The Mathematics of Dimensionality Reduction

Divy Murli

Kount, Inc

1 Introduction
- Overview
- Motivation

2 Singular Value Decomposition
- Mathematics
- Application: Image compression

3 Principal Component Analysis
- Some statistics
- Application: clustering

4 Linear Discriminant Analysis
- Setting up Fisher's problem
- Application: LDA vs PCA head-to-head

5 Concluding comments

Background

- BS, PhD in theoretical physics (dissertation in string theory)
- Data science/ML practictioner, started to transition mid-way through grad school via coursework/projects
- Unlike many other subjects, data science lies at the confluence of many disciplines, including but not limited to: computer science, statistics, applied maths, EE/signal processing, ...

Convention

$$\text{data} = \text{samples}\left\{\overbrace{\begin{matrix} \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \end{matrix}}^{\text{features}}\right.$$

Why dimensionality reduction?

- Method of easily exploring high-dimensional datasets
- Distill a dataset to its *essence*, in particular by choosing salient or 'eigen' features (more on this later)
- Project a high-dimensional dataset down to low-dimensions for easy interpretability and visualisation
- Uncover the structure of a dataset in a (perhaps) unsupervised manner, potentially for classification as well
- Uncover features that account for the most variance in the data

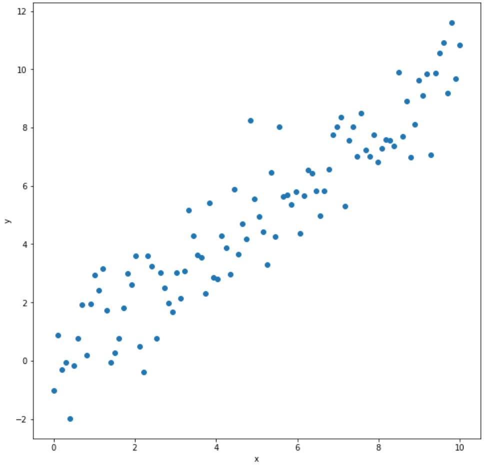
Methods of Dimensionality Reduction/Feature Reduction

- Remove features by hand in an ad-hoc manner, and check effect of performance on a a model of interest (e.g. neural network, logistic regression, random forest, etc). Remove features that don't affect model performance
- Feed through a random forest, select important features through e.g. Gini purity/information gain

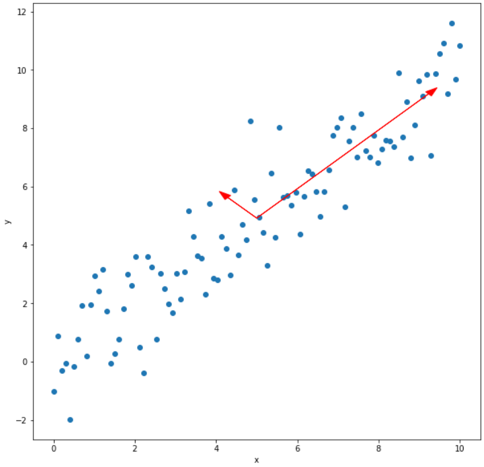
Methods of Dimensionality Reduction/Feature Reduction

- Remove features by hand in an ad-hoc manner, and check effect of performance on a a model of interest (e.g. neural network, logistic regression, random forest, etc). Remove features that don't affect model performance
- Feed through a random forest, select important features through e.g. Gini purity/information gain
- Select salient 'features' as directions of maximal *variance*: Principal Component Analysis (unsupervised)
- Select salient 'features' that maximally separate classes: Linear Discriminant Analysis (supervised)

Let's look at an example of correlated data:



As you'd intuitively expect, there's a strong positive correlation between these two variables, and and you can probably just draw on the most important features:



But, what if our dataset has 100, or maybe even 1,000 features? There's no way we can possibly visualise that. And we *definitely* can't just draw arrows like we did above.

