

Latent Structures and PCA

Daniel Lawson University of Bristol

Lecture 03.1 (v2.0.0)

Signposting

- ▶ In Block 2, we covered:
 - ▶ Regression
 - ▶ Classical Statistical Testing
 - ▶ Resampling - Bootstrap and Cross-Validation
 - ▶ Model Selection
- ▶ These may be applied directly to data, but what if this is high dimensional?
 - ▶ These methods and more can be used in **dimensionality reduced space**.

Questions

- ▶ How do we interpret Principle Components Analysis?
- ▶ Is an Eigenvector space a latent space?
- ▶ Are parameters of a model a latent space?
- ▶ How are these to be used in Data Science?

Latent Structures

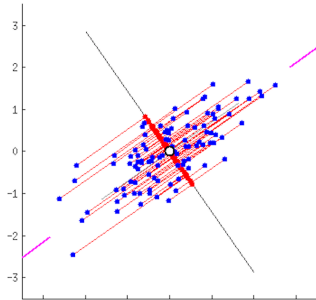
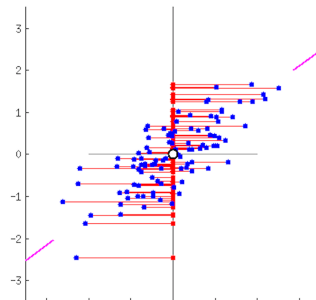
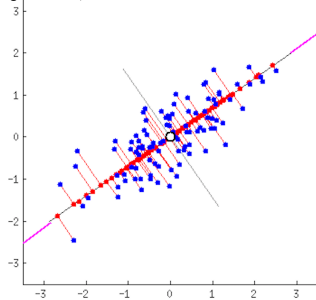
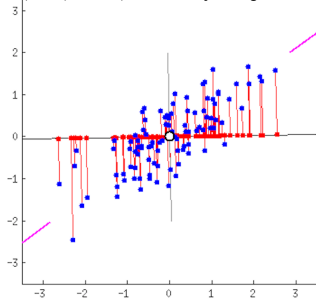
- ▶ It is often useful to think of data X being represented in some (usually lower-dimensional) space θ .
- ▶ θ is the **latent space** for X .
- ▶ Examples:
 - ▶ Parameters: $X_i \sim f(\theta_i)$ for some model f
 - ▶ Kernel representation: $X_i = \sum_{j=1}^K K(\theta_i)$
 - ▶ Factor analysis
 - ▶ Spectral decomposition, Principal Components Analysis, Singular Value Decomposition

Latent Structure

- ▶ What makes this a **latent space** instead of a parameterisation is the modelling done on that space.
- ▶ i.e. it is constructed to *mean something*
- ▶ If it is done with the intention of making *similar data be close* then we might call this an embedding
- ▶ Much care is needed around the words “similar” and “close”!

PCA Example from Stack Exchange

<https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues/140579#140579>



Covariance

- ▶ Let X be an n by m matrix.
- ▶ Consider that X has been **mean centred**, so that $\mathbb{E}(X_{:,j}) = 0$ for columns j . Then:

$$C = \text{Cov}(X) = \frac{1}{n-1} X^T X$$

- ▶ this is an unbiased estimator; the factor $1/(n-1)$ arises because we used the data to estimate the mean.
- ▶ C is an m by m matrix.
- ▶ We might also standardize the variance so that $\text{Var}(X_{:,j}) = 1$.

Principal Components Analysis

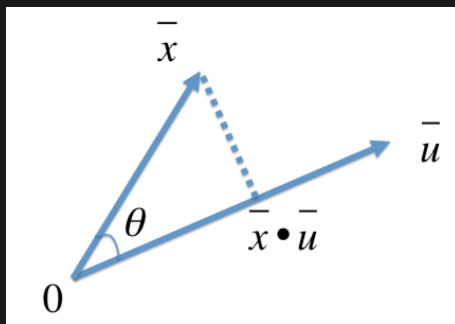
- ▶ Consider data X for which we seek the decomposition:

$$C = \text{Cov}(X) = U\Sigma U^T$$

- ▶ Where:
 - ▶ D is a diagonal matrix of the m **eigenvalues**.
 - ▶ U is a matrix of m **eigenvectors** in columns.
- ▶ We'll construct this matrix by sequentially:
 - ▶ Finding a **projection** of X onto a k dimensional subspace;
 - ▶ that **maximises the explained variance**, or equivalently, **minimises the squared error** in the prediction;
 - ▶ conditional on this being **orthogonal** to all previous subspaces.

