Math for Data Science

Dimensionality Reduction

Joanna Bieri DATA100

Important Information

- Email: joanna_bieri@redlands.edu
- Office Hours take place in Duke 209 unless otherwise noted – Office Hours Schedule

Today's Goals:

- Applications of Linear Algebra
- Dimensionality Reduction

Vectors and Matrices (review)

Span of a set of vectors is the whole set of possible vectors that can be created by taking a linear combination of the vectors in the set.

Two vectors are **Linearly Independent** if they do not lie along the same line.

We can tell if two vectors are **Linearly Dependent** because one will be a scalar multiple of the other... they have different magnitudes but the same direction.

Vectors and Matrices (review)

Matrix Dot Product

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

Vectors and Matrices (review)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

Vectors and Matrices (review)

Using Numpy

```
v = np.array([3,2])
A = np.array([[0,-1],[1,0]])
v_new = np.matmul(A,v)
print(v,v_new)

v_new = A @ v # Shortcut for matmul
```

Vectors and Matrices (review)

Matrix Identity and Inverse

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

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

`np.linalg.inv(A)`

Vectors and Matrices (review)

Matrix Determinants

The **determinant** measures something about the size of a transformation.

In two dimensions the determinant is give by

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = a * d - b * c$$

`np.linalg.det()`

Vectors and Matrices (review)

Eigenvalues and Eigenvectors

Eigenvalues and vectors are defined by the following equation

$$A\vec{v} = \lambda\vec{v}$$

```
eigenvalues, eigenvectors = np.linalg.eig(A)
```

Dimensionality Reduction

Dimensionality reduction in data science is a set of techniques used to reduce the number of features (or dimensions) in a dataset while retaining its most important information.

Dimensionality Reduction

Often real world data has thousands or millions of features - think a 1000 dimensional vector - and we want to reduce this so that the very complex data set is simplified in some way. BUT we don't want to loose the most important information!

Dimensionality Reduction

Dimensionality reduction can also be used to help with visualization of the data and improving computational efficiency.

Principal Component Analysis

Principal component analysis (PCA) is the most popular dimensionality reduction algorithms. The overall idea is the identify a hyperplane (plane or line in lower dimensions) that lies closest to the data so that we can project the data down onto that hyperplane. The whole algorithm focuses on choosing the correct hyperplane for the projection.

Principal Component Analysis

Here are the basic steps of the algorithm:

- Center the Data
- Preserve the Variance
- Eigenvalue Decomposition
- Component Selection
- Projection

Principal Component Analysis

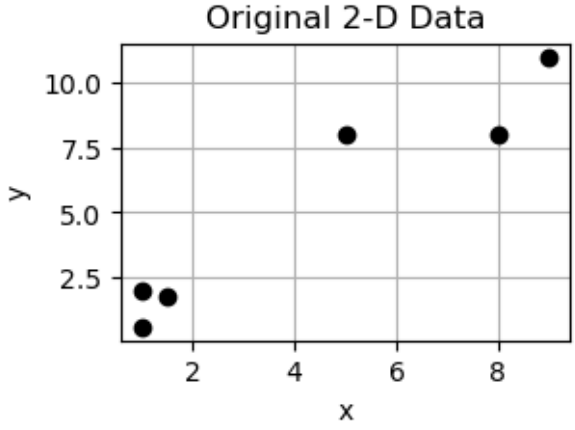
We will first go through a complete example to see how the linear algebra works and then I will introduce you to the sklearn function that does the PCA for us

Data

Here is a simple data set for us to use as an example

	x-feature	y-feature
0	1.0	2.0
1	1.5	1.8
2	5.0	8.0
3	8.0	8.0
4	1.0	0.6
5	9.0	11.0

Data



Data

Our data is two dimensional - there are two features labeled x-feature and y-feature. Each observation can be represented as a vector of length two

```
[1, 2]
[1.5, 1.8]
[5, 8]
[8, 8]
[1, 0.6]
[9, 11]
```

PCA

Now let's say that we want to reduce this two dimensional data down to a line (1D). But we don't want to lose the clear separation between the points.

