Unsupervised Learning

- No targets
- Why use it ?
    - Understand your features
    - Better use of features in supervised models

Plan

- Principal Components
    - Highly popular model for dimensionality reduction
- Clustering
    - K-means to cluster samples
    - Hierarchical clustering
- Recommender systems
    - Netflix prize
    - Pseudo SVD

Alternate basis

We can find an *alternate* set of n basis vectors of length n

$$\tilde{\mathbf{v}}_{(1)}, \ldots, \tilde{\mathbf{v}}_{(n)}$$

and translate $\mathbf{x}^{(\mathbf{i})}$ into coordinates $\tilde{\mathbf{x}}^{(\mathbf{i})}$ in the alternate basis

$$\tilde{\mathbf{x}}^{(\mathbf{i})} = \sum_{j=1}^{n} \tilde{\mathbf{x}}_j^{(\mathbf{i})} * \tilde{\mathbf{v}}_{(j)}$$

Principal Components Analysis (PCA) is a method for finding an alternate basis $\tilde{\mathbf{v}}_{(1)}, \ldots, \tilde{\mathbf{v}}_{(n)}$

- $\tilde{\mathbf{v}}_{(j)}$ is called *Principal Component j*
- That are mutually orthogonal
$$\tilde{\mathbf{v}}_{(j)} \cdot \tilde{\mathbf{v}}_{(j')} = 0, \text{ for } j \neq j'$$
- $\tilde{\mathbf{v}}_{(j)}$ has more variation than $\tilde{\mathbf{v}}_{(j')}$ for $j < j'$

The number of basis vectors in the original and alternate bases is both n.

Suppose we reduced the number of alternate basis vectors to $r \leq n$.

- We set $\tilde{\mathbf{x}}_j^{(i)} = 0$ for $j > r$

This is the *reduced dimension* approximation of $\mathbf{x}^{(i)}$.

$$\tilde{\mathbf{x}}^{(i)} = \sum_{j=1}^{r} \tilde{\mathbf{x}}_j^{(i)} * \tilde{\mathbf{v}}_{(j)}$$

Since the basis vectors are ordered such that $\tilde{\mathbf{v}}_{(j)}$ captures more variation than $\tilde{\mathbf{v}}_{(j')}$ for $j < j'$

- Dropping the alternate bases of higher index loss minimal information

PCA is the process of

- Finding alternate bases $\tilde{\mathbf{v}}$
- The alternate bases capture correlation among original features $\mathbf{x}$
- Projecting $\mathbf{x}^{(i)}$ onto the alternate basis $\tilde{\mathbf{v}}$ to obtain transformed vector $\tilde{\mathbf{x}}^{(i)}$ of synthetic features
- Choosing an r so that $\tilde{\mathbf{x}}^{(i)}$ is of dimension $r \leq n$

Return to parent notebook
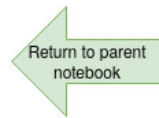
What is PCA

- A way to achieve dimensionality reduction
- Through the interdependence of features

As we mentioned: one use of dimensionality reduction is to find clusters of examples.

- One important difference from other methods for finding clusters
- The Decision Tree associates a cluster of examples with each node of the tree
- But the process of defining the clusters is guided by the **targets**
- Which are not present in Unsupervised Learning.

PCA: High Level

TL;DR

- PCA is a technique for creating "synthetic features" from the original set of features
- The synthetic features may better reveal relationships among original features
- May be able to use reduced set of synthetic features (dimensionality reduction)
- Synthetic features as a means of clustering samples
- **All features need (and will be assumed to be) centered: zero mean**
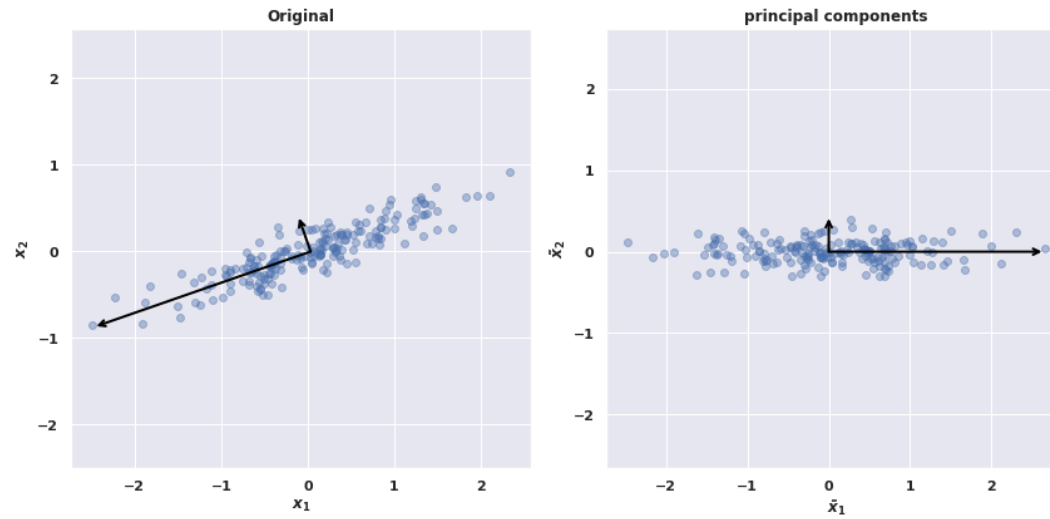- PCA is **very scale sensitive**; often normalize each feature to put on same scale

The key ideas behind PCA:

- Synthetic features are more like "concepts" than simple attributes
    - Commonality of purpose rather than surface similarity
    - e.g., the "factors" in the equity factor model example
- The synthetic features are mutually *independent* (uncorrelated)
- A transformation of examples from a basis of original features to a basis of synthetic features
- Order of "importance" of synthetic features
    - Facilitates dimensionality reduction by dropping "less important" features

Preview

In one picture:

```
In [5]:  X = vp.create_data()

         vp.show_2D(X)
```

The points in the left and right plots are the same, except for the coordinate system.

- Left plot: coordinate system is the horizontal and vertical axes, as usual
  - Features $\mathbf{x}_1, \mathbf{x}_2$
- Right plot: coordinate system changed
  - So that the arrowed lines of the left plot
  - Become the horizontal and vertical axes of the right plot (rotate and flip)
  - Features $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2$

In the left plot, we can clearly see that the data set's features $\mathbf{x}_1$, $\mathbf{x}_2$ are correlated.

In the right plot: $\tilde{\mathbf{x}}_1$, $\tilde{\mathbf{x}}_2$ are

- Independent
- With $\tilde{\mathbf{x}}_1$ expressed greater variation

Note

- The long arrowed line in the left plot
- Moves in the negative direction
- And is "flipped" to move in the positive direction of the right plot
- So the examples are the same, but rotated and flipped
    - We can more clearly see that in the examples highlighted in red

In [6]:
```
vp.show_2D(X, points=X[ X[:,0] > 1.5 ])
```
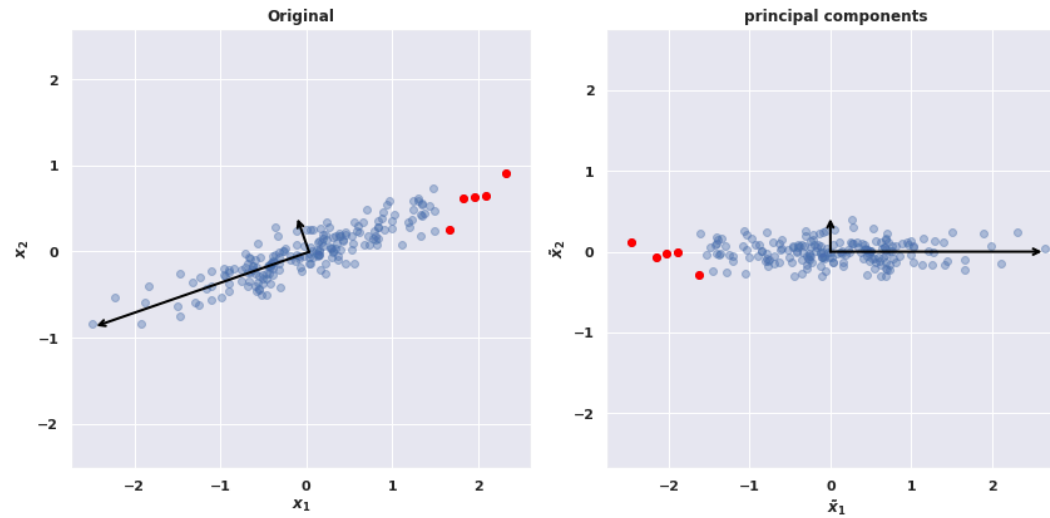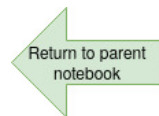
We can use matrix notation to summarize the process
$$\mathbf{X} = \tilde{\mathbf{X}}V^T$$

