded in our captured observations → $X \subset Z + \epsilon$
    - We want to find a function that when we give it our limited data, it should do reasonably well at approximating the true ideal underlying function → $g(\mathbf{x}) \approx f(\mathbf{z})$

- We'll look at **linear discriminate analysis (LDA)**:
    - It was NOT designed for classification, but it is meant to address the shortcomings of PCA on data that will ultimately be used for classification.
        - Recall: PCA is looking at a data matrix X and is trying to find the direction of greatest variablilty s.t. when we project in this direction, our data has maximum variance.
            - All the PCA does is use the data matrix to find ***maximum variability*** → it doesn't concern itself with class labels!
                - Ex: In reducing our dimensionality, class data can become intermingled, making a classification ploblem more difficult.

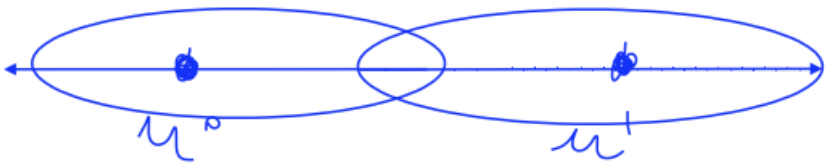


- Like PCA, **linear discriminant analysis** wants to look for an axis to project onto!
    - **However, it will formulate the problem differently than PCA** → w/ PCA, we want to maximize variance, post-projection
    - We'll define a function $g(x)$, which creates a projection → $g(\mathbf{x}) = \mathbf{xw}$
        - Unlike PCA, we want this projection to result in locations that are useful for binary classification, which PCA didn't concern itself with.

- We will find a projection to a line s.t. samples from different classes are well separated.



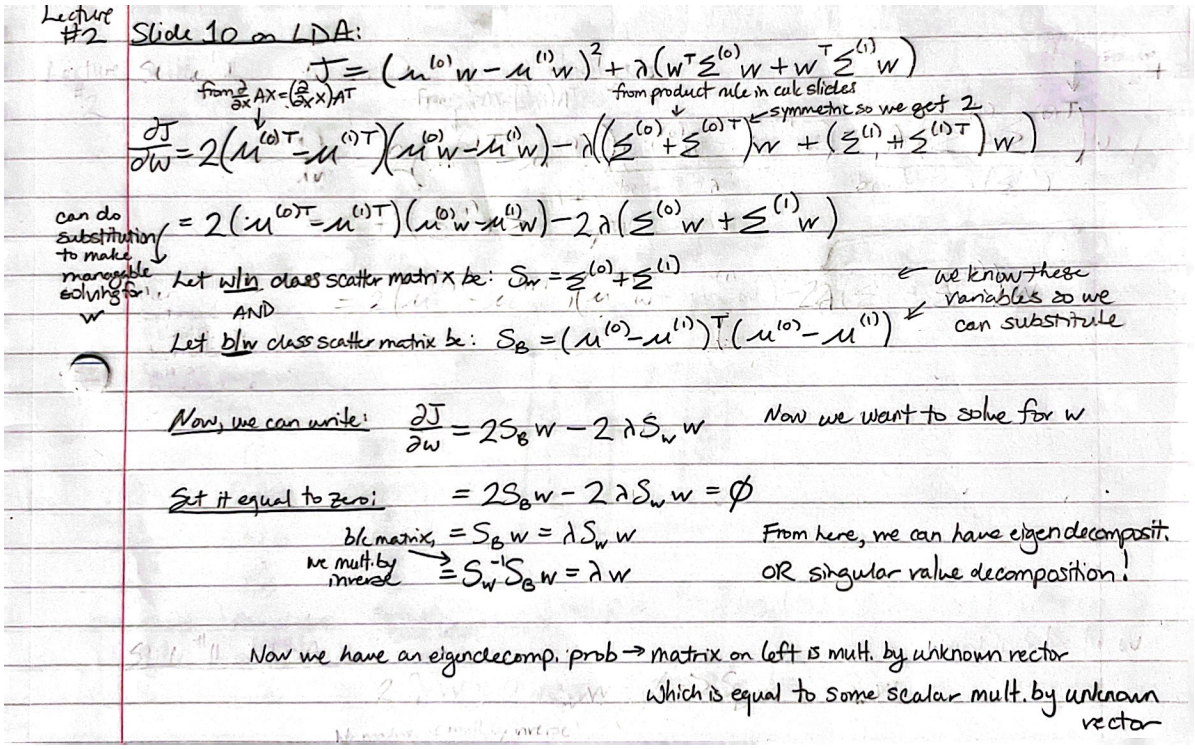
- <u>Say:</u> Data is in 2D space, where we have D features for each observation with two classes (meaning it is binary classification data
  - C_0 is the subset of sample from class 0
  - C_1 is the subset of samples from class 1
  - Again, we want projection matrix W that provides good class separation after projection
    - Let $\mu^{(0)}$ and $\mu^{(1)}$ be our mean features **before** projection.
      - To evaluate how good class separation is after separation, we can take our mean vector for class 0 $\mu^{(0)}$ and project into $\mathbf{w}$ and take mean vector for class 1 $\mu^{(1)}$ and project it onto $\mathbf{w}$.
        - We want locations of projected means to be far apart s.t. want $\mathbf{w}$ that does this
        - We want to say after projection that our means are nicely separated
      - **<u>Problem with formulation above:</u>** Just by having data means projected far from each other doesn't mean that there'll be great class separation, there can still exist a lot of class variance!