One dimensional projection

- ▶ We will project \vec{x} onto a subspace U .



- ▶ In one dimension this is a line defined by \vec{u} of unit length through the origin.
- ▶ The projection of \vec{x}_i onto \vec{u} is $(\vec{x}_i \cdot \vec{u})$ in the new coordinate system.
- ▶ Recall $\vec{x} \cdot \vec{y} = |\vec{x}||\vec{y}| \cos(\theta)$

One dimensional projection

- ▶ We will compute the projection in the old coordinate system, written $(\vec{x}_i \cdot \vec{u})\vec{u}$.
- ▶ Using the properties of \vec{u} the residuals are therefore:

$$\|\vec{x}_i - (\vec{x}_i \cdot \vec{u})\vec{u}\|^2 = (\vec{x}_i - (\vec{x}_i \cdot \vec{u})\vec{u}) \cdot (\vec{x}_i - (\vec{x}_i \cdot \vec{u})\vec{u}) \quad (1)$$

$$= \vec{x}_i \cdot \vec{x}_i - 2(\vec{u} \cdot \vec{x}_i)^2 + (\vec{u} \cdot \vec{x}_i)^2(\vec{u} \cdot \vec{u}) \quad (2)$$

$$= \vec{x}_i \cdot \vec{x}_i - (\vec{u} \cdot \vec{x}_i)^2 \quad (3)$$

One dimensional projection

- Averaging over all vectors:

$$MSE(\vec{u}) = \frac{1}{n} \sum_{i=1}^n \vec{x}_i \cdot \vec{x}_i - (\vec{u} \cdot \vec{x}_i)^2,$$

- But the first term is constant, so we therefore are seeking to **maximise**:

$$\frac{1}{n} \sum_{i=1}^n (\vec{u} \cdot \vec{x}_i)^2,$$

- This is the second moment of $\vec{u} \cdot \vec{x}_i$ which can be written as:

$$\mathbb{E}(\vec{u} \cdot \vec{x}_i)^2 + \text{Var}(\vec{u} \cdot \vec{x}_i).$$

- However, the mean of the projection is zero by construction so **minimising the MSE is equivalent to maximising the variance explained.**

Multiple dimensions

- ▶ To work with multiple dimensions:
 - ▶ replace the single vector projection $(\vec{u} \cdot \vec{x}_i)\vec{u}$ with a sum over all dimensions $\sum_{k=1}^K (\vec{u}_k \cdot \vec{x}_i)\vec{u}_k$.
- ▶ In matrix notation, with our conventions, $\vec{U} = XU$
- ▶ It is straightforward to show that the cross terms cancel out.
- ▶ This is due to the orthogonality constraint.

Maximising variance

- ▶ Using matrix notation,

$$\text{Var}(\vec{U}) = \frac{1}{n-1} (\mathbf{XU})^T (\mathbf{XU}) \quad (4)$$

$$= \mathbf{U}^T \frac{\mathbf{X}^T \mathbf{X}}{n-1} \mathbf{U} \quad (5)$$

$$= \mathbf{U}^T \mathbf{C} \mathbf{U} \quad (6)$$

- ▶ Where \mathbf{C} is the covariance.
- ▶ We need to constrain the search over \mathbf{U} to look for unit vectors.
- ▶ We do this with a **Lagrange multiplier** λ , which allows unconstrained optimisation of a different problem.
- ▶ Lagrange multipliers are important tools for many optimisation problems in Data Science.

Leveling with Lagrange: An Alternate View of Constrained Optimization

DAN KALMAN

American University
Washington, D.C. 20016
kalman@american.edu

In most calculus books today [11, 14, 15], Lagrange multipliers are explained as follows. Say that we wish to find the maximum value of f subject to the condition that $g = 0$. Under certain assumptions about f and g , the Lagrange multipliers theorem asserts that at the solution point, the gradient vectors ∇f and ∇g are parallel. Therefore, either $\nabla f = \lambda \nabla g$ for some real number λ , or $\nabla g = 0$. Combined with the equation $g = 0$, this gives necessary conditions for a solution to the constrained optimization problem. We will refer to this as the standard approach to Lagrange multipliers.