- The examples $\mathbf{X}$, expressed in the original basis
- Are the same as the examples
  - Expressed in alternate basis defined by V
  - With new feature values $\tilde{\mathbf{X}}$
  - When the number of new basis vectors is n, the number of original basis vectors

That is

- Examples $\mathbf{x}^{(i)}$ becomes $\tilde{\mathbf{x}}^{(i)}$

PCA via Matrix factorization

Our objective in this section is to show how to obtain $\tilde{\mathbf{X}}$ and V such that
$$\mathbf{X} = \tilde{\mathbf{X}}V^T$$

Decomposing $\mathbf{X}$ into a product (as above) is called *matrix factorization*

Some types of matrix factorization we'll mention

- Singular Value Decomposition
- Eigen Decomposition
- CUR Decomposition

Important note

We will assume that $\mathbf{X}$ has been **zero centered**: each feature value has had the mean value of the feature (across all examples) subtracted

Singular Value Decomposition (SVD) Factorization

Our first method for factoring $\mathbf{X}$ is called *Singular Value Decomposition (SVD)*.

Matrix X is factored into the product of 3 matrices:
$$\mathbf{X} = U\Sigma V^T$$

- $U: m \times n$, columns are orthogonal unit vectors
  - $UU^T = I$
- $\Sigma : n \times n$ diagonal matrix
  $$\mathrm{diag}(\Sigma) = [\sigma_1, \sigma_2, \ldots, \sigma_n]$$
- $V : n \times n$, columns are orthogonal unit vectors
  - $VV^T = I$

Moreover, the diagonal elements of Σ are in descending order of magnitude
$$\sigma_j > \sigma_{j'} \ \ \text{for } j < j'$$

Let V^T be the new basis vectors

- Since $VV^T = I$ these vectors are orthogonal

We need to find the synthetic features $\tilde{\mathbf{X}}$ relative to these bases such that
$$\mathbf{X} = \tilde{\mathbf{X}}V^T$$

A little math will show us how:

$$
\begin{aligned}
\mathbf{X} &= \tilde{\mathbf{X}}V^T & \\
U\Sigma V^T &= \tilde{\mathbf{X}}V^T & \text{factorization } \mathbf{X} = U\Sigma V^T \\
U\Sigma V^T V &= \tilde{\mathbf{X}}V^T V & \text{multiple both sides by } V \\
U\Sigma &= \tilde{\mathbf{X}} & \text{since } V^T V = I
\end{aligned}
$$

Thus SVD gives us both $\tilde{\mathbf{X}}$ and V^T as desired.

Moreover, $\tilde{\mathbf{X}}$ being equal to

$$\tilde{\mathbf{X}} = U\Sigma$$

has an interesting interpretation.

Recall that U is orthonormal

$$UU^T = I$$

so that its vectors are unit length

Since Σ is diagonal (all zero for non-diagonal elements)

- The diagonal of Σ, denoted
$$\text{diag}(\Sigma) = [\sigma_1, \sigma_2, \ldots, \sigma_n]$$
scales the rows of U
$$
\begin{aligned}
\tilde{\mathbf{X}}^{(\mathbf{i})} &= (U\Sigma)^{(\mathbf{i})} \\
&= U^{(\mathbf{i})} * \text{diag}(\Sigma) \\
&= [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_n^{(\mathbf{i})} * \sigma_n]
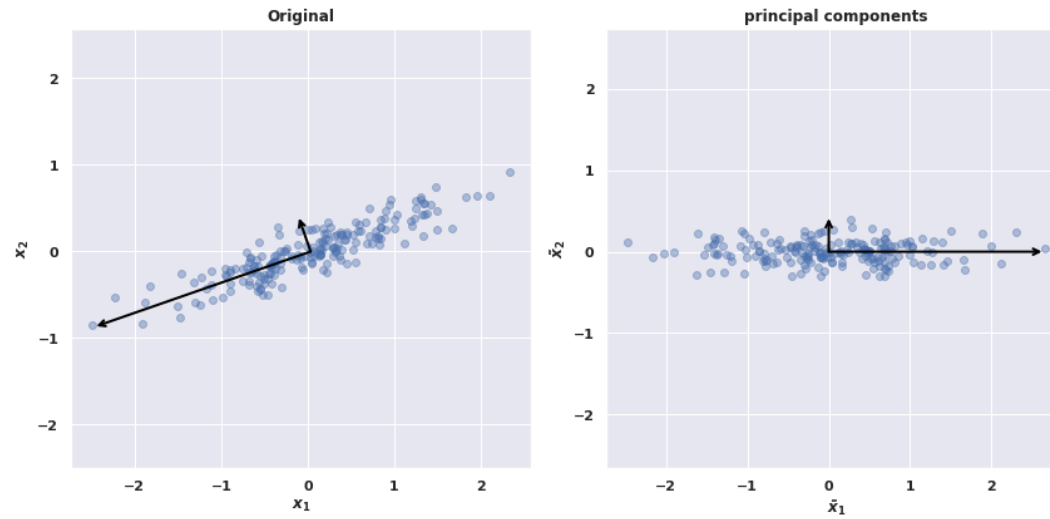\end{aligned}
$$

Thus

- U can be thought of as a "standardized" version of features $\tilde{\mathbf{X}}$
    - Unit standard deviation
- $\tilde{\mathbf{X}} = U\Sigma$ is the non-standardized features

A picture may clarify the distinction between the standardized and non-standardized $\tilde{\mathbf{X}}$.

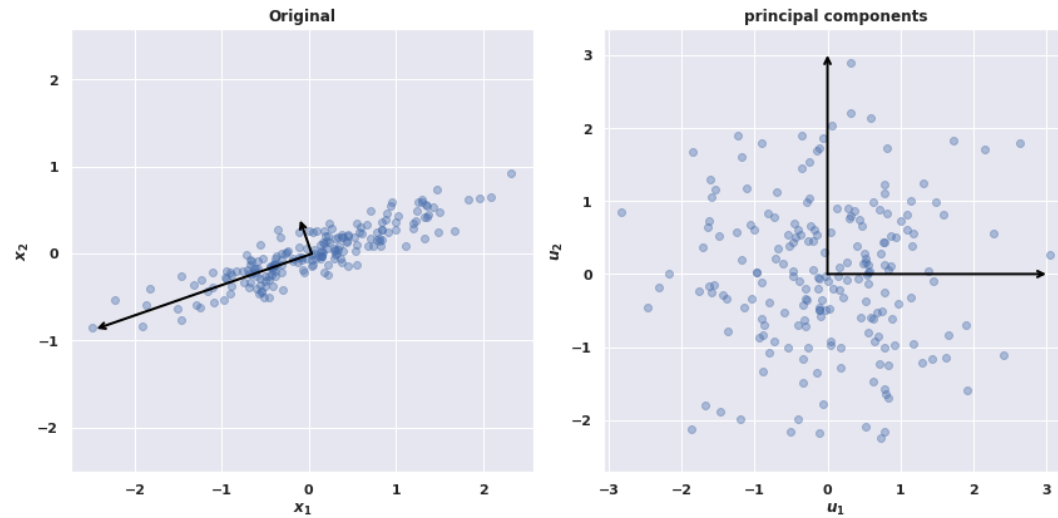Here is the non-standardized $\tilde{\mathbf{X}}$ that we've seen previously

In [7]:
```python
X = vp.create_data()

vp.show_2D(X)
```

And here is the standardized plot of U

- The length of each basis vector is 1
- Rather than σ_j
- By stretching each component by σ_j we recover the non-standardized plot

```
In [8]: vp.show_2D(X, whiten=True)
```

So

- The synthetic features $\tilde{\mathbf{X}}$
- Are "standardized" synthetic features U
- Scaled by $\mathrm{diag}(\Sigma)$

- We can see that, in the standardized coordinates: the feature values are cloud-like, independent
- The magnitude of the diagonal elements σ_i
    - Is related to the variation (how far the spread) of non-standardized synthetic feature i
    - Which we will associate with the "importance" of the non-standardized feature

Dimensionality reduction

Thus far we have exactly replicated $\mathbf{X}$ via new bases V^T
$$\mathbf{X} = \tilde{\mathbf{X}} V^T$$

$\tilde{\mathbf{X}}$ is the same dimensions as $\mathbf{X}$, so each example is of length n in both the original and alternate representation.

