# Preface

I plan to explore quite a lot in this book under the pretext of, and largely guided by ideas of Machine Learning. Considering the lack of hyper linking, image insertions or rearrangability in this medieval instrument known as a book, I will likely scan & reorder or add sections into these notes along with an index. Hope to have fun!

# Prologue

After a great deal of calculus, linear algebra & algorithms built with them, I've come to wonder, What is ML? The word suggests that something is "learnt" which naturally means that there is something to learn from, & that is data. Data, often, large volumes of it, is characteristic of ML algorithms.

The YouTube algorithm, Weather prediction, Stock & other market predictions, Speech to text, conversational AI, Computer Vision & Object recognition (not like the algorithmic OpenCV but like YOLOv5) and maybe one day, a sentient AI, are all things we strive to achieve with ML, because fundamentally, we have absolutely no idea how things work under the hood, so we hope & pray that with enough data, a machine figures out a pattern. This is what ML is for :- it's to make a machine construct a practically useful, yet imperfect solution to a problem that is beyond our present capabilities & understanding or is simply unsolvable.

Sure, house prices don't follow a 5th order polynomial in 20 variables, but if works approximately & the level of error is acceptable, that's a good enough answer, despite the fact that it lacks rhyme, reason or cleverness. It's an answer found by sheer force, that happens to generalise well.

With that out of the way, I'd like to point out that there are 3 broad types of ML: The first, is supervised learning, where you have labelled data telling the computer what is right & wrong.

For instance, a classification problem of identifying if an email is spam or not is supervised. We have a large set of labelled data & we try to find trends in it. A regression problem, where we fit equations to data like house prices is also supervised with past data.

The second type is unsupervised learning, where nobody knows the right answer. For example, you may want to classify all tweets about Elon musk into some number of groups. You don't know what groups or even on what basis to start. An ML model can find commonalities & make broad segregations which a human can then look at & go "Oh, these are all dank memes!" & so on.

Since we saw "supervised" & "not supervised" a third type seems impossible. Well, it's a mix of the two: Sequential learning. It involves iteratively performing tasks, unsupervised, but using the outcome as feedback, and improving, something akin to supervision. For instance, you can teach a PC to play Pacman, or the Dinosaur Game or heck, Chess, with Reinforcement learning. This is a form of Sequential Learning which uses to its advantage that there is a clear metric of victory, manifested as a "Reward Function." Although I might have uses for RL, I will not deal with it for now. Another type of SL that has caught my eye is that of Multi Armed Bandits, a complex optimisation tool, which I will get into if I take the course of the same name under prashanth sir.

With the big picture clear, it's Time to get off on our journey. Conventionally, we begin at supervised learning, but since I've already dealt with that in the past, I'll be the contrarian for fun and start with Representation Learning a type of Unsupervised learning that is both intrinsically useful & vastly enhances the performance of supervised learning algorithms.

# Part 1: Representation Learning

## Chapter 1: Principal Component Analysis

Consider the points: $\begin{bmatrix}-7\\-14\\7\end{bmatrix}$ $\begin{bmatrix}7.5\\5\\-2.5\end{bmatrix}$ $\begin{bmatrix}0.5\\1\\-0.5\end{bmatrix}$ $\begin{bmatrix}0\\0\\0\end{bmatrix}$

How many numbers does it take store this data?

Naively, you may say $3\times4=12$. But really, you only need 7.

$\begin{bmatrix}-1\\-2\\1\end{bmatrix}$ is scaled by $\{7, -2.5, -0.5, 0\}$ that's 7 numbers.

This ↑ is called the Representative, & these ↑, the coefficients. It exploits the fact that despite existing a 3D space, this data is fundamentally 1D.

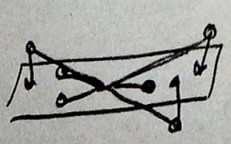
Now, you can think of this as a 'compression' in what data we are using.

If we had photos, of resolution $1024\times1024$, greyscale, all of human faces, each image can be thought of as a $2^{10}\times2^{10}$ matrix of numbers from 0 to 255.

If we 'vectorise' this, & lay it out as a $2^{20}\times1$ vector, each image is a point in $2^{20}$ dimensional space. That's about a million dimensions ($2^{10}\sim10^3$)

but do you really think that this data is evenly spread out in a million dimensions?