        - We don't only want means to be far away, bu we'd like variances for our classes' data to be small!
    - Let $\Sigma^{(0)}$ and $\Sigma^{(1)}$ be the covariance of the features of the classes 0 and 1 respectively **before** projection.
        - We can then add a term to our objective function penalizing a large *within-class* covariance in the covariance matrices after projection while maximizing means after projection:

$$J = \left(\boldsymbol{\mu}^{(0)}\boldsymbol{w} - \boldsymbol{\mu}^{(1)}\boldsymbol{w}\right)^2 - \lambda\left(\boldsymbol{w}^T\Sigma^{(0)}\boldsymbol{w} + \boldsymbol{w}^T\Sigma^{(1)}\boldsymbol{w}\right)$$

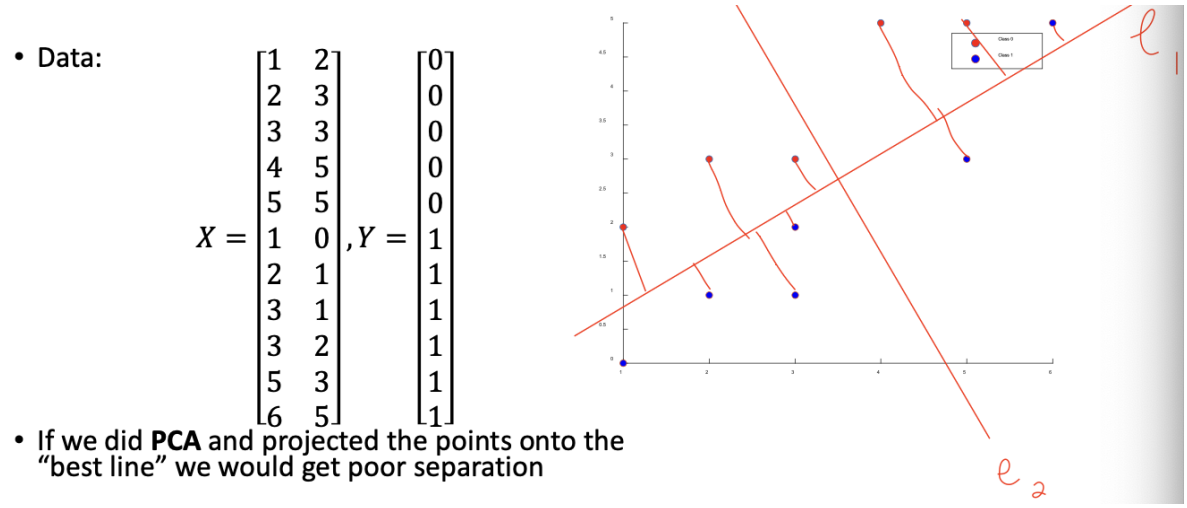
        - The negative sign allows us to maximize the negation of this term, effectively penalizing for this variation.
    - λ is effectively a weight, telling us how much we care about the second term.
        - **The larger the lambda, the more we care about the term after it**
        - Note: This isn't a value that we really set, as it is instead the eigenvalue of the eigenvector that we project onto.
            - Larger eigenvalue = variance must be smaller
    - Again, overall we want to find values of $\mathbf{w}$ that maximize this overall!
- **Notes on math on separate page:**