We will now change $\tilde{\mathbf{X}}$ to $(m \times r)$ for $r \leq n$.

That is: the alternate representation may be of reduced dimension.

Recall that

$$
\begin{aligned}
\mathbf{\tilde{X}}^{(\mathbf{i})} &= (U\Sigma)^{(\mathbf{i})} \\
&= U^{(\mathbf{i})} * \mathrm{diag}(\Sigma) \\
&= [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_n^{(\mathbf{i})} * \sigma_n]
\end{aligned}
$$

By setting

$$
\sigma_j = 0, \text{ for all } j > r
$$

we zero out all features with index exceeding r

$$
\begin{aligned}
\mathbf{\tilde{X}}'^{(\mathbf{i})} &= (U\Sigma)^{(\mathbf{i})} \\
&= [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_r^{(\mathbf{i})} * \sigma_r, \mathbf{0}, \ldots \mathbf{0}]
\end{aligned}
$$

The dimensions of $\mathbf{\tilde{X}}'^{(\mathbf{i})}$ is effectively reduced from n to $r \leq n$.

Zeroing out the diagonal elements of Σ with index $j > r$ makes the values in

- The columns of U with index $j > r$
- The rows of (V^T) with index $j > r$

irrelevant.

We can therefore write

$$\mathbf{X}' \approx \mathbf{X}$$

where

$$\mathbf{X}' = U'\Sigma'(V^T)'$$

where

- $\mathbf{X}'$ is $(m$
    $\times n$
    $)$
- U' is $(m$
    $\times r$
    $)$
- Σ' is $(r$
    $\times r$
    $)$
- $(V^T$ is $(r$

$\mathbf{X}$ and $\mathbf{X}'$ have the *same* dimensions, but the values in $\mathbf{X}'$ can only *approximate* the values in $\mathbf{X}$.

The advantage is that $\tilde{\mathbf{X}}'$ is of lower dimension.

Best lower rank approximation of $\mathbf{X}$

We could have reduced the dimension of $\tilde{\mathbf{X}}'$ by dropping *any* set of $(n - r)$ columns.

Let D denote the set of size $(n - r)$ containing the indexes of the columns we choose to drop.

Is there a particular reason for dropping the columns
$$D = \{j \mid j > r\}$$

To answer the question, we first define the *error* of the approximation $\mathbf{X}'$ relative to the true $\mathbf{X}$

$$||\mathbf{X}' - \mathbf{X}||_2 = \sum_{i,j} \left(\mathbf{X'}_j^{(\mathbf{i})} - \mathbf{X}_j^{(\mathbf{i})} \right)^2$$

The above is called the Froebenius Norm (and looks like MSE in form).

The "best" set of columns to drop (from Σ) are the one resulting in the *lowest* error.

Recall that

$$\tilde{\mathbf{X}}^{(\mathbf{i})} \quad = \quad [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \dots, U_n^{(\mathbf{i})} * \sigma_n]$$

and

$$\mathbf{X}^{(\mathbf{i})} = \tilde{\mathbf{X}}^{(\mathbf{i})} V^T$$

So

$$\mathbf{X}_j^{(\mathbf{i})} \quad = \quad [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \dots, U_n^{(\mathbf{i})} * \sigma_n] \cdot (V^T)_j \qquad \text{where } (V^T)_j \text{ is column}$$

$$= \quad \sum_{k=1}^n U_k^{(\mathbf{i})} * \sigma_k * (V^T)_j^{(k)} \qquad \qquad \text{multiply row } i \text{ of U by c}$$

$$= \quad \sum_{k=1}^n \sigma_k * (U_k^{(\mathbf{i})} * (V^T)_j^{(k)})$$

The approximation error in $\mathbf{X}_j^{(\mathbf{i})}$ induced by dropping the columns in D is

$$(\mathbf{X}_j^{(\mathbf{i})} - \mathbf{X}_j'^{(\mathbf{i})})^2 \quad = \quad \left(\sum_{k \in D} \sigma_k * (U_k^{(\mathbf{i})} * (V^T)_j^{(k)}) \right)^2$$

Because the diagonal elements of Σ are in decreasing order of magnitude
$$\sigma_j > \sigma_{j'} \ \text{ for } j < j'$$

choosing D to be
$$D = \{j \mid j > r\}$$
results in dropping terms $U_k^{(\mathbf{i})} * (V^T)_j^{(k)}$ that are scaled by the $(n - r)$ *smallest* values o
σ_k.

Although this is not mathematically precise, hopefully this provides some intuition as to why choosing D this way is a good idea.

Aside

It will turn out that

$$\left(\sum_{k \in D} \sigma_k * (U_k^{(\mathbf{i})} * (V^T)_j^{(k)}) \right)^2$$
$$= \sum_{k \in D} \sigma_k^2$$

which makes the intuitive argument precise.

How many dimensions to keep ?

Since the diagonal elements of Σ are ordered

- We can compute a cumulative, normalized sum s of σ^2

$$s_j = \frac{\sum_{j'=1}^{j} \sigma_{j'}^2}{\sum_{j'=1}^{n} \sigma_{j'}^2}$$

- Such that

$$s_n = 1$$

- So that s_j is the *fraction* of total variance associated with the first j components

We can then chose the number $r \leq n$ of reduced dimensions
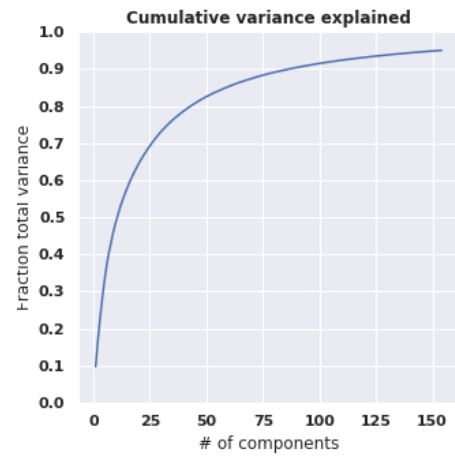
- As the r such that $s_r \geq T$
- Where T is a threshold fraction for explained variance
- For example
$$T = 95\%$$

A detailed example of PCA performed on the MNIST digits follows in a later section.

The cumulative variance, as a function of number of features kept, looks like:

PCA: MNIST digits, cumulative variance

Cumulative variance explained

From the chart: we can capture 95% of the cumulative variance using roughly 150 synthetic features.

Return to parent
notebook
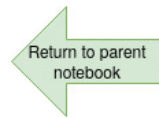
The inverse transformation

We have shown how to transform original features $\mathbf{X}$ to synthetic features $\tilde{\mathbf{X}}$.

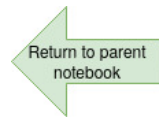How about inverting the transformation: recover $\mathbf{X}$ from $\tilde{\mathbf{X}}$?