Just like our line in 3D, this might actually live in a lower dimensional subspace of the $2^{20}$ dimensional space. In reality you might get away with taking projections on a 1000 dimensional subspace & retain almost ALL useful information.

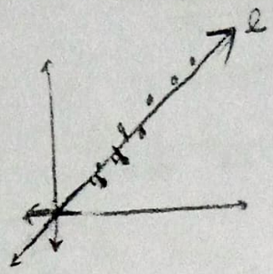


Roughly planar data = Project onto plane & things won't change much = You got rid of one dimension.

$10^6$ dimensional space → Step 1: Find an appropriate subspace to project onto
Step 2: Project & Enjoy!

The Core idea is this: project onto a low D space but retain almost all relevant data. Now, we will quantify this.

Simple case: 2D → 1D




Let the unit vector along line $l$ be $\hat{\omega}$

For each point $x$, the projection on the line is taken to be its 'proxy'. We will consider $x$ to be equal to $k\hat{\omega}$, for some $k$. This $k$, of course, is $\vec{x} \cdot \hat{\omega}$.

Goal: find the best possible $\hat{\omega}$.

$\Rightarrow$ Goal: minimise 'Representation error' $= \frac{1}{n}\sum_{i=1}^{n} \left| \vec{x_i} - (\vec{x_i} \cdot \hat{\omega})\,\hat{\omega} \right|^2$

Note: $\vec{a} \cdot \vec{b} = a^T b$ $\quad (1\times n \;\; n\times 1 \Rightarrow 1\times 1)$ ... on the other hand,

where $\hat{i}$ lands

$$\begin{bmatrix} \downarrow & & \\ - & - & - \end{bmatrix} \begin{bmatrix} - \\ - \\ - \end{bmatrix} \begin{matrix} \leftarrow \hat{i} \\ \leftarrow \hat{j} \\ \leftarrow \hat{k} \end{matrix}$$

where $\hat{j}$ lands

$$\begin{bmatrix} - \\ - \\ - \end{bmatrix} \begin{bmatrix} - & - & - \end{bmatrix}$$ all have only $\hat{i}$

where $\hat{i}$ lands $\quad \searrow \quad [1 \;\; 1 \;\; 1]$

Since we will only deal with vectors, I will drop all arrows & hats.

Minimise $\sum_i (x_i - \overbrace{x_i^T \omega}^{\text{scalar}}\, \omega)^T (x_i - x_i^T \omega \omega)$

$$= \sum_i x_i^T x_i - (x_i^T \omega)\, \omega^T x_i - x_i^T (x_i^T \omega)\, \omega + \underbrace{(x_i^T \omega)\, \omega^T (x_i^T \omega)\, \omega}_{\omega^T\omega = 1 \;\Rightarrow\; \text{Both are } (x_i^T\omega)^2}$$

$$= \sum_i x_i^T x_i - (x_i^T \omega)\, \omega^T x_i$$

Doesn't matter to the minimisation because it doesn't depend on $\omega$.

$\Rightarrow$ Maximise $\sum_i \underbrace{(x_i^T \omega)}_{\text{Scalar}} \underbrace{\omega^T x_i}_{\text{Scalar}} = \sum_i \omega^T (x_i x_i^T)\, \omega$

$$= \omega^T \left[ \sum x_i x_i^T \right] \omega$$

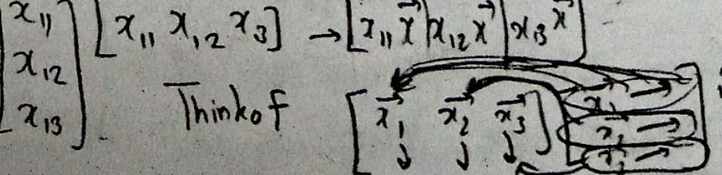
$$= \omega^T C \omega$$

$C = XX^T$ Convince yourself that it makes sense.

$$\begin{bmatrix} | & | & | & | & | & | \\ & & & x_i & & \end{bmatrix}$$

$$\begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} & x_3 \end{bmatrix} \rightarrow \left[ x_{11}\vec{x} \,\middle|\, x_{12}\vec{x} \,\middle|\, x_3\vec{x} \right]$$

Think of



I'm suggesting that you visualise each row scaling $\hat{i}$ & making a matrix, $\hat{j}$ doing the same & adding all these matrices together.