PCA - Centering and Normalizing the data

PCA is very affected by the scale of the data and subtracting the mean of each feature centers the data so that each feature has a mean of zero.

PCA - Centering and Normalizing the data

It is also good practice in machine learning to restrict the magnitude of your data to the range $[-1, 1]$. There is not one unique way to normalize your data!

PCA - Centering and Normalizing the data

We will use

$$\bar{x_i} = \frac{x_i - \mu_x}{\sigma_x}$$

Notice that we are subtracting the mean and dividing by the standard deviation. We apply this to each feature independently. This should remind you of the z score from probability!

PCA - Centering and Normalizing the data

```
xmean = DF['x-feature'].mean()
ymean = DF['y-feature'].mean()
xstdv = DF['x-feature'].std()
ystdv = DF['y-feature'].std()

DF['x-centered'] = (DF['x-feature']-xmean)/xstdv
DF['y-centered'] = (DF['y-feature']-ymean)/ystdv
```

	x-feature	y-feature	x-centered	y-centered
0	1.0	2.0	-0.895381	-0.752657
1	1.5	1.8	-0.757630	-0.799214
2	5.0	8.0	0.206626	0.644026
3	8.0	8.0	1.033132	0.644026
4	1.0	0.6	-0.895381	-1.078550
5	9.0	11.0	1.308634	1.342368

PCA - Preserve the Variance

Next we want to calculate the covariance matrix to capture how each of the features in the data vary together. If the data set has n-features, then the result of this will be an nXn square matrix.

We have calculated the covariance before using

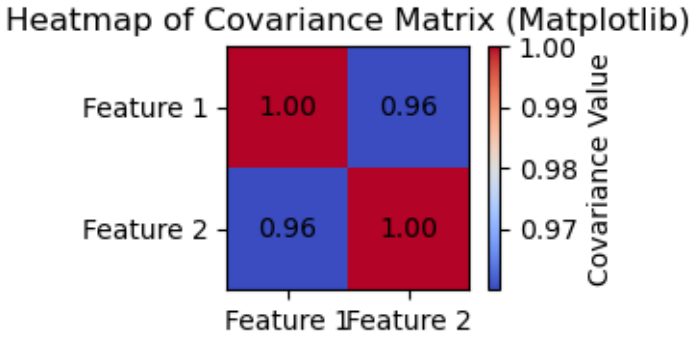
```
np.cov()
```

PCA - Preserve the Variance

```
covariance_matrix = np.cov(DF['x-centered'],DF['y-centered'])

# Then create a heatmap plot...
```

PCA - Preserve the Variance



PCA - Eigenvalue Decomposition

Next we want to use our covariance matrix and find the eigenvalues and eigenvectors of this matrix.

$$COV = \begin{bmatrix} 1. & 0.96004897 \\ 0.96004897 & 1. \end{bmatrix}$$

PCA - Eigenvalue Decomposition

Eigenvectors indicate the directions of maximum variance in the data (the principal components), while eigenvalues quantify the variance captured by each principal component.