An earlier tradition approaches this subject far differently. It defines a new function, $F = f + \lambda g$, that incorporates both the objective function and the constraint, and in which λ is considered to be an additional variable. Here, F is referred to as a *Lagrangian function*. The conditions for F to achieve an unconstrained extremum are then determined, and these become necessary conditions for a solution to the original problem. This is the Lagrangian function approach to Lagrange multipliers.

Optimisation

- ▶ For simplicity we'll just add the k -th dimension, conditioning on orthogonality to the previous $k - 1$
 - ▶ This is how many algorithms work in practice
- ▶ Constraint: $\mathbf{u}^T \mathbf{u} = 1$. Therefore

$$\mathbb{L}(\mathbf{u}, \lambda) = \mathbf{u}^T \mathbf{c} \mathbf{u} - \lambda(\mathbf{u}^T \mathbf{u} - 1) \quad (7)$$

$$\frac{\partial \mathbb{L}}{\partial \lambda} = \mathbf{u}^T \mathbf{u} - 1 \quad (8)$$

$$\frac{\partial \mathbb{L}}{\partial \mathbf{u}} = 2\mathbf{c} \mathbf{u} - 2\lambda \mathbf{u} \quad (9)$$

- ▶ Which can be solved for when the derivatives are zero to get

$$\mathbf{u}^T \mathbf{u} = 1 \quad (10)$$

$$\mathbf{c} \mathbf{u} = \lambda \mathbf{u} \quad (11)$$

PCA

- ▶ We have shown that the **eigenvector** \mathbf{u} and **eigenvalue** λ solve:

$$\mathbf{C}\mathbf{u} = \lambda\mathbf{u}$$

- ▶ The eigenvectors can be arranged into a matrix U with eigenvectors on the columns, and the eigenvalues into a diagonal matrix Σ . Then:

$$\mathbf{C} = \mathbf{U}\Sigma\mathbf{U}^T$$

- ▶ (Care needs to be taken with zero/repeated eigenvalues.)

Interpretation

- ▶ Our basis \mathbf{U} is **orthonormal**, and
- ▶ As \mathbf{C} is a covariance matrix, it is symmetric.
- ▶ Therefore the **eigenvectors are orthogonal**.
- ▶ The **eigenvalue** (i.e. λ_k) is the **variance explained** by the k -th PC.
- ▶ The proportion of **variance explained** by K PCs is
$$R^2 = \frac{\sum_{k=1}^K \lambda_k}{\sum_{m=1}^M \lambda_m}.$$
- ▶ If the data are really in a K dimensional subspace, the eigenvalues beyond that should be 0.

Singular value decomposition

- ▶ PCA and SVD are related:

$$\text{SVD}(X) = UDV^T$$

- ▶ and therefore

$$\frac{1}{n-1}X^TX = \frac{1}{n-1}VD^TU^TUDV^T = V\frac{D^TD}{n-1}V^T = V\Sigma V^T$$

- ▶ where $\Sigma = D^TD/(n-1)$ are both diagonal matrices.

Data preparation

```
## Make a 4 column dataset and log-transform
testdata=conndata[,c("orig_bytes","resp_bytes",
                     "orig_ip_bytes","resp_ip_bytes")]
testdata[testdata=="-"]=0
testdata[testdata=="0"]=0
for(i in 1:4) testdata[,i]=log10(as.numeric(testdata[,i])+1)
rownames(testdata)=NULL
```

Data preparation

```
## Make a 4 column dataset and log-transform
testdata=conndata[,c("orig_bytes","resp_bytes",
                     "orig_ip_bytes","resp_ip_bytes")]
testdata[testdata=="-"]=0
testdata[testdata=="0"]=0
for(i in 1:4) testdata[,i]=log10(as.numeric(testdata[,i])+1)
rownames(testdata)=NULL

## Make a test dataset for example purposes
set.seed(1)
myindex=sample(1:dim(testdata)[1],2000)
testdata_sample=testdata[myindex,]
## And categories, for plotting purposes
testdatacat=as.factor(paste(conndata[, "proto"],
                             conndata[, "service"], sep="_"))[myindex]
```