But, what if our dataset has 100, or maybe even 1,000 features? There's no way we can possibly visualise that. And we *definitely* can't just draw arrows like we did above.

Enter PCA.

Before we get there though ...

Singular Value Decomposition (SVD)

$$X = U\Sigma V^T$$

Introducing SVD

(Most) square matrices can be eigendecomposed, namely given a square matrix $A \in \mathbb{R}^{n \times n}$, we can write it as

$$A = PDP^{-1}$$

where $P = [\vec{v}_1|...|\vec{v}_n]$ and $D = \mathrm{diag}(\lambda_1, ..., \lambda_n)$.

Introducing SVD

(Most) square matrices can be eigendecomposed, namely given a square matrix $A \in \mathbb{R}^{n \times n}$, we can write it as

$$A = PDP^{-1}$$

where $P = [\vec{v}_1|...|\vec{v}_n]$ and $D = \mathrm{diag}(\lambda_1, ..., \lambda_n)$.
But most matrices aren't square. And, practically all *data* matrices aren't square.

Introducing SVD

To generalise, it turns out that given any rectangular matrix $X \in \mathbb{R}^{m \times n}$,

$$X = U \Sigma V^T$$

where U and V are respectively $m \times m$ and $n \times n$ orthogonal matrices whose columns are the left and right singular vectors of X. These are generalisations of the eigenvectors of a square matrix.

What is Σ?

The analogy of the eigenvalues of a square matrix, Σ is a rectangular $m \times n$ matrix whose diagonal entries are the *singular* values. If $m > n$ (as is the case in most datasets; we typically have more samples than features), it can be written as

$$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & & \\ & & \end{bmatrix}, \quad \sigma_1 \geq \ldots \geq \sigma_n$$

What is Σ?

The analogy of the eigenvalues of a square matrix, Σ is a rectangular $m \times n$ matrix whose diagonal entries are the *singular* values. If $m > n$ (as is the case in most datasets; we typically have more samples than features), it can be written as

$$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & & \\ & & \end{bmatrix}, \quad \sigma_1 \geq \ldots \geq \sigma_n$$

If X has column rank $r < n$, then only the first r singular values of X will be nonzero.

What is Σ?

The analogy of the eigenvalues of a square matrix, Σ is a rectangular $m \times n$ matrix whose diagonal entries are the *singular* values. If $m > n$ (as is the case in most datasets; we typically have more samples than features), it can be written as

$$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & & \\ & & \end{bmatrix}, \quad \sigma_1 \geq \ldots \geq \sigma_n$$

If X has column rank $r < n$, then only the first r singular values of X will be nonzero. The same is true for eigendecomposition of a square matrix.

Computing Singular Values

Let u_i, v_i respectively be the rows and columns of U and V. The SVD can equivalently be written as

$$X = u_1\sigma_1 v_1^T + ... + u_r\sigma_r v_r^T$$

Computing Singular Values

Let u_i, v_i respectively be the rows and columns of U and V. The SVD can equivalently be written as

$$X = u_1\sigma_1 v_1^T + ... + u_r\sigma_r v_r^T$$

$$XX^T = U\Sigma\Sigma^T U^T \in \mathbb{R}^{m\times m}$$
$$X^T X = V\Sigma^T\Sigma V^T \in \mathbb{R}^{n\times n}$$

Computing Singular Values

Let u_i, v_i respectively be the rows and columns of U and V. The SVD can equivalently be written as

$$X = u_1\sigma_1 v_1^T + ... + u_r\sigma_r v_r^T$$

$$XX^T = U\Sigma\Sigma^T U^T \in \mathbb{R}^{m\times m}$$
$$X^TX = V\Sigma^T\Sigma V^T \in \mathbb{R}^{n\times n}$$

From above, columns of U are the eigenvectors of XX^T and columns of V are eigenvectors of X^TX. $\sigma_1^2, ..., \sigma_n^2$ are eigenvalues of both X^TX and XX^T. Assuming $m > n$, XX^T will have $m - n$ additional zero eigenvalues.