- To be noted above in why we use the term $\mathbf{w}^T\Sigma^{(0)}\mathbf{w}$:
  - Ultimately, for the objective function, we want our output to be a single value.
    - $\mathbf{w}^T$ is a $1 \times D$ sized matrix, while Σ is a $D \times D$ sized matrix, while $\mathbf{w}$ is $D \times 1$, ultimately leaving us a single value when they're multiplied together.
    - If we only tried to do $\Sigma \times \mathbf{w}$, then we'd be left with a $D \times 1$ matrix that effectively represents the sum of the product done on this matrix multiplication.
      - By taking $\mathbf{w}^T$ and multiplying by this sum of the product, we are getting a singular value, effectively taking a weighted sum of these values altogether.
- In simplying our derivation, we can see that we can solve our problem with eigen-decomposition or singular value decomposition as in PCA.
  - In this problem, however, we will now decompose the matrix $S_W^{-1}S_B$ to find eigenvectors that are possible solutions for $\mathbf{w}$.

    - Since this problem is effectively a **maximization problem**, the best solution here will be the one that gives us a value closest to 0 from an eigenvector associated with the smallest eigenvalue.

- Note: If we have two classes (**binary classification**), then $S_W^{-1}S_B$ has only one non-zero eigenvalue (and therefore one eigenvector). The rest of the eigenvalues/eigenvectors will have a value of 0.
    - This eigenvector will be the axis that we project onto that will give us the desired result of maximizing the mean of the classes post-projection while minimzing covariances post-projection.
- In an example problem we can see that if we did PCA and projected the points onto the "best line", we would get poor separation when reducing dimensionality.



    - We can see that when projecting on e_1 to 1D space that we would get intermingling here of the classes.

Steps for LDA

For **LDA**, we know that we'd need the ***between class scatter matrix*** and the ***within class scatter matrix***.

1. We begin by computing the means and covariance matrix for each class:

$$\boldsymbol{\mu}^{(0)} = [3, 3.6],\ \boldsymbol{\mu}^{(1)} = [3.33, 2]$$

$$\Sigma^{(0)} = \begin{bmatrix} 2.5 & 2 \\ 2 & 1.8 \end{bmatrix}, \Sigma^{(1)} = \begin{bmatrix} 3.47 & 3.2 \\ 3.2 & 3.2 \end{bmatrix}$$

2. We can now compute the ***within*** and ***between class scatter matrices***:

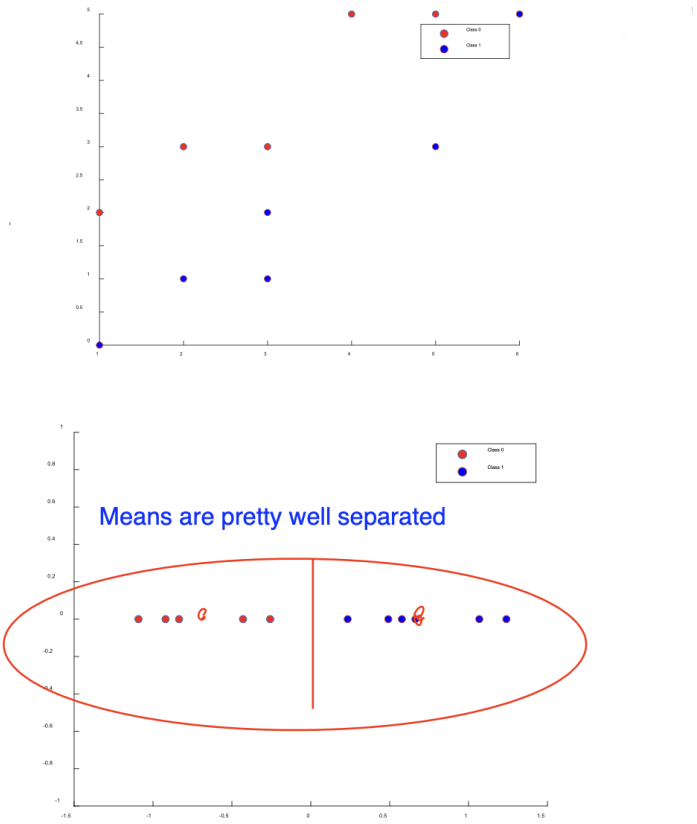
$$S_b = \left(\boldsymbol{\mu}^{(0)} - \boldsymbol{\mu}^{(1)}\right)^T \left(\boldsymbol{\mu}^{(0)} - \boldsymbol{\mu}^{(1)}\right) \approx \begin{bmatrix} 0.11 & -0.53 \\ -0.53 & 2.56 \end{bmatrix}$$