Do the decompositions

```
## The direct (naive) way to standardize  
testdata_scaled <- apply(testdata_sample, 2, scale)  
  
## But we need to standardize *samples*  
testdata_t=t(testdata_scaled)  
testdata_t_scaled <- apply(testdata_t, 2, scale)
```

Do the decompositions

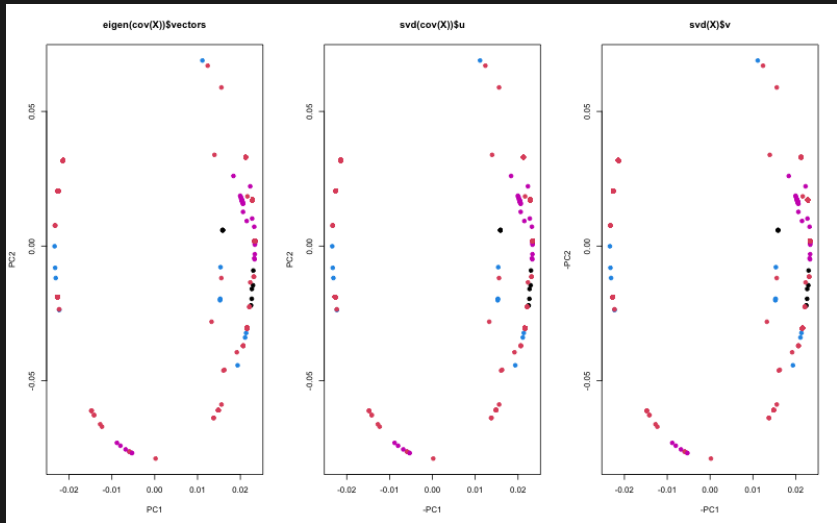
```
## The direct (naive) way to standardize
testdata_scaled <- apply(testdata_sample, 2, scale)

## But we need to standardize *samples*
testdata_t=t(testdata_scaled)
testdata_t_scaled <- apply(testdata_t, 2, scale)

## Different ways to compute Spectral decompositions
testdata.cov <- cov(testdata_t_scaled)
testdata.eigen <- eigen(testdata.cov)
testdata.cov.svd <- svd(testdata.cov)

# Faster: SVD on the original data matrix
testdata.svd <- svd(testdata_t_scaled)
testdata.prcomp <- prcomp(testdata_t_scaled)
```

PCA/SVD/Covariance plots



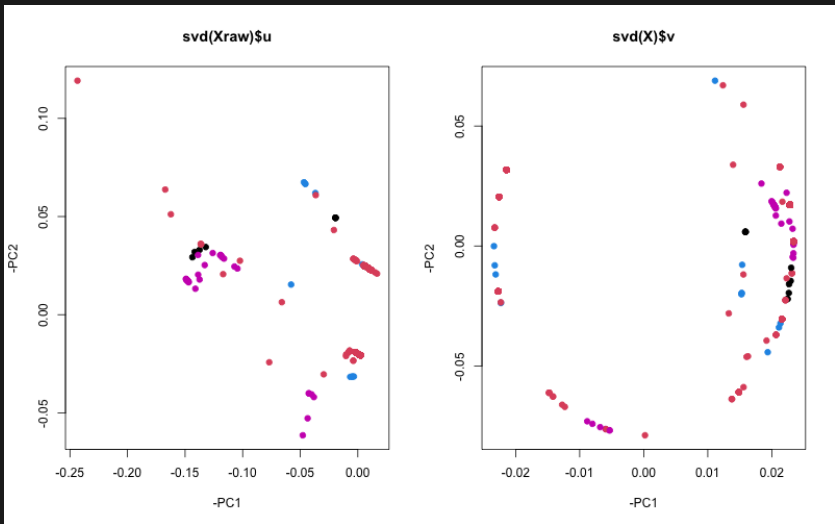
Plotting code

```
png("../media/03.1.1-EigenExample.png",height=500,width=800)
par(mfrow=c(1,3))
plot(testdata.eigen$vectors[,1],
testdata.eigen$vectors[,2],xlab="PC1",ylab="PC2",
     main="eigen(cov(X))$vectors",
     col=as.numeric(testdata.cat),pch=19,cex=0.5)
plot(-testdata.cov.svd$u[,1],
     -testdata.cov.svd$u[,2],xlab="-PC1",
     ylab="PC2",main="svd(cov(X))$u",
     col=as.numeric(testdata.cat),pch=19,cex=0.5)
plot(-testdata.svd$v[,1],
     -testdata.svd$v[,2],xlab="-PC1",ylab="-PC2",
     main="svd(X)$v",
     col=as.numeric(testdata.cat),pch=19,cex=0.5)
dev.off()
```