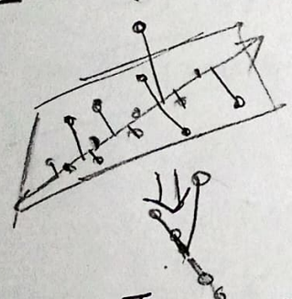
This is an alternative, less geometric way to see the product, instead of each column in $M_2$ seen as mapped to a new point scaled by the columns of $M_1$.

$C = \frac{1}{n} X X^T$ is known as the covariance matrix.

We want the dot product of $\hat{w}$ & the transformed version of $\hat{w}$ to be as big as is possible. $\Rightarrow$ It's BEST if $\hat{w}$ & its transformation are <u>linearly dependent</u>. $\Rightarrow$ $\hat{w}$ is an eigenvector of $C$. (See Min-Max Theorem for Rigour)

There's bound to be more than one eigenvector, so pick the one with the maximum eigenvalue.

Let's say I did this on 3D data.



I find that <u>most</u> <u>error vectors</u> lie on a line in 2D space. (only 2 components for error vector: along plane & out of plane.)

Now, I can run PCA again, to get another line for the error vectors, so that we get another <u>basis vector</u> for our 2D <u>subspace</u>.

It just so happens that this next basis vector is the next eigenvector of the covariance matrix. In fact, what we are doing, is switching to an <u>eigen basis</u> of $C$. If we use $n$ vectors, there's no compression, but typically the top $k$ vectors will suffice.

$\frac{\lambda_1 + \lambda_2 + \dots \lambda_k}{\sum \lambda_i}$ is a measure of how much data we retain.

$k$ is chosen such that this is $\geq 95\%$.

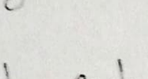
I've just made 2 unjustified claims, both of which must feel rather reasonable/intuitive after the fact.

Considering that Wikipedia itself says 'it turns out' that the sum of eigenvectors, I'd rather not try to prove it.

The 95% benchmark parameter is just a heuristic & doesn't appear to hold mathematical significance.

Nevertheless, I'd like a small detour to appreciate the covariance matrix because we keep running into each other, and I promptly sweep it under the rug. The first time I encounted it was in the context of random variables.

The interdependence of 2 variables is measured by $E[(x-\bar{x})(y-\bar{y})]$ which ought to average to zero for independent events.



For more than 2 variables, each variable has a covariance with each other one, represented in an $n \times n$ symmetric matrix.

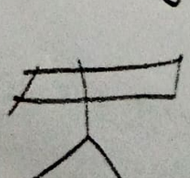
The covariance as a number represents how strongly 2 variables are correlated.

$$\begin{bmatrix} x_1x_1 & x_1x_2 & x_1x_3 \\ x_2x_1 & x_2x_2 & x_2x_3 \\ x_3x_1 & x_3x_2 & x_3x_3 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$$

Now, clearly summing this over a large amount of data gives the actual matrix because we approach the expected value.

This is EXACTLY what we had too. Only, the subtraction of the mean is missing. That's fine, because before we run PCA, we must center the data by subtracting away means.

This is so that



planes that are NOT a subspace (don't contain Zero so can't be represented with basis change) don't escape the radar of PCA.

We merely do $\vec{x}_i \to \vec{x}_i - \vec{\bar{x}}$

C is thus symmetric by virtue of the symmetry in the alternate definition & of the obvious symmetry in multiplying out the x vector.

A cool thing about C is that it's "Positive Semidefinite".

Positive definite matrix = A Hermitian matrix that satisfies

$$z^* H z > 0 \quad \forall z \in \mathbb{C}^n$$

For Real matrices, this means $z^T S z > 0 \; \forall z \in \mathbb{R}^n$

Positive semidefinite matrix $\Rightarrow z^* H z \geq 0 \; \forall z \in \mathbb{C}^n$

That equals sign is all the difference.

Because it's real symmetric, it's Hermitian, which means automatically that all its eigenvalues are real & orthogonal*. Further, due to the semidefiniteness no eigenvalue is negative. (That would give $v^T S v < 0$)

→ C is Hermitian, Positive Semidefinite & has Positive real eigenvalues.