$$S_W = \Sigma^{(0)} + \Sigma^{(1)} = \begin{bmatrix} 5.97 & 5.2 \\ 5.2 & 5 \end{bmatrix}$$

3. Perform eigen-decomposition on $S_W^{-1} S_B$ matrix:
    a. The eigenvector pertaining to the only non-zero eigenvalue is our projection matrix:
        i. There is only one non-zero eigenvalue, so its corresponding eigenvector becomes our direction of projection.

$$\boldsymbol{w} = \begin{bmatrix} 0.66 \\ -0.75 \end{bmatrix}$$

4. Now, we can simply project each point onto the line(s):
    a. $Z = XW$



Variance is relatively small s.t. we don't have overlap between classes

b. For **binary classification**, we can then just impost a threshold c s.t.:

$$\hat{y} = \begin{cases} 0, & z < c \\ 1, & otherwise \end{cases}$$

- For example, in this case, we can say if anything projects to a positive value that it is the blue class, while if anything projects to a negative value then we'd assign it to the red class.

Again, **LDA** wasn't necessarily defined as a classification problem, rather a way to reduce dimensionality while taking into account class separation into account.

- It was found that if we're considering a binary classification problem, LDA *can be used* as a technique to do binary classification!

Multi-Class Classification

If we had more than two classes, we'd still need the $S_W^{-1}S_B$ matrices to decompose and get our projection matrix.

- For our ***within class scatter matrix***, we would add all of the classes' covariance matrices.

$$S_W = \sum_{i=1}^{K} \Sigma^{(i)}$$

- The ***between class scatter matrix*** is trickier:

$$S_B = \sum_{i=1}^{K} |C_i| \left(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}\right)^T \left(\boldsymbol{\mu}^{(i)} - \boldsymbol{\mu}\right)$$

  - Where $\boldsymbol{\mu}$ is the mean of **all** data

    - We'd take the *mean of all of the data,* μ, iterate through all the classes, computing the difference of each classes' means with the entire data transposed, multiplied by the difference of that classes' mean with mean of the entire data.

      - This entire calculation would then be weighted by the number of elements (observations) in class i, giving it a kind of weighted sum of the difference between each classes' means and the overall mean.

- In decomposing $S_W^{-1}S_B$ to get the eigenvectors and eigenvalues, we'll find there are $K-1$ eigenvalues that are non-zero, where K is the number of classes that we have.

- To do classification, we'd have to check the distance between the projected location in $K-1$ dimensional space and that of its' classes mean, seeing which mean that the observation is closest to decide on which class it is a part of.

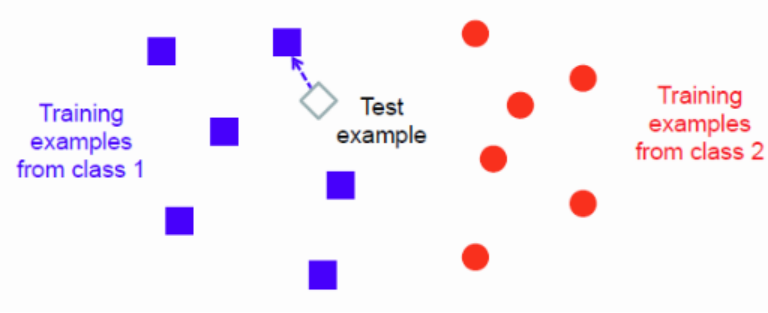
    - Ex: If we had 4 classes, regardless of number of features we'd go down to 3D space.