Scaling matters

```
testdata.direct.svd <- svd(testdata_scaled)
png("../media/03.1.2-SVDscaling.png",height=500,width=800)
par(mfrow=c(1,2))
plot(-testdata.direct.svd$u[,1],
     -testdata.direct.svd$u[,2],
     xlab="-PC1",ylab="-PC2",main="svd(Xraw)$u",
     col=as.numeric(testdatacat),pch=19,cex=0.5)
plot(-testdata.svd$v[,1],
     -testdata.svd$v[,2],xlab="-PC1",
     ylab="-PC2",main="svd(X)$v",
     col=as.numeric(testdatacat),pch=19,cex=0.5)
dev.off()
```

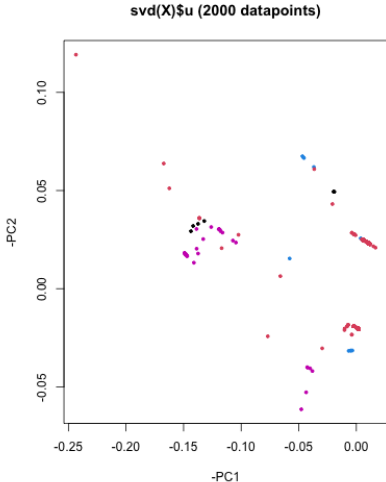
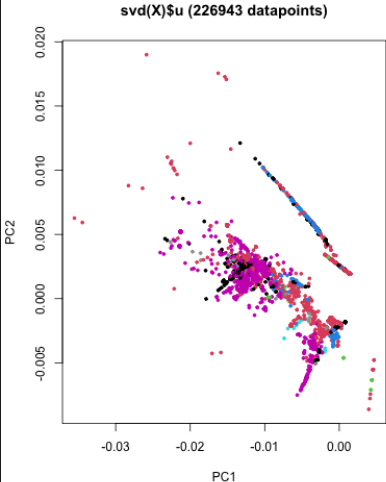
Scaling matters



Some notes

- ▶ The mean usually shows up in PC1 if you leave it in
- ▶ Here, mean centring X to \tilde{X} is weird; its re-weighting features differently for each observation!
- ▶ $SVD(X)$ and $SVD(\text{Cov}(\tilde{X}))$ usually contain the same structure, elucidated differently
- ▶ When there are many fewer features than observations, working with X directly is **much faster**
- ▶ This is because we work with an M^2 covariance matrix.
- ▶ If the number of features is higher than the number of datapoints, working with Covariance makes sense.

Full data



Full data

```
testdata_all_scaled <- apply(testdata, 2, scale)
dim(testdata_all_scaled)
## [1] 226943      4
testdata_all.svd=svd(testdata_all_scaled)

testdatacat_all <-
  as.factor(paste(conndata[, "proto"],
    conndata[, "service"], sep="_"))
```

How many PCs?

- ▶ This last insight is often used to select only those PCs whose eigenvalues are “large enough” to justify inclusion. There are many procedures, including:
 - ▶ **scree plot**, looking for an elbow in the distribution
 - ▶ **EVs > 1** , justified by random graph theory
 - ▶ **Tracey-Widom theory**, with a similar prediction
 - ▶ **Horn’s criterion**, based on simulating random matrices for the remaining matrix structure after K are chosen
 - ▶ **Velicer’s MAP criterion**, similar
- ▶ In practice they are all similar, and any can be “wrong”, so common sense should be applied.
- ▶ What is always true is that Eigenvectors associated with Eigenvalues that are “too small” will contain some noise, even if they still contain a signal.