Why is it positive semidefinite? We already saw it, almost:

$$\frac{1}{n}\sum_i (x_i^T \omega)(\omega^T x_i) = \omega^T \left(\frac{1}{n}\sum x_i^T x_i\right)\omega$$

But this also $\frac{1}{n}\sum (\omega^T x_i)^T (\omega^T x_i) = \boxed{\frac{1}{n}\sum (\omega^T x_i)^2 \geq 0}$

$\Rightarrow \omega^T C \omega \geq 0 \; \forall \; \omega \in \mathbb{R}^n$ QED

---

* $\langle e_j | H | e_i \rangle = \lambda_i \langle e_j | e_i \rangle$

$\langle e_i | H | e_j \rangle = \lambda_j^* \langle e_j | e_i \rangle$

$\rightarrow (\lambda_i - \lambda_j^*)\langle e_j | e_i \rangle = 0$

when $i=j$, $\langle e_i | e_i \rangle \neq 0$, so $\lambda_i = \lambda_i^* \; \forall i$

$i \neq j$ $\lambda_i \neq \lambda_j$ so $\langle e_i | e_j \rangle = 0$

---

Anyway, I'm happy we got an intuitive feel for & manged to understand the properties of the Covariance matrix. Maybe I'll read more on Kalman Filters later, now that I won't be hit with full panic attacks every 2 seconds.

That aside, let's get back to PCA.

Notice that in maximising $w^T C w$, we are maximising $\sum (w^T x)^2$ the sum of the lengths of the component of $x_i$ along $w$.

If we do PCA once, we get $\{w^T x_1, w^T x_2 \ldots\}$ as the coefficients & $w$ as representative. The variance of this data is $\frac{1}{n}\sum_i \left(w^T x_i - \frac{\sum w^T x_j}{n}\right)^2$

$\frac{1}{n}\sum_j w^T x_j = \frac{w^T}{n}\sum_j x_j = 0$ because we centered the data.

$\Rightarrow$ Var $= \frac{1}{n}\sum (w^T x_i)^2$ $\Rightarrow$ We maximised the Variance of data using $w$. We managed to find that direction which could show the greatest variability even after projection, thereby capturing data.

Further: $Cv = \lambda v \Rightarrow v^T C v = v^T \lambda v$

$\Rightarrow \frac{1}{n}\sum (v^T x)^2 = \lambda$

The quantity we maximise is also the eigenvalue of the vector. That's a more concrete justification for choosing top k eigenvectors, but we pretty much had the same line of thought earlier as well.

---

## SUMMARY

→ Data may be in low D subspace of data. Relevant & Important features are sought.

→ Compute matrix $C = \frac{1}{n} XX^T$ $\left[O(n^{\log_2 7}) \text{ or naively, } O(n^3)\right]$

→ Find eigenvectors of $C$ $\left[O(kn^2) \text{ for top } k, \text{ or naively, } O(n^3)\right]$

→ Pick top k eigenvalues & use the corresponding vectors as your basis.

$n$ D space → $k$ eigenvectors (all $n$ D vectors) → Projections of each data point along each eigenvector

no of data points

$n \cdot d \longrightarrow \boxed{kn} + \boxed{d \cdot k}$

eg: $n = 10^6$ $d = 10^{12}$ $k = 10^3$

$10^{18}$ before $10^9 + 10^{15} \approx 10^{15}$ approx $\frac{1}{1000}$ compression.

More importantly $\frac{1}{1000}$ dimensions ⇒ Better features for further training.

# CHAPTER 2: Extending PCA

There are a couple of issues with what we just outlined.

- First, is the time complexity.
- Second, is the fact that lower dimension manifolds that are non linear in nature cannot be found this way. For instance this 2D sheet that's curved will not be accurately represented by any plane. PCA will just tell you, "you need 3D"

---

One day, you get your hands on a dataset such that d, the no of data points is much much lesser than n.
This gets you thinking of a way to bypass the $O(n^3)$ complexity.
To be fair, it's unlikely I'd have come up with this, but the core idea is to pivot on the fact that $XX^T$ is $n \times n$, while $X^TX$ is $d \times d$, and the two have connected eigenvectors.

Let v be an eigenvector of $C = XX^T$. $\quad XX^Tv = \lambda v \Rightarrow X^TX(X^Tv) = \lambda(X^Tv)$

$\Rightarrow X^TX$ has eigenvectors $= X^Tv \; \forall \; v \in EV$ of $XX^T$ & SAME EIGENVALUES!

Further, because $X^TXw = \lambda w \Rightarrow XX^T(Xw) = \lambda(Xw)$, $XX^T$ & $X^TX$ share the entire set of non zero eigenvalues. Anything different about the two is just the number of zero eigenvalues.