PCA - Eigenvalue Decomposition

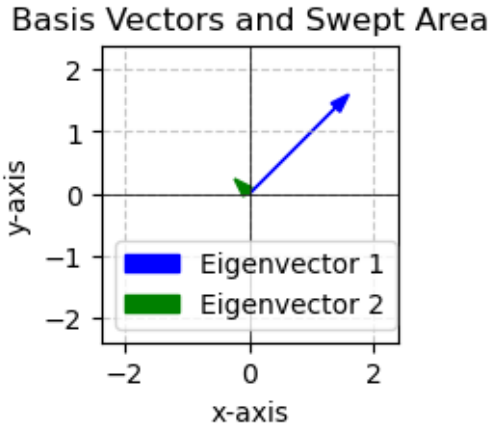
```
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

print("EIGENVALUES")
print(eigenvalues)

print("EIGENVECTORS")
print(eigenvectors)
```

```
EIGENVALUES
[1.96004897 0.03995103]
EIGENVECTORS
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

PCA - Eigenvalue Decomposition



PCA - Eigenvalue Decomposition

Here we see that one of the eigenvalues is MUCH bigger than the other!

$$\lambda_1 = 1.96004897$$

$$\vec{v_1} = \begin{bmatrix} 0.70710678 \\ 0.70710678 \end{bmatrix}$$

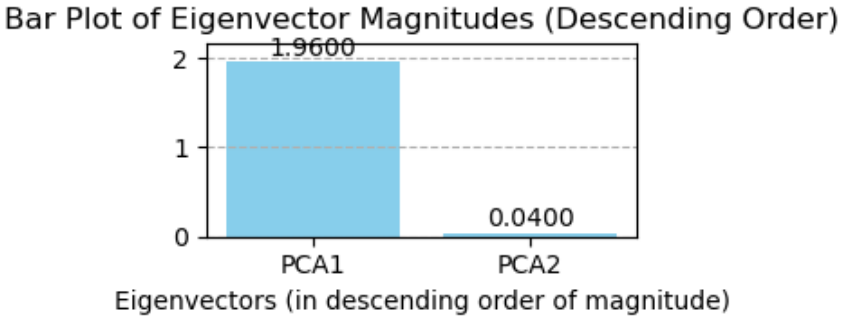
PCA - Eigenvalue Decomposition

The blue eigenvector points in the direction with the most variance, the green eigenvector points in the direction of the second most (in this case the least) variance.

PCA - Component Selection

The eigenvalues tell us about the data's variance the eigenvector tells us the direction. We often make a bar plot of the eigenvalues in descending order.

PCA - Component Selection



PCA - Component Selection

Looking at this bar plot we select the principal components with the most variance. In this two dimensional case we will choose just PCA1, and reduce our data down to one dimension.

In higher dimensional cases we can choose how many dimensions to project down to by choosing the number of principal components.

PCA - Project the Data

We are ready to project the data using a matrix multiplication. To do this we multiply the data vectors by the matrix of eigenvectors corresponding to the principal components we choose.

$$\vec{x}_{proj} = \vec{x} \cdot \tilde{Q}$$

PCA - Project the Data

Since we are projecting into one dimension our projection matrix is given by

$$\tilde{Q} = \begin{bmatrix} 0.70710678 \\ 0.70710678 \end{bmatrix}$$

PCA - Project the Data

So for the very first data point

$$\vec{x} = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

we first have to do the normalization, and then we can calculate the projected point as

$$\vec{x}_{proj} = \begin{bmatrix} -0.89538136 & -0.75265747 \end{bmatrix} \begin{bmatrix} 0.70710678 \\ 0.70710678 \end{bmatrix}$$

```
$$= -0.89538136*0.70710678+-0.75265747*0.70710678$$
```

```
$$= -1.1653394311885341$$
```

PCA - Project the Data

Do this same operation to each vector in our set of **normalized** observations

```
observations = DF[['x-centered', 'y-centered']].values
observations
```

```
array([[-0.89538136, -0.75265747],
       [-0.75763038, -0.7992136 ],
       [ 0.20662647,  0.6440265 ],
       [ 1.03313234,  0.6440265 ],
       [-0.89538136, -1.0785504 ],
       [ 1.3086343 ,  1.34236848]])
```

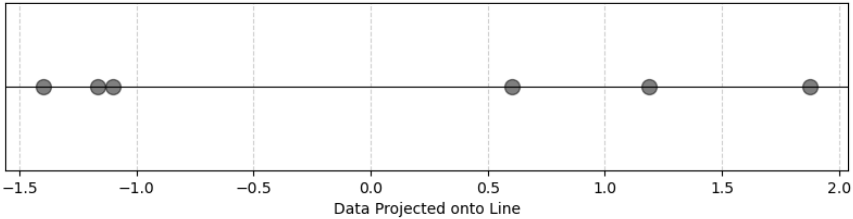
PCA - Project the Data