SVD for approximation

- Often, storing a large matrix can be infeasible. How do we approximate it?

SVD for approximation

- Often, storing a large matrix can be infeasible. How do we approximate it?
- More specifically, how do we give a rank k approximation?

SVD for approximation

- Often, storing a large matrix can be infeasible. How do we approximate it?
- More specifically, how do we give a rank k approximation?
- Even more specifically, given a matrix $X \in \mathbb{R}^{m \times n}$ with rank $r \leq n$, what's the 'best' rank k matrix B that approximates X for some some $k < r$?

SVD for approximation

To answer the question of ‘best’, we need a notion of distance. That is, given B, we want to answer the optimisation question

$$\min_B \|X - B\|$$

where the minimum is taken over all rank k matrices B.

SVD for approximation

To answer the question of 'best', we need a notion of distance. That is, given B, we want to answer the optimisation question

$$\min_{B} \|X - B\|$$

where the minimum is taken over all rank k matrices B. Here are a couple of common choices of 'distance' one can use:

$$\|B\|_2 = \max_{x \in \mathbb{R}^n} \frac{\|Bx\|}{\|x\|} = \sigma_1 \text{ (spectral norm)}$$
$$\|B\|_F = \sqrt{\sigma_1^2 + \ldots + \sigma_k^2} \text{ (Frobenius norm)}$$

Eckart-Young-Mirsky

Eckart-Young-Mirsky give a solution to the optimisation problem on the previous slide:

$$\|X - B\| \geq \|X - A_k\|$$

where

$$A_k = \sigma_1 u_1 v_1^T + ... + \sigma_k u_k v_k^T$$

is the optimal solution.

Image compression

- Given a generic (black and white) image represented as an $m \times n$ matrix, one would need to store mn values.
- However, by approximating this image with a rank k matrix, one would need to store $k(m+n)$ values.

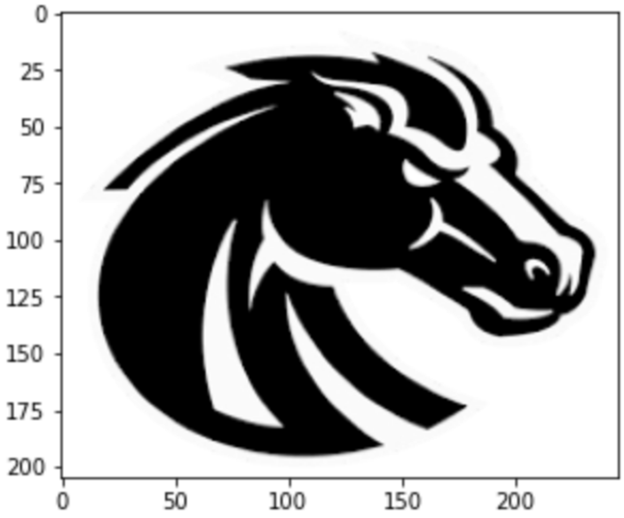
Image compression

- Given a generic (black and white) image represented as an $m \times n$ matrix, one would need to store mn values.
- However, by approximating this image with a rank k matrix, one would need to store $k(m+n)$ values.