- Overall, this tells us that we can still use this model for multi-class classification as well.

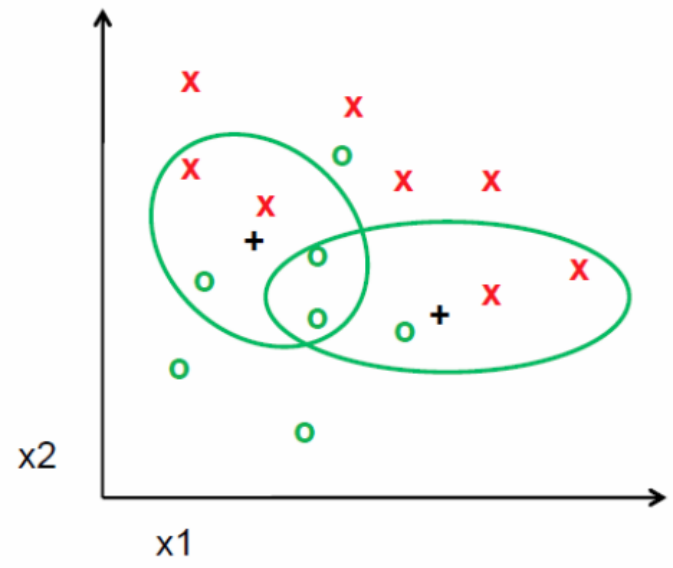
K-Nearest Neighbors

The second machine learning algorithm for classification that we'll be covering is **K-nearest neighbors**:

- Idea: Say we have a bunch of training data with a class label associated with it
  - Simplest thing we can do when getting new observations, possible some validation observations, we find out which of the training samples is it the closest to (**distance measure**) or more similar to (**similarity measure**).
    - It may be easier to visualize a distance measure.
      - We may be able to calculate the distance with all other training data samples and decide based on distance, classify the new point.



  - No training required!!
    - With **LDA**, you can call training the finding of eigenvector(s).
    - If you have a lot of training data, computing all distances can be relatively computationally expensive!
- However, solely using the ONE nearest neighbor is susceptible to noise or outliers.
  - Ex: If we look at 5 nearest neighbors rather than solely one, one observation may be classified differently than if we only considered one.



- There is no training time involved → Other systems we'll look into can take months or years to train!
    - There IS cost in evaluating since we have to keep training data in memory and compute all of the distances against each training sample → actual evaluation time can lead to being computation can be expensive in space and time
- Question: Is there a rule of thumb about how big/small k should be?
    - If we're doing binary classification, it's good for k to be odd so mode doesn't end up being a tie.
    - In practice, this could be where we use a validation set, helping us decide between different systems' parameters.
        - Maybe we start with $k = 1$, seeing how it does on the validation set, then increasing k incrementally based on how we perform on the validation set.
            - The smaller k is, the more "local" your decision is made on, losing globalization.
            - The larger k is, the larger the area you'll be looking at, losing localization.
                - It's effectively a trade-off → could try weighing neighbors in having more local neighbors have a higher weight of impact on your decision made.

- Question: When is it appropriate to z-score our data?
    - Since we're computing the distance between observations for KNN, solely because a feature is on a different scale than another larger scale, it doesn't mean that this feature should contibute more to the distance.
        - As a result, for KNN, it does logically make sense to z-score data so all features can contribute equally to the distance computation regardless of their intrinsic scale.
    - Note: We don't always want to z-score our data! It depends on what we're doing with the data in a particular algorithm.

Homework #2

We'll be implementing a **k-nearest neighbors classifier**.

Resources

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c6e0e3b7-51b0-44b2-a850-14ec67f5e7e6/data-reduction-linear-discriminant-analysis.pdf

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/3d051e04-9603-40b1-9a40-20ed5fda3092/non-linear-dimensionality-reduction.pdf