Now, let's try & use this to our advantage.

---

We want set of w satisfying $Cw = \lambda w \Rightarrow \frac{1}{n}(\sum x_i \underbrace{x_i^T}_{\text{scalar}})w = \lambda w$

$\Rightarrow w = \sum \underbrace{\left[\frac{x_i^T w}{n\lambda}\right]}_{\text{scalar}} x_i$ each ev is a LINEAR COMBINATION of EVERY DATA POINT.

① $w_k = \sum \alpha_i x_i = X\alpha_k$ where $\alpha$ is a vector of $\alpha_i$ s.

But, we already saw that the vector w having ev $\lambda$ for $XX^T$, corresponds to the vector $X^Tw$ having ev $\lambda$ for $X^TX$.

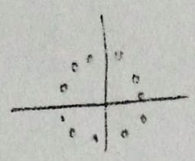
$\Rightarrow (X^TX)(X^TX\alpha_k) = (X^TX\alpha_k) \cdot \lambda_k$

This equation will certainly hold true if $X^TX\alpha_k = \lambda_k \alpha_k$

---

That is, if we find eigenvectors of $X^TX$, we get the corresponding eigenvectors of $XX^T$ for free by doing $X\alpha$. Hey! I know that 10 steps ago! Did I just RUN IN CIRCLES???

Anyway, we can find evs in $O(d^3)$ instead of $O(n^3)$ if $n \gg d$.

Consider  PCA will tell you no line can represent this

But it is a line. It's a Curved Line, $f(t)$.

We need an arbitrary 2nd order polynomial to represent this particular

$$ax^2 + by^2 + gx + fy + hxy + c = 0$$

$$\equiv \underbrace{[a\ b\ g\ f\ h\ c]}_{\Phi^T} \begin{bmatrix} x^2 \\ y^2 \\ x \\ y \\ xy \\ 1 \end{bmatrix} = 0$$

We can do the mapping $(x,y) \longrightarrow (x^2, y^2, x, y, xy, 1)$

$\Rightarrow$ In this 6D space, There exists $\phi$ such that $\phi^T x_i \approx 0 \ \forall i$

$\Rightarrow$ All the data is <u>ORTHOGONAL</u> to $\phi \Rightarrow x_i$ is along a LINE

$\Rightarrow$ PCA will find it.

But we had to move our data into a higher dimensional space.

But that's okay! You can take to a ridiculously high D space.

When $N \gg d$, we have a workaround: It's only $O(d^3)$ now.

But umm... we need to ~~do~~ $x_i \longrightarrow f(x_i)$ & this mapping to a higher D space might get expensive.

Not 'might', it WILL get expensive. We are mapping to a very high D so each data point needs to compute D numbers.

Let's outline our process & see if we can bypass this step.

→ Design $f: \mathbb{R}^d \longrightarrow \mathbb{R}^D$ a little abuse of notation, $f$ acts on each $x_i$

→ Compute $f(x_i) \ \forall x_i$ → Find $f(X)^T f(X)$

→ Find eigenvalues & eigenvectors of this matrix.

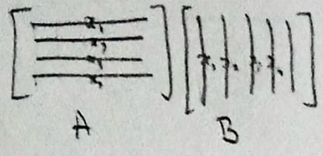
→ Convert eigenvectors to those of $f(X)f(X)^T$ by doing $f(X)v$

→ Project $f(x_i)$ onto the top k eigenvectors.

Store $\left\{ \begin{bmatrix} f(x_1)^T w_1 \\ \vdots \\ w_2 \end{bmatrix}, \begin{bmatrix} f(x_2)^T w_1 \\ \vdots \\ \vdots \end{bmatrix} \dots \right\} \quad \{w_1, w_2 \dots\}$

Ignoring the penultimate step, we don't really need $f(x_i)$. We only need $f(x)^T f(x)$ whose elements will be: $e_{ij} = \sum_k A_{ki} B_{jk}$

$= f(x_i)^T f(x_j)$

E - 


⇒ The matrix elements need only the dot products of all pairs of data points in the mapped space.

This means we can get away with finding dot products BEFORE mapping, & bypass the entire f mapping if we can find a function that can transform the dot product of 2 vectors into their dot product in the transformed space.