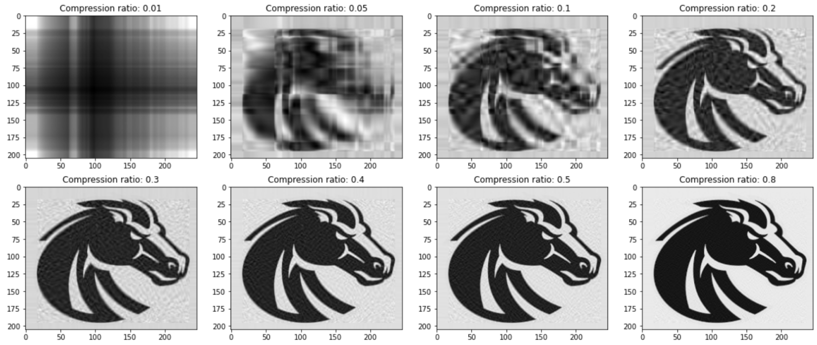
Define the *compression ratio* to be

$$\frac{k(m+n)}{mn}$$

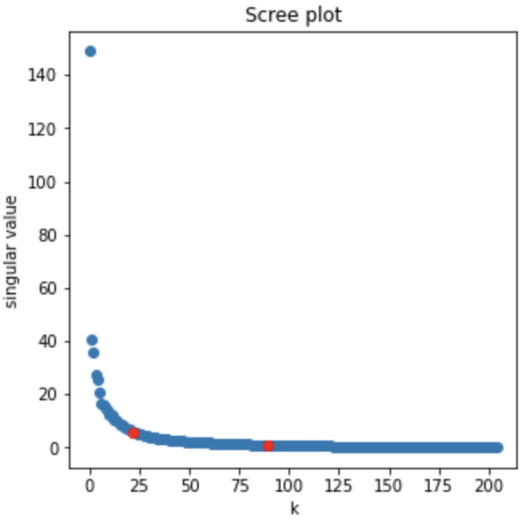
Reconstructing an image with SVD



Reconstructing an image with SVD



Scree plot


Scree plot
singular value
140
120
100
80
60
40
20
0
0
25
50
75
100
125
150
175
200
k

- Considering the image like a ‘dataset’, this suggests that we can ‘reconstruct’ the image from building blocks of rank one matrices.
- This is exactly how PCA works: capture the *salient* information in a dataset by projecting it onto the most important components.
- Said differently, PCA is a *statistical interpretation* of the SVD. We’ll see this below.

Introducing PCA

Given tabular data

$$X_0 = \rotatebox{90}{samples}\left\{ \overbrace{\begin{matrix} \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \\ \bullet & \bullet & \cdots & \cdots & \bullet \end{matrix}}^{\text{features}} \right.$$

Sample mean and covariance

Given data samples $x_{1k}, ..., x_{mk}$ (rows of X_0, $1 \leq k \leq n$ indexing features), we can define the *sample mean* as

$$\mu_k = \frac{1}{m} \sum_{i=1}^{m} x_{ik}$$

Given a set of observations $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}$ (rows of X_0) the *sample covariance* is

$$Q_{ab} = \frac{1}{m-1} \sum_{k=1}^{m} (x_{ka} - \mu_a)(x_{kb} - \mu_b)$$

Sample mean and covariance

Given data samples $x_{1k}, \ldots, x_{mk}$ (rows of X_0, $1 \leq k \leq n$ indexing features), we can define the *sample mean* as

$$\mu_k = \frac{1}{m} \sum_{i=1}^{m} x_{ik}$$

Given a set of observations $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ (rows of X_0) the *sample covariance* is

$$Q_{ab} = \frac{1}{m-1} \sum_{k=1}^{m} (x_{ka} - \mu_a)(x_{kb} - \mu_b)$$

with $1 \leq a, b \leq n$. (Recall $\mathrm{Cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))]$.)

PCA as a statistical interpretation of SVD

Define X to be the mean-subtracted matrix. Then, the covariance matrix can be written as

$$Q \propto X^T X$$

PCA as a statistical interpretation of SVD

Define X to be the mean-subtracted matrix. Then, the covariance matrix can be written as

$$Q \propto X^T X$$

But this is what we saw earlier! (remember the blue equation). In fact, in

$$X^T X = V \Sigma^T \Sigma V^T$$

$\Sigma^T \Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_n)$ and V is diagonalises $X^T X$.