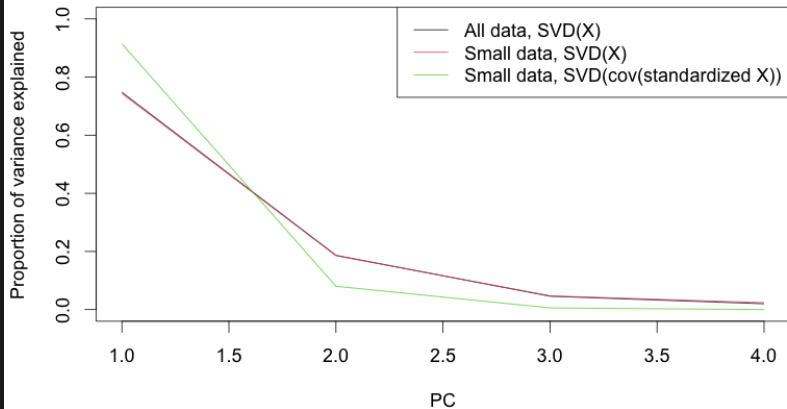
How many PCs?

- ▶ Here we have only 4 features. And we mean centred, which removes a degree of freedom. So the data should lie on a 3D subspace:

```
> round(testdata.eigen$values[1:4],8)
[1] 1859.25164 124.70157 16.04679 0.00000
> round(((testdata.svd$d)^2)/3,8)
[1] 1859.25164 124.70157 16.04679 0.00000
```

- ▶ Scaling by the number of features is not important for the SVD when computing the *proportion* of variance explained. Just square the singular values.

Variance Explained



Important properties:

- ▶ If X is **positive definite**, its eigenvalues are > 0 .
- ▶ If X is **positive semi-definite**, its eigenvalues are ≥ 0 .
- ▶ positive definite: If $\forall v \neq 0$ then $v \cdot xv > 0$
- ▶ positive semi-definite: If $\forall v \neq 0$ then $v \cdot xv \geq 0$
- ▶ The matrix A is **orthogonal** if $A^T = A^{-1}$. This is true iff all column vectors of A are orthonormal, and (equivalently) the row vectors are too. All eigenvalues of an orthogonal matrix are 1.
- ▶ If X is square and non-degenerate (distinct eigenvalues), its eigenvectors U form an **orthonormal basis**.

Projections are idempotent

- ▶ Once you project a vector into a subspace, projecting it again does nothing. Such projections P are called **idempotent**:

$$P^K = P$$

- ▶ Spectral projections have this property.

R commands for matrix operations

```
A %*% B # matrix multiplication  
t(A) # matrix transpose  
diag(A) # diagonal vector of A  
diag(x) # a diagonal matrix formed of the vector x  
det(A) # determinant  
sum(diag(A)) # trace  
solve(A) # matrix inverse  
eigen(A) # Eigenvalue decomposition  
svd(A) # Singular Value Decomposition
```

Reflection

- ▶ You should:
 - ▶ Be able to intuitively explain PCA, and perform simple calculations using it
 - ▶ Be able to relate PCA to SVD both mathematically and intuitively
 - ▶ Be able to deploy either appropriately on real data
- ▶ What is Spectral Decomposition doing? Why is it a good idea?
- ▶ How is it different to a model?
- ▶ What might it mean that two datapoints are “close” in a PCA plot?
- ▶ What might go wrong when making a 2-D PCA plot?

Signposting

- ▶ We will look at **clustering** - both on the raw data but also on PCs, which can often be used to avoid discrete features posing a problem as well as being efficient.
- ▶ We'll explore PCA in the Workshop.
- ▶ Further reading could include:
 - ▶ **Cosma Shalizi's Advanced Data Analysis, Lecture 18**
 - ▶ **Boyd and Vandenberghe: Convex Optimization** is an excellent and thorough resource.
 - ▶ I showed **Kalman: Leveling with Lagrange: An Alternate View of Constrained Optimization**