Since

$$\begin{array}{rcll}
\mathbf{X} & = & \tilde{\mathbf{X}}V^T & \text{definition} \\
\mathbf{X}V & = & \tilde{\mathbf{X}}V^TV & \text{multiply both sides by } V \\
\mathbf{X}V & = & \tilde{\mathbf{X}} & \text{since } V^TV = I
\end{array}$$

So

- V transforms from original features $\mathbf{X}$ to synthetic feature $\tilde{\mathbf{X}}$
- V^T transforms synthetic features $\tilde{\mathbf{X}}$ to original features $\mathbf{X}$

Return to parent
notebook

PCA in `sklearn`

Principal Components Analysis is a *transformation* in `sklearn`

As with all transformations in `sklearn`

- it must be *fit* with the training data
- once fit, the transformation may be *applied*
    - to the features of the training data, and used to train a model, e.g., a Classifier
    - to out of sample examples, to make predictions

Relating this to our terminology:

- fit to the training data

  - This computes U, Σ, V)
  - These computed values are used to transform examples

- transform examples

  - training data $\mathbf{X}$ to reduced dimension alternate basis: $\tilde{\mathbf{X}}'$
  - out of sample test example $\mathbf{x}_{\text{test}}$ to reduced dimension alternate basis: $\tilde{\mathbf{x}}'_{\text{test}}$
- Use the transformed examples, e.g., to build and evaluate the predictive model
  - e.g., use the transformed training data as input to a predictive model `model`
  - e.g., use the transformed text examples to evaluate Performance Metric on predictive model `model`

Here is some pseudo-code to illustrate

```python
# Split example <X, y> into training and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
_state=42)

# Instantiate the PCA object: reduce dimension from X.shape[-1] to n_components
pca = PCA(n_components=n_components)

# Fit PCA to training examples
pca.fit(X_train)

# Transform the training examples
X_tilde = pca.transform(X_train)

# Use the transformed training examples to fit a model for prediction
model = ...
model.fit(X_tilde, y_train)

# Transform the test examples
X_test_tilde = pca.transform(X_test)
```

Example: Reconstructing $\mathbf{x}$ from $\tilde{\mathbf{x}}$ and the principal components

It may be helpful to visualize

- The transformation from example features $\mathbf{x}^{(i)}$
- To synthetic features $\tilde{\mathbf{x}}^{(i)}$

We will use a subset of the "smaller digits" (8×8) data

```
In [9]:  subset1 = [ 0, 4, 7, 9 ]
         rh_digits = unsupervised_helper.Reconstruct_Helper( subset=[])
         rh_digits.create_data_digits(subset=subset1)
```
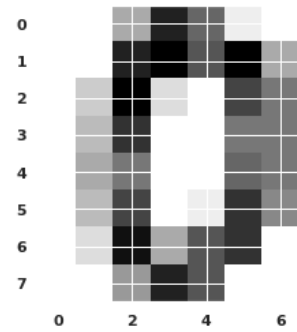
```
In [10]: n_components = 8
         _= rh_digits.fit(n_components=n_components)
```

```python
# Which example to show
data_idx = 0
fig0, ax0, figm, axm, figc, axc = rh_digits.show_data_comp(data_idx=data_idx)

plt.close(fig0)
plt.close(figm)
plt.close(figc)
```
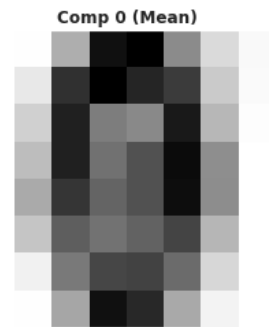
Here is one example:

In [12]: fig0

Out[12]:

And here is the "mean" of $\mathbf{X}$

- Recall: we assume $\mathbf{X}$ has been zero-centered
- So the mean is subtracted before performing PCA
- Which means it has to be *added* to the reconstructed image
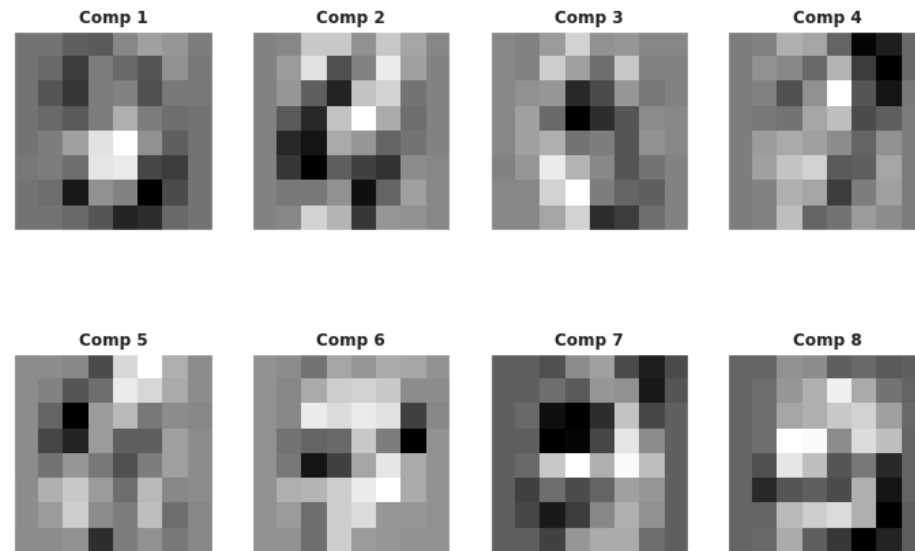
In [13]: figm

Out[13]:

Comp 0 (Mean)

And here are the Principal Components (new bases)

- We performed PCA with a reduced number of components ($r < n$)
- There are n_components such basis vector
- *Each* component is of length n, but there are only $r < n$ components

In [14]: figc

Out[14]:

Comp 1	Comp 2	Comp 3	Comp 4

Comp 5	Comp 6	Comp 7	Comp 8

It's not necessarily easy to interpret the components, particularly when n is large

- Component 1 *might* be the "concept" corresponding to the digit 0
- Component 4 *might* be the "concept" corresponding to the digit 4
- The other components *might* be *partial* shape concept, rather than entire digits

Let's progressively examine the reconstruction using an increasing r (number of synthetic features)

$$\mathbf{x}^{(\mathbf{i})} = \sum_{j=1}^{r} \tilde{\mathbf{x}}_{j}^{(\mathbf{i})} * (V^{T})^{(j)}$$

It may be helpful to remind ourselves of the shapes of each element in the equation

- $\mathbf{x}^{(i)}$ is of length n
- The components V^T are $(r \times n)$
    - Each component is of length n
    - There are $r < n$ components (e.g., $r = $ n_components)
- $\tilde{\mathbf{x}}^{(i)}$ is of length r

Note

- We treat the "mean $\mathbf{X}$ as component 0
- With weight 1
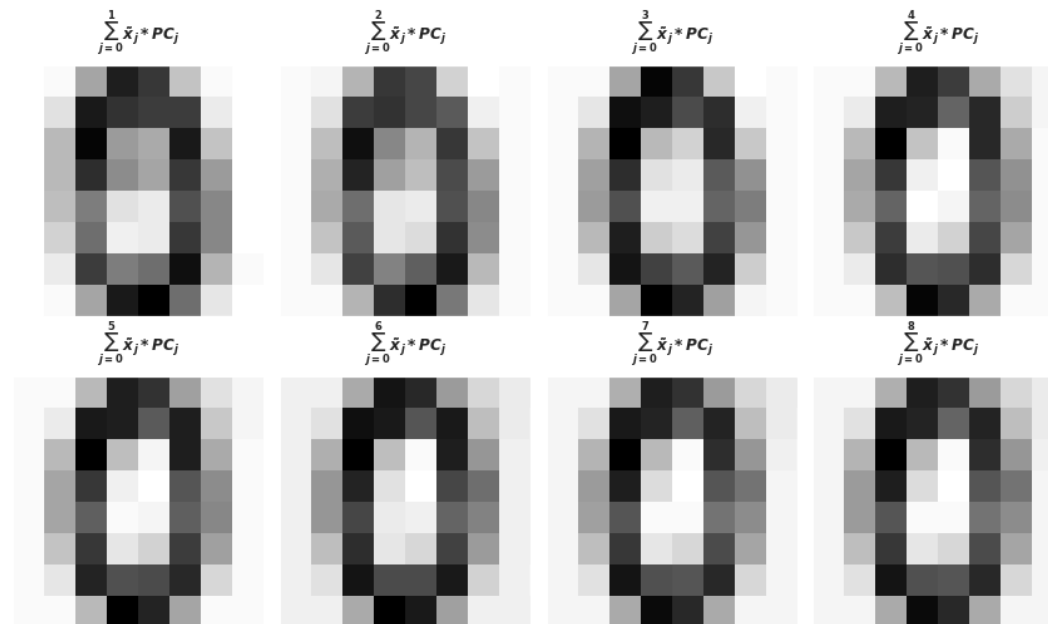- So that it is "added back" into the reconstruction