PCA as a statistical interpretation of SVD

Define X to be the mean-subtracted matrix. Then, the covariance matrix can be written as

$$Q \propto X^T X$$

But this is what we saw earlier! (remember the blue equation). In fact, in

$$X^T X = V \Sigma^T \Sigma V^T$$

$\Sigma^T \Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_n)$ and V is diagonalises $X^T X$. The right singular vectors of X (or the eigenvectors of $X^T X$) are the directions of variance of the data, the top k directions of largest variance correspond to the top k eigenvectors (arranged by eigenvalue).

PCA as a statistical interpretation of SVD

Define X to be the mean-subtracted matrix. Then, the covariance matrix can be written as

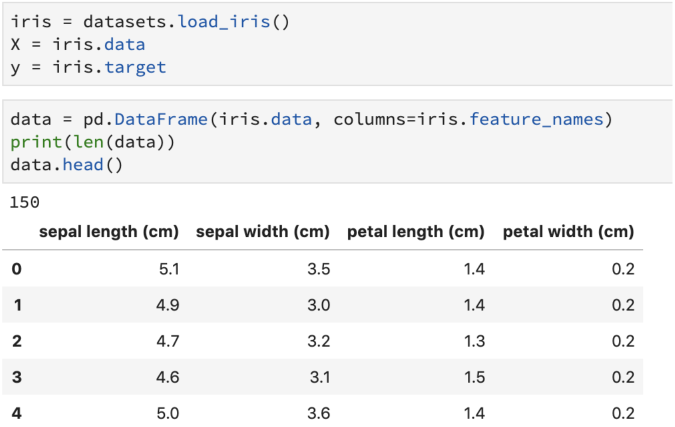
$$Q \propto X^T X$$

But this is what we saw earlier! (remember the blue equation). In fact, in

$$X^T X = V \Sigma^T \Sigma V^T$$

$\Sigma^T \Sigma = \mathrm{diag}(\sigma_1, ..., \sigma_n)$ and V is diagonalises $X^T X$. The right singular vectors of X (or the eigenvectors of $X^T X$) are the directions of variance of the data, the top k directions of largest variance correspond to the top k eigenvectors (arranged by eigenvalue). The singular values of X are the variances themselves.

Example with iris data



```
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```
data = pd.DataFrame(iris.data, columns=iris.feature_names)
print(len(data))
data.head()
```

```
150
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Example with iris data

```
X_recentered = X - feature_means
```

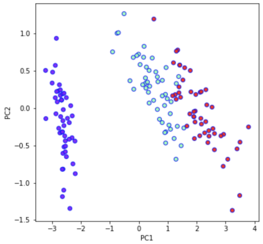
```
#Compute SVD
U, S, Vt = np.linalg.svd(X_recentered)
print(U.shape)
print(S.shape)
print(Vt.shape)
```

```
(150, 150)
(4,)
(4, 4)
```

```
V = np.transpose(Vt)
PC1 = np.dot(X_recentered, V[:, 0])
PC2 = np.dot(X_recentered, V[:, 1])

fig = plt.figure(figsize=(6,6), facecolor="white")
ax=fig.add_subplot(111)
ax.scatter(PC1, PC2, c=y, cmap="rainbow", alpha=0.9, edgecolor="b")
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
```