Consider $(x,y) \rightarrow \begin{bmatrix} x^2 \\ y^2 \\ x \\ y \\ xy \\ 1 \end{bmatrix}$ $f(x_1)^T f(x_2)$, the dot product of 2 arbitrary vectors is

$$x_1^2 x_2^2 + y_1^2 y_2^2 + x_1 x_2 + y_1 y_2 + x_1 y_1 x_2 y_2 + 1$$

Clearly, we only ever see $x_1 x_2$ & $y_1 y_2$ terms, the elements of $x_1^T x_2$

Maybe this just could work out.

Consider $(x_1^T x_2 + 1)^2$. It's $x_1^2 x_2^2 + y_1^2 y_2^2 + 2x_1 x_2 y_1 y_2 + 1 + 2(x_1 x_2 + y_1 y_2)$

Ignoring coefficients, it's the same as $f(x)^T f(x)$.

If we choose $f$ to be $(x,y) \longrightarrow (x^2, y^2, \sqrt{2}xy, \sqrt{2}x, \sqrt{2}y, 1)$

then $(x_1^T x_2 + 1)^2$ Is $f(x_1^T) f(x_2)$

This choice of $f$ has not caused any loss of generality. (We can work with any $f$ as long as the data lies in a linear subspace of the mapped space.

Since it's guaranteed that every combination of $x_1 x_2, y_1 y_2$ & their powers exist in $(x_1^T x_2 + 1)^p$, this can be used to find the dot product in the $p^{th}$ dimension, although the mapping function will have to be chosen with $\sqrt{\text{coefficients}}$.

Now, we need to work around the use of $f(x)$ in the last two steps. $f(x_i)^T w_i$ is what we wish to know.

$$\Rightarrow \quad f(x_1)^T f(X) v = f(x_1)^T \sum_i f(x_i) v_i = \sum_i v_i \underbrace{\left(f(x_1)^T f(x_i)\right)}$$

A dot product we know.

# KERNEL PCA

Step 1: Given data $X_{n \times d}$, find a suitable kernel function $k: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$

Step 2: Compute matrix $K_{d \times d}$ where $K_{ij} = k(x_i, x_j)$ Center the matrix $K$

Step 3: Find the eigenvectors $P_1', P_2', P_3' \ldots P_\ell'$ of $K$. Normalize $P_i = \frac{P_i'}{\sqrt{n \lambda_i}}$

Step 4: Compute Coefficients: $x_i \to \begin{bmatrix} \sum_j K_{ij} \alpha_{1j} \\ \sum_j K_{ij} \alpha_{2j} \\ \vdots \end{bmatrix}_{\ell \times 1}$ where $P_i = \begin{bmatrix} \alpha_{i1} \\ \alpha_{i2} \\ \vdots \end{bmatrix}$

Use this new vector in place of $x_i$

---

Notice that we simply gave up on finding the Representatives. We transfered $x$ into SOME high dimensional space, and it formed some subspace. We do not care about the function mapping $x$ to a higher dimension space, We do not care about the exact vectors in the higher dimensional space or about inverting the simplified vector to the original space.

We could do any of that, the information is here, we just don't care. We extracted out the MOST RELEVANT data, & put $x$ into a lower dimensional subspace. That's a success.

---

Note 1: Kernel functions

$k(x_i, x) = (x_i^T x + 1)^7$ polynomial kernel

$= e^{\frac{\|x_i - x\|^2}{2\sigma^2}}$ gaussian kernel / radial map

$(\sigma \in \mathbb{R}^+)$

This popular map, takes $x$ into a vector of "infinite dimensions, so $f(x)$ & the representative vectors become simply unknowable.

A kernel is meant to capture the dot product after a function transforms the vector. Thus a Kernel fn must always be symmetric. $k(x_i, x_j) = k(x_j, x_i)$ (& the Kernel)

Mercer's theorem states necessary & sufficient conditions for a function $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ to be a valid kernel fn, but I can't make sense of the theorem. This much is clear though, the matrix $K$, made with any valid is a positive semidefinite matrix, as its non zero eigenvalues are shared with the mapped space Covariance matrix, which we know to be positive semidefinite.

An important thing to note is that the kernel needs to be centered. The only subspaces PCA can detect pass through the origin. Even if $X$ is centered, there's no guarantee $f(X)$ is centered.

So, the subtraction on $f(X)$ is algebraically extended to $K$ & an expression can be derived to center the matrix $K$.