So we construct an approximation of $\mathbf{x}^{(\mathbf{i})}$

- By adding weighted components $(V^T)^{(j)}$, each of length n
- The weight associated with component j is $\tilde{\mathbf{x}}_j^{(\mathbf{i})}$
- So the sum is of length n

```
In [15]: figi, axsi, fig_comp, axs_comp, x_tilde, error = rh_digits.show_recon(data_idx=d
         ata_idx)
         plt.close(figi)
         plt.close(fig_comp)
```

In [16]: `figi`

Out[16]:

$\sum\limits_{j=0}^{1} \tilde{x}_j * PC_j$ $\sum\limits_{j=0}^{2} \tilde{x}_j * PC_j$ $\sum\limits_{j=0}^{3} \tilde{x}_j * PC_j$ $\sum\limits_{j=0}^{4} \tilde{x}_j * PC_j$

$\sum\limits_{j=0}^{5} \tilde{x}_j * PC_j$ $\sum\limits_{j=0}^{6} \tilde{x}_j * PC_j$ $\sum\limits_{j=0}^{7} \tilde{x}_j * PC_j$ $\sum\limits_{j=0}^{8} \tilde{x}_j * PC_j$

You can see that the approximation using just the first component (and the mean)

- Is already a good approximation of $\mathbf{x}^{(i)}$
- Somewhat confirming our *guess* that component 1 represents the concept 0

We can confirm this by looking at $\tilde{\mathbf{x}}^{(i)}$ numerically:

```
In [17]:  print("x tilde = ", x_tilde)

          arg_max = np.argmax(x_tilde)
          print("Largest feature at index {idx:d}".format(idx=arg_max+1))
```

```
x tilde =  [ 18.94186593   5.09554009 -11.11744189   6.39736343  -0.9635152
   3.24065443   2.53258896   0.05016427]
Largest feature at index 1
```

As you can see, the magnitude of $\tilde{\mathbf{x}}_1^{(i)}$ is the largest among $[\tilde{\mathbf{x}}_j^{(i)} \mid 1 \leq j \leq r]$

In fact, we might try to confirm our intuition

- By examining $\tilde{\mathbf{x}}^{(i')}$ for all i' where $\mathbf{y}^{(i')} = 0$ (assuming we have targets/labels)

```
In [18]:  # Get X tilde and the targets
          Xtilde = rh_digits.dataProj
          y = rh_digits.targets

          # Filter to identify examples where target is equal to digit
          digit = 0
          mask = (y == digit)
          Xtilde_digit = Xtilde[mask]
```

```python
print("x tilde, when y=0:")

for i in range(0,10):
    print( [ "{x:3.2f}".format(x=x_tilde_j) for x_tilde_j in Xtilde_digit[i] ])
```

```
x tilde, when y=0:
['18.94', '5.10', '-11.12', '6.40', '-0.96', '3.24', '2.53', '0.05']
['10.94', '11.59', '-8.82', '8.34', '-6.50', '4.20', '4.21', '-4.46']
['15.16', '8.46', '-9.70', '2.85', '-3.39', '-5.12', '-3.82', '-7.20']
['21.68', '9.93', '-12.65', '3.27', '1.30', '4.79', '-0.86', '-3.95']
['13.36', '8.83', '-11.29', '4.24', '-0.36', '-1.21', '0.98', '-12.04']
['19.57', '7.25', '-9.87', '-3.35', '-1.75', '7.12', '4.04', '0.31']
['20.21', '9.81', '-7.82', '-2.10', '-2.18', '-1.37', '-0.84', '3.75']
['10.75', '12.33', '-9.70', '-1.01', '-4.81', '-7.05', '-4.92', '-12.91']
['17.96', '12.42', '-5.09', '-5.08', '2.48', '-8.19', '-2.90', '-11.08']
['22.48', '1.51', '-7.24', '-10.33', '3.19', '3.57', '0.18', '4.98']
```
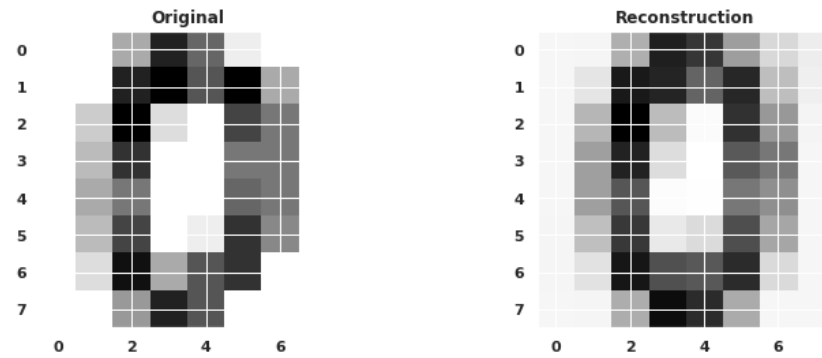
As you can see

- For examples $\mathbf{x}^{(i')}$ where $\mathbf{y}^{(i')} = 0$
- $\tilde{\mathbf{x}}_1^{(i')}$ is the largest value in $\tilde{\mathbf{x}}^{(i')}$, for all i' that we examined

Here is a comparison of the original $\mathbf{x}^{(i)}$ and its reconstructed approximation

```
In [20]: fig_comp
```

Out[20]:

One advantage of reduced dimensionality

- we can plot the higher dimensional n-vectors in lower dimensional (r) space
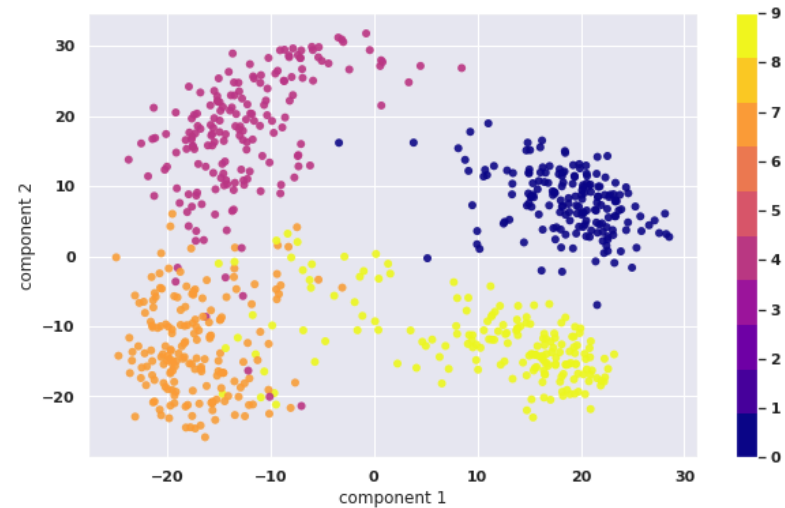
Doing so allows us to see

- how "dense" the r-space
- whether examples form clusters

But $r = $ `n_components` is too large in our case; let's just plot using the first two features of $\tilde{\mathbf{x}}$

```
vpt = unsupervised_helper.VanderPlas()

print("Number of examples: {n:d}".format(n=Xtilde.shape[0]))
vpt.digits_subset_show_clustering(Xtilde, y, save_file="/tmp/digits_subset_clust
er.jpg" )
```

Number of examples: 718

We can definitely see clusters in each of the 4 quadrants.

But interpreting what is in each cluster takes some guess work

- hypothesizing some source of "commonality" among the examples in the cluster
- verifying the hypothesis by examining the examples in the cluster

We can "cheat" in making the interpretation of the meaning of clusters in this case

- we have targets/labels available
- leading to a natural hypothesis: the clusters correspond to the *labels*

In general

- we do not have labels
- the clusters may be defined by non-labels
    - e.g., subsets of common features
- interpretation is more difficult

But, given the targets/labels:

- We can color the points according to their target
- We see that the 4 digits in the restricted examples cluster according to their features in $\tilde{\mathbf{x}}$
- Digit "0" is associated with (high $\tilde{\mathbf{x}}_1$, high $\tilde{\mathbf{x}}_2$)
- Digit "4" is associated with (low $\tilde{\mathbf{x}}_1$, high $\tilde{\mathbf{x}}_2$
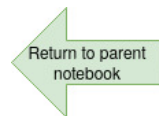
Dimensionality reduction:examples

MNIST example

In our introduction we illustrated representing MNIST digits

- With $r \approx 150$ synthetic features
- Rather than $n = 28 * 28 = 784$ original features.