```
Text(0,0.5,'PC2')
```



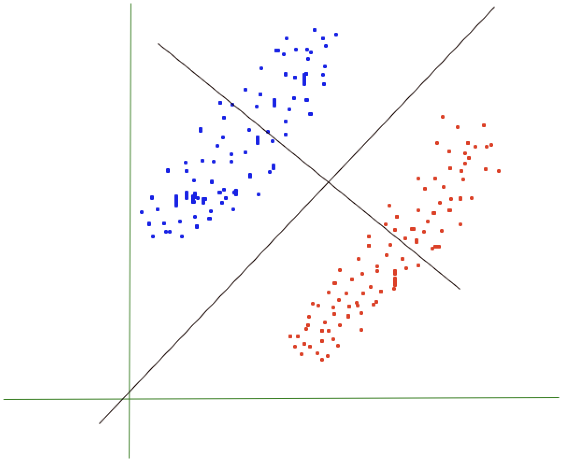
Linear discriminant analysis

- PCA is an excellent exploratory tool to understand the structure of a dataset, in an *unsupervised* manner

Linear discriminant analysis

- PCA is an excellent exploratory tool to understand the structure of a dataset, in an *unsupervised* manner
- Suppose we also had access to class labels – how can we utilise them in dimensionality reduction?
- Linear discriminant analysis (LDA) provides a way to reduce dimensions whilst maximising the separation between classes
- LDA is used both for classification as well as dimensionality reduction

We'll be focussing on Fisher's setup for LDA.



Fisher's problem

Suppose we have two classes, with data points $x_1, ..., x_{n_1} \in \mathbb{R}^n$ and $y_1, ..., y_{n_2} \in \mathbb{R}^n$. Define The respective means are

$$\mu_1 = \frac{1}{n_1} \sum_{j=1}^{n_1} x_i$$

respectively for μ_2.

Fisher's problem

Suppose we have two classes, with data points $x_1, ..., x_{n_1} \in \mathbb{R}^n$ and $y_1, ..., y_{n_2} \in \mathbb{R}^n$. Define The respective means are

$$\mu_1 = \frac{1}{n_1} \sum_{j=1}^{n_1} x_i$$

respectively for μ_2. Let $v \in \mathbb{R}^n$ be the vector that optimally separates the classes upon projection. Define the projected means to be

$$\tilde{\mu}_1 = \frac{1}{n_1} \sum_{j=1}^{n_1} v^T x_i$$

respectively for $\tilde{\mu}_2$.

Fisher's problem

Define the *within class* scatter matrices to be

$$\begin{aligned}\tilde{s}_1^2 &= \sum_{j=1}^{n_1} (v^T x_j - \tilde{\mu}_1)^2 \\ &= v^T \underbrace{\left(\sum_{j=1}^{n_1} (x_j - \mu_1)(x_j - \mu_1)^T \right)}_{s_1} v\end{aligned}$$

and respectively $\tilde{s}_2^2$.

Fisher's problem

Define the *within class* scatter matrices to be

$$\begin{aligned}\tilde{s}_1^2 &= \sum_{j=1}^{n_1} (v^T x_j - \tilde{\mu}_1)^2 \\ &= v^T \underbrace{\left(\sum_{j=1}^{n_1} (x_j - \mu_1)(x_j - \mu_1)^T \right)}_{s_1} v\end{aligned}$$

and respectively $\tilde{s}_2^2$. Seek to maximise

$$R(v) \equiv \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

wrt v.

Fisher's problem

Define the *within class* scatter matrices to be

$$\tilde{s}_1^2 = \sum_{j=1}^{n_1} (v^T x_j - \tilde{\mu}_1)^2$$
$$= v^T \underbrace{\left(\sum_{j=1}^{n_1} (x_j - \mu_1)(x_j - \mu_1)^T \right) v}_{s_1}$$

and respectively $\tilde{s}_2^2$. Seek to maximise

$$R(v) \equiv \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

wrt v. that is, maximise the distance the means of classes are away from one another relative to the spread of each class.

Fisher's problem

Write the ratio as

$$R(v) = \frac{v^T S_b v}{v^T S_w v}$$

where

$$\begin{aligned} S_b &= (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \\ S_w &= s_1 + s_2 \end{aligned}$$

are respectively the *between* and *within* class scatter matrices.

Fisher's problem

Write the ratio as

$$R(v) = \frac{v^T S_b v}{v^T S_w v}$$

where

$$\begin{aligned} S_b &= (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \\ S_w &= s_1 + s_2 \end{aligned}$$

are respectively the *between* and *within* class scatter matrices. Ratio is maximised at $v \propto S_w^{-1}(\mu_1 - \mu_2)$ (cf generalised Rayleigh quotient).

Multiclass case

We must maximise the same ratio (for k classes)

$$R(v) = \frac{v^T S_b v}{v^T S_w v}$$

where

$$S_b = n_1(\mu_1 - \mu)(\mu_1 - \mu)^T + ... + n_k(\mu_k - \mu)(\mu_k - \mu)^T$$
$$S_w = s_1 + s_2 + ... + s_k$$

μ_i, n_i, μ are respectively class means, class numbers and total mean.

Multiclass case

We must maximise the same ratio (for k classes)

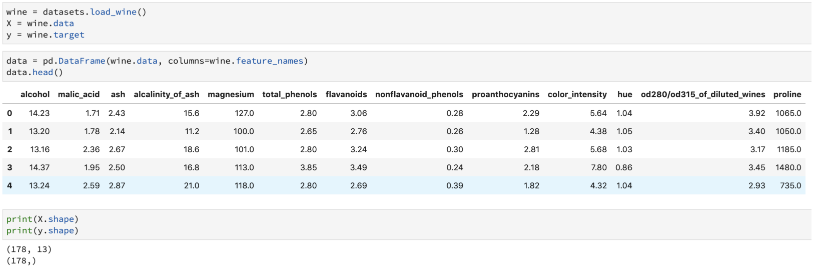
$$R(v) = \frac{v^T S_b v}{v^T S_w v}$$

where

$$\begin{aligned} S_b &= n_1(\mu_1 - \mu)(\mu_1 - \mu)^T + \ldots + n_k(\mu_k - \mu)(\mu_k - \mu)^T \\ S_w &= s_1 + s_2 + \ldots + s_k \end{aligned}$$

μ_i, n_i, μ are respectively class means, class numbers and total mean. (Generally, for k classes we must find the top $k-1$ generalised eigenvectors of $S_b v = \lambda S_w v$. In fact, the rank of S_b turns out to be $k-1$, generalising from the one-class case.)

PCA vs LDA