```
x_proj = []
Q = np.array([0.70710678, 0.70710678])
for x in observations:
    x_proj.append(x @ Q)

x_proj
```

```
[-1.1653394311885341,
 -1.1008549371158458,
 0.6015024779757782,
 1.185930382924984,
 -1.3957805284537876,
 1.874542035857405]
```

PCA - Project the Data



Here you see that we preserved the idea that the points were clearly separated in space even though we reduced the dimensions!

PCA - Summary

Principal component analysis leverages the eigenvalues and eigenvectors of the covariance matrix to find directions along which we can project our data to reduce the dimension.

PCA - Python Code - Automatic PCA

We can use tools from sklearn to automate this process!

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

PCA - Python Code - Automatic PCA

- Reload the data to start from scratch

```
filename='https://joannabieri.com/mathdatascience/data/PCAsimp
DF = pd.read_csv(filename)
```

- Normalize the data

```
# Z-score normalization
scalar = StandardScaler()
inputs = scalar.fit_transform(DF[features])
```

PCA - Python Code - Automatic PCA

- Create the model

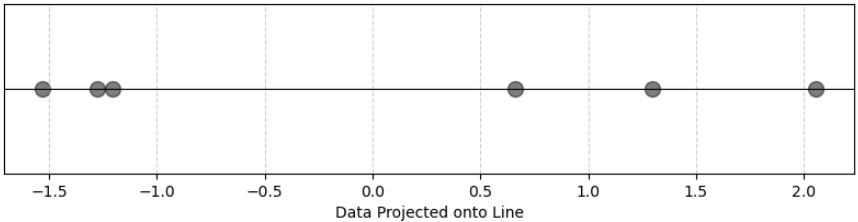
```
x_proj = model.fit_transform(inputs)
```

- Find eigenvalues (fit) and project the data (transform)

```
x_proj = model.fit_transform(inputs)
```

- Plot the result

PCA - Python Code - Automatic PCA



PCA - Python Code - Automatic PCA

You can look at eigenvectors used

```
model.components_
```

```
array([[0.70710678, 0.70710678]])
```

You can see the eigenvalue

```
model.explained_variance_
```

```
array([2.35205876])
```

You can ask about how much variance is explained by the projected data

```
model.explained_variance_ratio_
```

```
array([0.98002448])
```

Applied Example

Next we will consider a famous data set, the Diagnostic Wisconsin Breast Cancer Database.

https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic

In this data set is information about diagnostic images of breast masses along with labels telling us if the mass was cancerous or not. It is a famous data set used to learn about binary classification and PCA.

Load the data

	mean radius	mean texture	mean perimeter	mean area	mean smo
0	17.99	10.38	122.80	1001.0	0.11840
1	20.57	17.77	132.90	1326.0	0.08474
2	19.69	21.25	130.00	1203.0	0.10960
3	11.42	20.38	77.58	386.1	0.14250
4	20.29	14.34	135.10	1297.0	0.10030
...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100
565	20.13	28.25	131.20	1261.0	0.09780
566	16.60	28.08	108.30	858.1	0.08455
567	20.60	29.33	140.10	1265.0	0.11780
568	7.76	24.54	47.92	181.0	0.05263

Load the data

This data has 30 features and 1 target. The features are the first 30 column labels and they include things like