Using the techniques illustrate for the "small digit subset" example above, you might try to interpret the components

- We had argued that "blocks of dark pixels" in each corner was a source of redundancy
    - Was such a concept discovered by PCA ? Is it more subtle ?

This section of code should be a playground for you to experiment and deepen your understanding of PCA.

Here we provide some helper code.

First, retrieve the full MNIST dataset (70K samples)

We had previously used only a fraction in order to make our demo faster.

```
In [22]: ush = unsupervised_helper.PCA_Helper()

X_mnist, y_mnist = ush.mnist_init()
```

Retrieving MNIST_784 from cache

```python
from sklearn.model_selection import train_test_split
X_mnist.shape, y_mnist.shape
X_mnist_train, X_mnist_test, y_mnist_train, y_mnist_test = train_test_split(X_mn
ist, y_mnist)
X_mnist_train.shape
```

Out[23]: ((70000, 784), (70000,))

Out[23]: (52500, 784)

Perform PCA.

```
In [24]: pca_mnist = ush.mnist_PCA(X_mnist_train)
```

```
In [25]:  pca_mnist.n_components_
          X_mnist_train_reduced = ush.transform(X_mnist_train, pca_mnist)
          X_mnist_train_reduced.shape
```

Out[25]: 154

Out[25]: (52500, 154)

Let's plot the cumulative variance as a function of number of synthetic features.

This can help us determine how many synthetic features to keep.

```
_ = ush.plot_cum_variance(pca_mnist)

variance_goal_pct = 95
features_for_goal = ush.num_components_for_cum_variance(pca_mnist, .01 * varianc
e_goal_pct)
print("To capture {f:d}% of variance we need {d:d} synthetic features.".format(f
=variance_goal_pct, d=features_for_goal))
```

To capture 95% of variance we need 154 synthetic features.
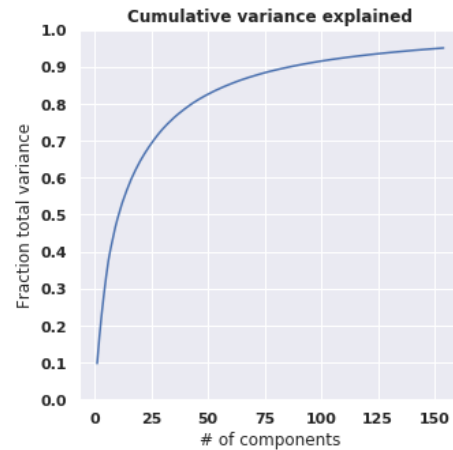
So we need only about 20% of the original 784 features to capture 95% of the variance.

We can invert the PCA transformation to go from synthetic feature space back to original features.

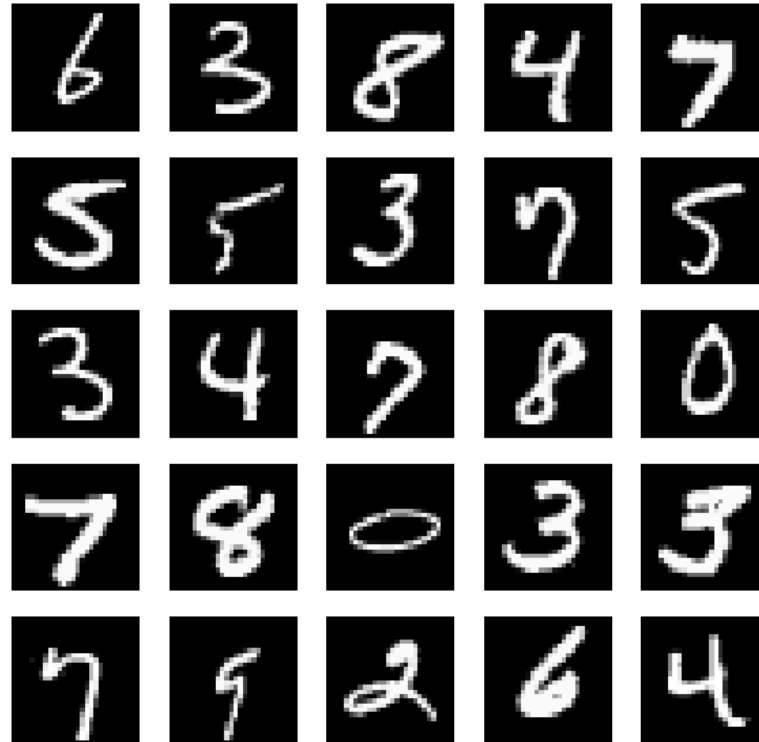That is, we can see what the digits look like when reconstructed from only 154 synthetic features.

First, let's look at the original:

```
X_mnist_train_reduced = ush.transform(X_mnist_train, pca_mnist)
X_mnist_train_reduced.shape

# Show the original dataset
ush.mnh.visualize(X_mnist_train, y_mnist_train)
```
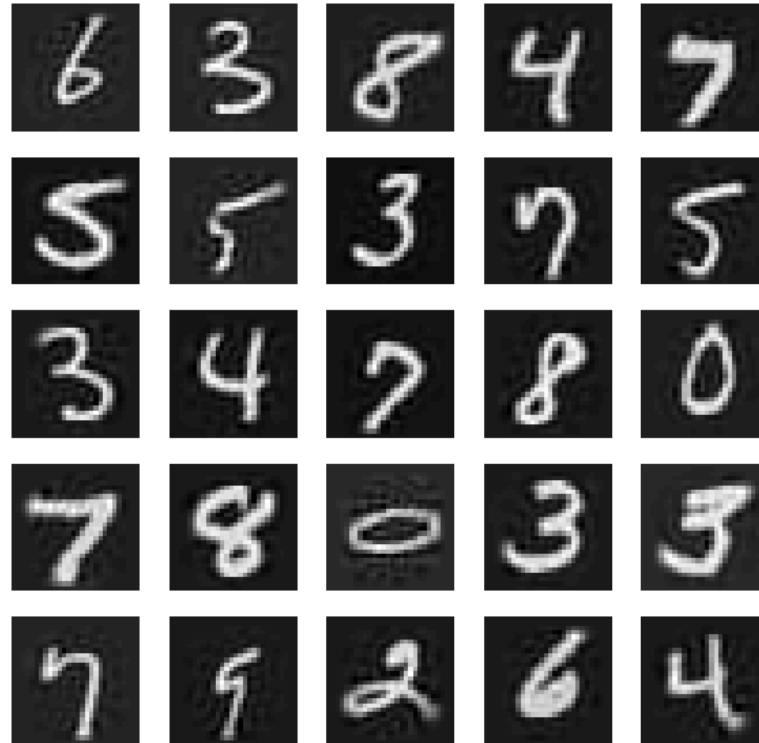
Out[27]: (52500, 154)

Out[27]:

Next, the reconstructed

```
In [28]: X_mnist_train_reconstruct = ush.inverse_transform(X_mnist_train_reduced, pca_mni
         st)
         ush.mnh.visualize(X_mnist_train_reconstruct, y_mnist_train)
```

Out[28]:

A little fuzzy, but pretty good.

Do the 10 digits form natural clusters (all images of the same digit in the same cluster) ?