```
wine = datasets.load_wine()
X = wine.data
y = wine.target
```

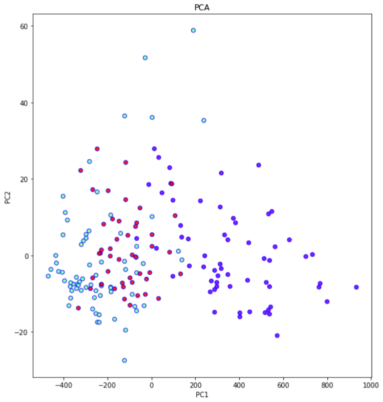
```
data = pd.DataFrame(wine.data, columns=wine.feature_names)
data.head()
```

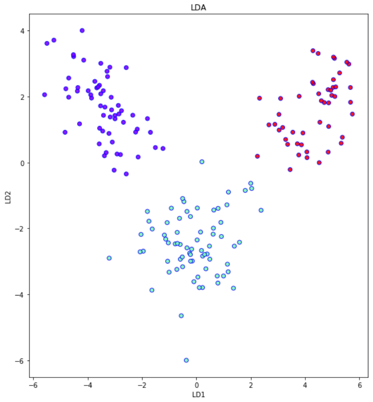
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

```
print(X.shape)
print(y.shape)
```

```
(178, 13)
(178,)
```

PCA vs LDA





More info at: https://towardsdatascience.com/linear-discriminant-analysis-in-python-76b8b17817c2

Tips for preparation

Here's a (non-exhaustive) list of methods and tools you should probably become familiar with ...

Maths

- linear algebra
- statistics
- vector calculus (for neural networks/gradient descent)

Programming

- Stuff is mostly done in python (some people also like R, though I find it to be less versatile)
- Bread and butter stuff like numpy, scipy, pandas, sklearn to tinker around with smallish datasets (like in this presentation)
- Spark/hadoop, SQL and cloud services (AWS/GCP/Azure) for big data analytics – this one is huge. I use spark pretty extensively for my job
- Tensorflow/pytorch to play around with neural networks

Reach out via email/LinkedIn if you have any further questions. Thanks!