```
'mean radius', 'mean texture', 'mean perimeter', 'mean area',
   'mean smoothness', 'mean compactness', 'mean concavity',
   'mean concave points', 'mean symmetry' ...
```

the target is a column with a 1 or 0 saying whether the doctor found something that was malignant.

Visualization?

How can we visualize these numbers? Each observation (and there are 569 observations) consists of 30 features.

Visualization?

Look at just one observation

mean radius	17.990000
mean texture	10.380000
mean perimeter	122.800000
mean area	1001.000000
mean smoothness	0.118400
mean compactness	0.277600
mean concavity	0.300100
mean concave points	0.147100
mean symmetry	0.241900
mean fractal dimension	0.078710
radius error	1.095000
texture error	0.905300
perimeter error	8.589000
area error	153.400000
smoothness error	0.006399

Visualization?

This is why it is hard to visualize high dimensional data.

Is there something in the data (variance) that can tell me something about whether or not we see cancer in a patient?

Visualization?

If we want to actually visualize on a graph this we need to reduce it to 3 or fewer dimensions. This is not always successful for visualization even if PCA works. But it is usually worth a try!

Visualization?

Here is what we will do:

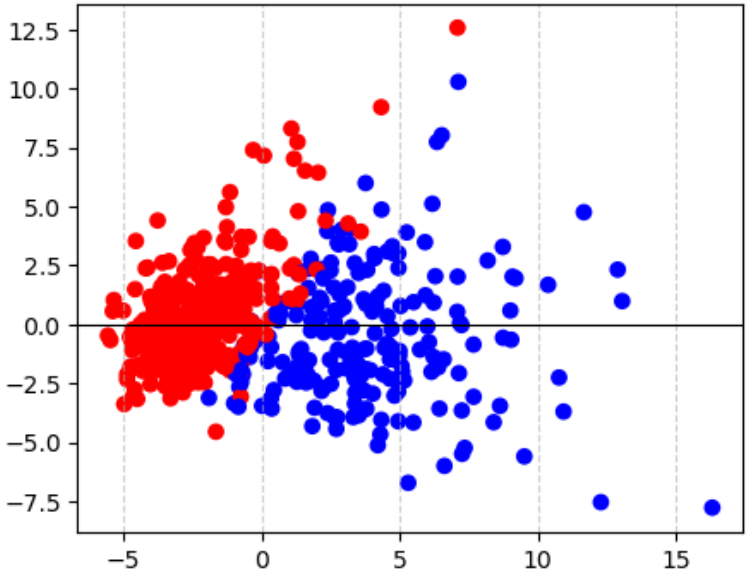
1. Take the features and normalize them
2. Run them through PCA and try to project them onto the 2d plane
3. Plot the resulting vectors (scatter plot)
4. The color the points by the label (malignant)
5. Ask if we see a pattern.

NOTE We don't need any labels to be able to do PCA, we just add them at the end to make our graph pretty!

PCA CODE

Look in the student notebook for the PCA code for this example!

Projection



Results

Lets look at the eigenvectors, what shape should they take?

Results

```
Eigenvalues:
[0.21890244370000278, 0.10372457821570166, 0.22753729300562497
[-0.23385713174765568, -0.05970608828045006, -0.21518136139490
```

Results

Let's look at the explained variance. What do these numbers mean?

```
Explained Variance: [0.44272026 0.18971182]
```

Summary

After doing PCA, we see that when we project the data from 30 dimensions onto the 2D plane and the color by the label, there are two fairly clear clusters that distinguish between malignant and not. This is good news if we hope to create a binary classifier!

We have 2 eigenvectors each of lenth 30. The the variance that is explained by the two principal components is $0.44272026 + 0.18971182 = 0.63243208$, so more than 60% of the variance in the data is described by these two components!

You Try

Our next very famous data set is the Iris data. This is a set of 150 observations of irises (flowers) along with measurements describing the flower. There are three types of flowers in the data set that have been assigned colors that you can use in the final graph:

- setosa = navy
- versicolor = turquoise
- virginica = darkorange

You Try

1. Load the data set and make a list of the features. How many features are there? How many labels?
2. Alter the code that we used for the breast cancer data so that it applies to the iris data (remember we changed the name DF –> DF_iris). This should result in a plot of your features projected onto the plane. You will need to change the color line to:

   ```
   color = DF_iris['color']
   ```

3. Produce the explained variance ratio. What does this tell you?
4. Summarize what you find in your results.

You Try

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width
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
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8