Let's look at the training images when reduced to only 2 synthetic features.

```
In [29]:   _ = ush.mnist_plot_2D(X_mnist_train_reduced, y_mnist_train.astype(int))
```
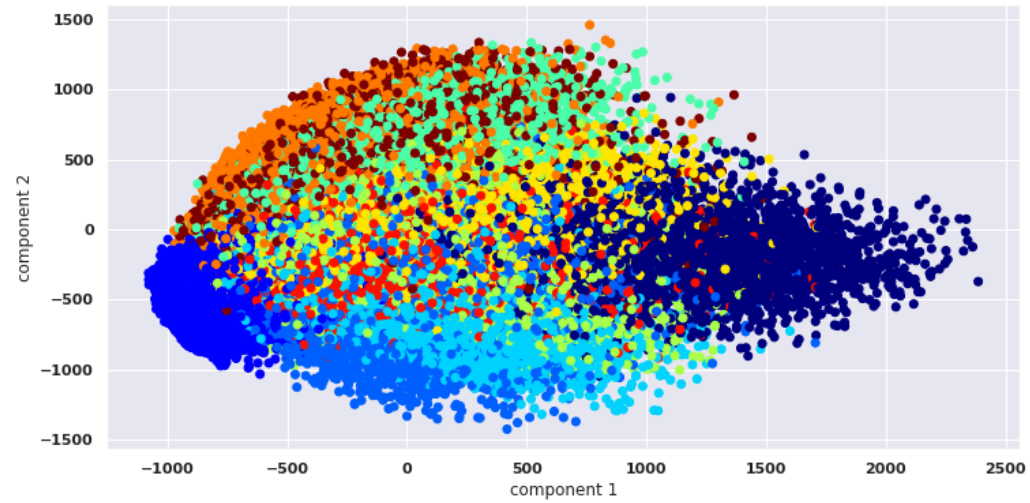
Each color is a different digit.

You can see that the clustering is far from perfect

- But also surprisingly good considering we're using only 2 out of 784 features

Let's see how much variance is captured by only the first two synthetic features.

```
In [30]:  cumvar_mnist = np.cumsum(pca_mnist.explained_variance_ratio_)
          first_comp =2
          cumvar_first = cumvar_mnist[first_comp-1]

          print("Cumulative variance of {d:d} PC's is {p:.2f}%, about {n:.1f} pixels".form
          at(
              d=first_comp, p=100 *cumvar_first, n=cumvar_first * X_mnist_train.shape[1]))
```

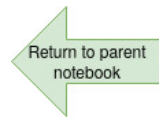Cumulative variance of 2 PC's is 16.90%, about 132.5 pixels

Is 17% good ? You bet !

With 784 original features (pixels)

- if each feature had equal importance, it would explain $1/784 = .12\%$ of the variance.

So the first synthetic feature captures the variance of 132 original features

- (assuming all were of equal importance).

PCA in Finance

PCA of yield curve

[Litterman Scheinkman (https://www.math.nyu.edu/faculty/avellane/Litterman1991.pdf)](https://www.math.nyu.edu/faculty/avellane/Litterman1991.pdf)

This is one of the most important papers (my opinion) in quantitative Fixed Income.

It allows us to hedge a large portfolio of bonds with a handful of instruments.

Before we show the result: why is this an important advance in Finance ?

- Imagine we had a large portfolio of bonds with many maturities.

- A common goal in Fixed Income Finance is to *immunize* (hedge) a portfolio to changes in the Yield Curve.

- A simple way to construct the hedge is to

  - Find the sensitivity of each bond in the portfolio to the $n = 14$ "benchmark" maturities
  - Sum (over bonds in the portfolio) the individual bond sensitivities
  - Minus 1 times resulting portfolio sensitivity is the hedge that minimizes the portfolio's exposure to Yield Curve changes

But there are transaction costs (and complexity) with $n = 14$ bonds in the hedge portfolio.

Can we do nearly as well with $n' < n$ hedge bonds ?

That's exactly what PCA is designed for: dimensionality reduction.

- In this case, reducing the number of "benchmark" hedge bonds
- With minimal impact on immunization goal

Let's get the history of Yield Curves and look at first few examples.

```
In [31]: ych = unsupervised_helper.YieldCurve_PCA()

         # Get the yield curve data
         data_yc = ych.create_data()
         data_yc.head()
```

Out[31]:

	1M	2M	3M	6M	1J	2J	3J	4J	5J	6J	7J	8J	9J	10J
1992-02-29	0.0961	0.09610	0.0961	0.0958	0.0898	0.0864	0.0849	0.0837	0.0826	0.0817	0.0810	0.0806	0.0803	0.0804
1992-03-31	0.0970	0.09700	0.0970	0.0969	0.0912	0.0889	0.0877	0.0864	0.0852	0.0841	0.0833	0.0827	0.0823	0.0823
1992-04-30	0.0975	0.09750	0.0975	0.0975	0.0920	0.0892	0.0877	0.0862	0.0848	0.0837	0.0828	0.0822	0.0817	0.0816
1992-05-31	0.0978	0.09785	0.0979	0.0979	0.0920	0.0889	0.0874	0.0860	0.0847	0.0836	0.0828	0.0821	0.0817	0.0815
1992-06-30	0.0974	0.09745	0.0975	0.0975	0.0931	0.0904	0.0889	0.0874	0.0860	0.0848	0.0839	0.0832	0.0827	0.0825

```
In [32]: data_yc.shape
```

Out[32]: (287, 14)

Each example (row) has 14 features: the yields for 14 maturity points on a given date.

Let's plot the history of yield curves

`ych.plot_YC(data_yc)`

```
/home/kjp/anaconda3/lib/python3.7/site-packages/pandas/plotting/_matplotlib/co
nverter.py:103: FutureWarning: Using an implicitly registered datetime convert
er for a matplotlib plotting method. The converter was registered by pandas on
import. Future versions of pandas will require you to explicitly register matp
lotlib converters.

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```
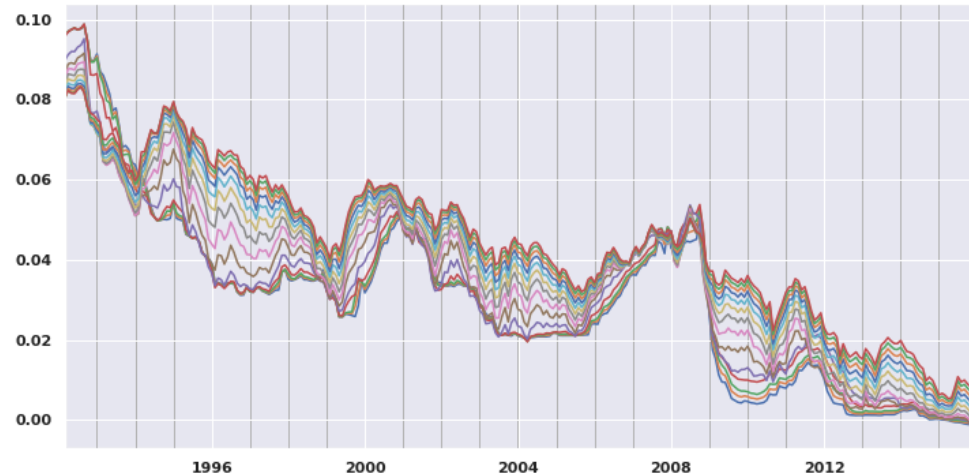
Let's perform PCA on the **changes** in Yield Curve

- Just like in Supervised Learning, we sometimes need to transform the data before fitting a model
- The features we feed to PCA are *yield changes* rather than yields

So $n = 14$ maturities, for m samples (many years of daily data)

How many bonds (i.e, what is the n') is "good enough" ?

The plot of cumulative variance explained, versus n' will give us an answer.

```
In [34]: pca_yc, df_pca_yc = ych.doPCA(data_yc, doDiff=True)
         _ = ych.plot_cum_variance(pca_yc)
```

Cumulative variance explained
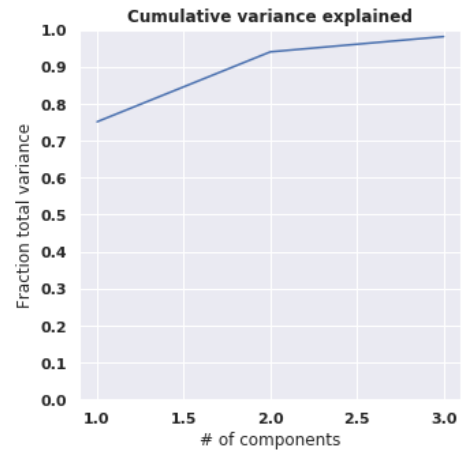
Wow !

Only $n' = 3$ synthetic features capture almost all the variance of the original $n = 14$ features !

It gets even better !

By examining the composition of the synthetic features, we can *interpret* what they are

Let's examine the effect of a 1 standard deviation move in each synthetic feature on original features.

We arrange the original features by maturity on the horizontal axis and plot the effect on the vertical.

The choice to arrange the horizontal axis by maturity is *very deliberate*, as we will see.

In [35]: `ych.plot_components(pca_yc)`

First 3 PCA components
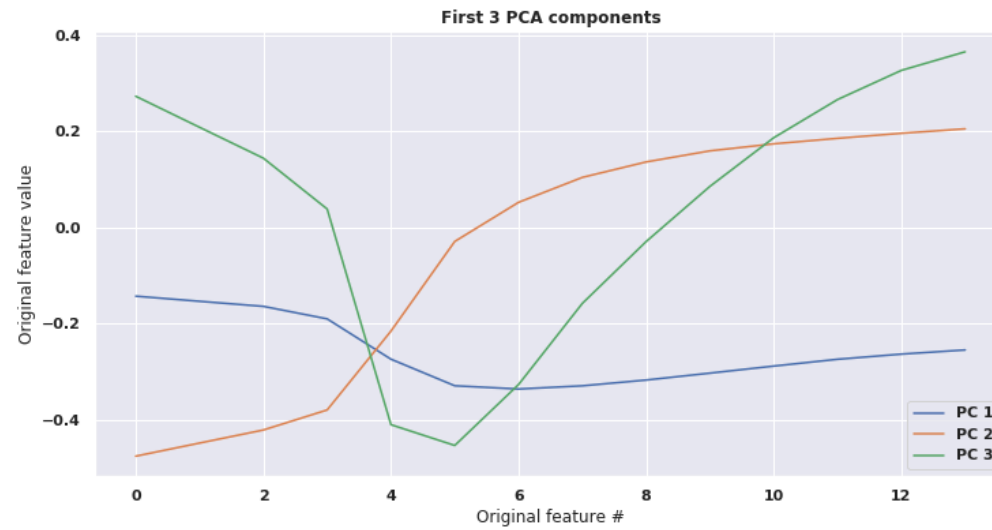
This is a very typical pattern in Finance:

- The first synthetic feature affects all original features with roughly equal effect
- Higher synthetic features often express a *dichotomy*
    - Positive effect on some original features
    - Negative effect on other original features

In our case the original features are yield changes.

A unit standard deviation value of synthetic feature j (PCj)

- $(j = 1)$: affects all original features (yield change) roughly equally
    - Corresponds to a parallel shift in the Yield Curve
- $(j = 2)$: shows a dichotomy (of yield changes) between near and far maturities
    - Corresponds to the slope of the Yield Curve changing
- $(j = 3)$: shows a dichotomy of yield changes of mid maturities versus extreme maturities
    - Corresponds to a twist in the Yield Curve about the 5 year maturity

Recall

The synthetic features are standardized, hence 1 unit is one standard deviation.

The Σ matrix re-scales from standard deviation to original feature space.

- So can't compare the levels on the vertical axis between synthetic features.
    - Since $\sigma_1 > \sigma_2$
    - The absolute effect of synthetic feature 1 is greater than that of synthetic feature 2
    - For a 1 standard deviation move in each.

We started with a complicated example (Yield Curve with $n = 14$ features)

- can almost completely explain (changes in the Yield Curve) with 3 intuitive market changes
    - Parallel Shift up/down of all maturities
    - Long end versus short end changes (slope)
    - Twist at intermediate maturity

That's interesting from a Machine Learning perspective but important for Finance:

- It shows show to construct efficient Hedge Portfolios.

Suppose we have a Target Portfolio consisting of long positions in many bonds, with many maturities.

Our goal is to construct another portfolio (the Hedge Portfolio)

- Consisting of a small number of bonds
- With the *same* sensitivity to Yield Curve changes as the Target Portfolio

By combining a long position in the Target Portfolio with a short position in the Hedge Portfolio

- The resulting Net Portfolio is immunized (hedged) to changes in the Yield Curve

If we can re-express our Target Portfolio in terms of synthetic features

- We can construct the Hedging Portfolio as one consisting only of 3 synthetic bonds
- In quantities that mimic the sensitivity of the Target Portfolio to the first 3 PCs.
    - Since the PCs are *independent*, this is easy
    - Normally: you would need to solve
        - a system of 3 equations in 3 unknowns (size of position in each synthetic bond)

The catch is that each synthetic bond (feature) is a linear combination of $n = 14$ real bonds.

So we would need n real bonds to create each synthetic.

One way to deal with this is to project the synthetic features onto $n'' < 14$ real bonds.

So we come up with "approximate" versions of the synthetic features themselves.

The interpretation of Yield Curve changes guides us into a simpler hedge

- Hedge the parallel shift with the 10 year Treasury (most liquid bond in the US)
    - works because all yield changes roughly the same for first PC
- Hedge the slope with a long/short portfolio of 2 year/30 year (or 10 year) bonds
    - also liquid instruments, that are "close enough" to PC 2
- Hedge the twist with a butterfly of long 2 year/10 year, short 5 year

So our Hedging Portfolio will consist of just a handful of very liquid bonds

- As opposed to $n = 14$ bonds, many of which may be illiquid

The primary way of hedging Fixed Income portfolios prior to this was by *duration* hedging

- Assuming that yields across all maturities moved the same amount
- Using a single liquid bond as the Hedge Portfolio

The PCA verified that the simple, intuitive hedge was actually the most important hedge to make !

Finance details

This section has little to do with Machine Learning but quite a bit to do with Fixed Income Finance.

- We have captured changes in *yield* of a bond
- To hedge *price* changes (our goal) we still need to translate a yield change to a price change

- For bond b with price P_b and yield y_b
  - We need $\dfrac{\partial P_b}{\partial y_b}$
    - the change in Price of bond b per unit change in its yield
  - This is known as the bond's *duration*
  - If we hold $\#_b$ units of bond b in a Portfolio
    - the bond's contribution to portfolio price change is $\#_b$ times the above sensitivity
  - Sometimes more convenient to compute the *percent price* change per unit yield change
    - size of the hedge now in *number of dollars* rather than *number of bonds*

PCA of the SP 500

The same analysis that we did for the Bond Universe works for other instruments.

Consider a universe of all stocks in a particular stock universe.

We can perform PCA on the returns (percent changes) of each stock

- Discover the common factors affecting all stocks in the universe.
- $n = 500$ features, for each example (one day return)

We don't have time to do it here, but the first components of many universes tends to be

- A first ("most important") component that has roughly equal impact
    - PC 1 is almost an equally weighted portfolio of all stocks
- Higher components expressing dichotomies
    - Cyclical stocks versus non-cyclical
    - Large cap versus small cap
    - Industry versus other industry

Interpretting the PC's

The key in our interpretation of the PCs for the Yield Curve

- Choice of ordering our original features by sorted maturity.

Had we chosen some other arrangement of the horizontal axis, we may not have seen the pattern.

So how do we find the "right" pattern ?

- Assign each original feature a set of attributes
    - Bonds: maturity
    - Stocks: industry, market capitalization
- Propose a "theory" about how the value of an original feature will respond to a level of the PC j
    - The theory should relate the attribute of an original feature to the value of feature
- As a horizontal axis: sort by the attribute proposed by your theory
    - Stocks:
        - $(j = 1)$: arbitrary arrangement works, since all stocks responds equally
        - $(j > 1)$: cluster stocks by attribute
            - Sort by market capitalization
            - Group by industry: first all Industrials, then all Techs, etc.

Let $\hat{u}$ be the $1 \times n$ vector of all 0's except for a 1 at position i

$$\hat{u}_i = 1$$
$$\hat{u}_j = 0, \quad j \neq i$$

Then

$$s = \hat{u} I V^T$$

is a $1 \times n$ vector whose elements are the effect of a one standard deviation shock in synthetic factor i on each original feature.

$$s_j = \text{change in } X_j \text{ for a 1 standard deviation change in } \tilde{X}_i$$

$$s' = \hat{u} \Sigma V^T$$

is a $1 \times n$ vector that is scaled by the actual standard deviation of $\tilde{X}_i$ since the absolute size of a 1 standard deviation change in $\tilde{X}_i$ is σ_i.

That is: row i of V^T is the effect on each original feature of a one standard deviation shock in synthetic feature $\tilde{X}_i$.

```python
In [36]: print("Done")
```

Done