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 $\mathbf{x^{(i)}}$ 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 $j < j'$

Observations

- Each basis vector (original or alternative) is of length n
- The number of basis vectors in the original and alternate basis is both n.
- We are describing the *same exact* examples, using different basis

$$\begin{aligned}
\mathbf{x^{(i)}} &= \sum_{j=1}^{n} \mathbf{x}_j^{(\mathbf{i})} * \mathbf{u}_{(j)} \quad \mathbf{x^{(i)}} \text{ defined in original basis} \\
&= \sum_{j=1}^{n} \tilde{\mathbf{x}}_j^{(\mathbf{i})} * \tilde{\mathbf{v}}_{(j)} \quad \tilde{\mathbf{x}}^{(\mathbf{i})} \text{ defined in alternate basis} \\
&= \tilde{\mathbf{x}}^{(\mathbf{i})}
\end{aligned}$$

We have done nothing more than find a different way to encode an example.

Now:

- Suppose we reduced the number of alternate basis vectors to $r < n$.
  - We set $\tilde{\mathbf{x}}_j^{(\mathbf{i})} = 0$ for $j > r$

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

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

Obviously

$$\tilde{\mathbf{x}}^{(\mathbf{i})} \neq \mathbf{x}^{(\mathbf{i})}$$

since we have lost information.

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

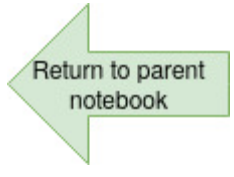
- Dropping the alternate basis of higher index loss minimal information

Hopefully the reduced dimensions $\tilde{\mathbf{x}}^{(\mathbf{i})}$ is close to the original $\mathbf{x}^{(\mathbf{i})}$

$$\tilde{\mathbf{x}}^{(\mathbf{i})} \approx \mathbf{x}^{(\mathbf{i})}$$

PCA is the process of

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


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

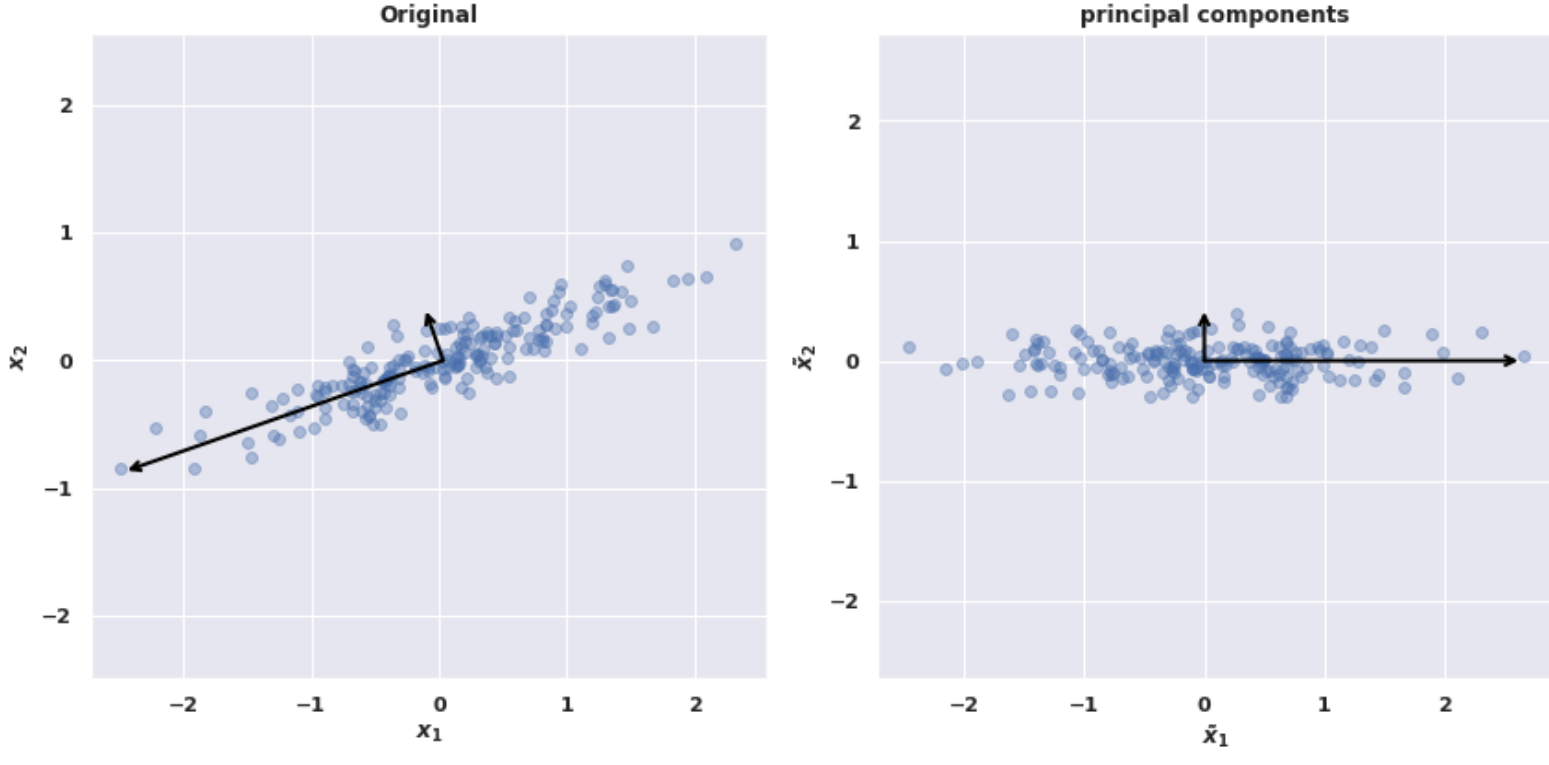
The key ideas behind PCA:

- 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.

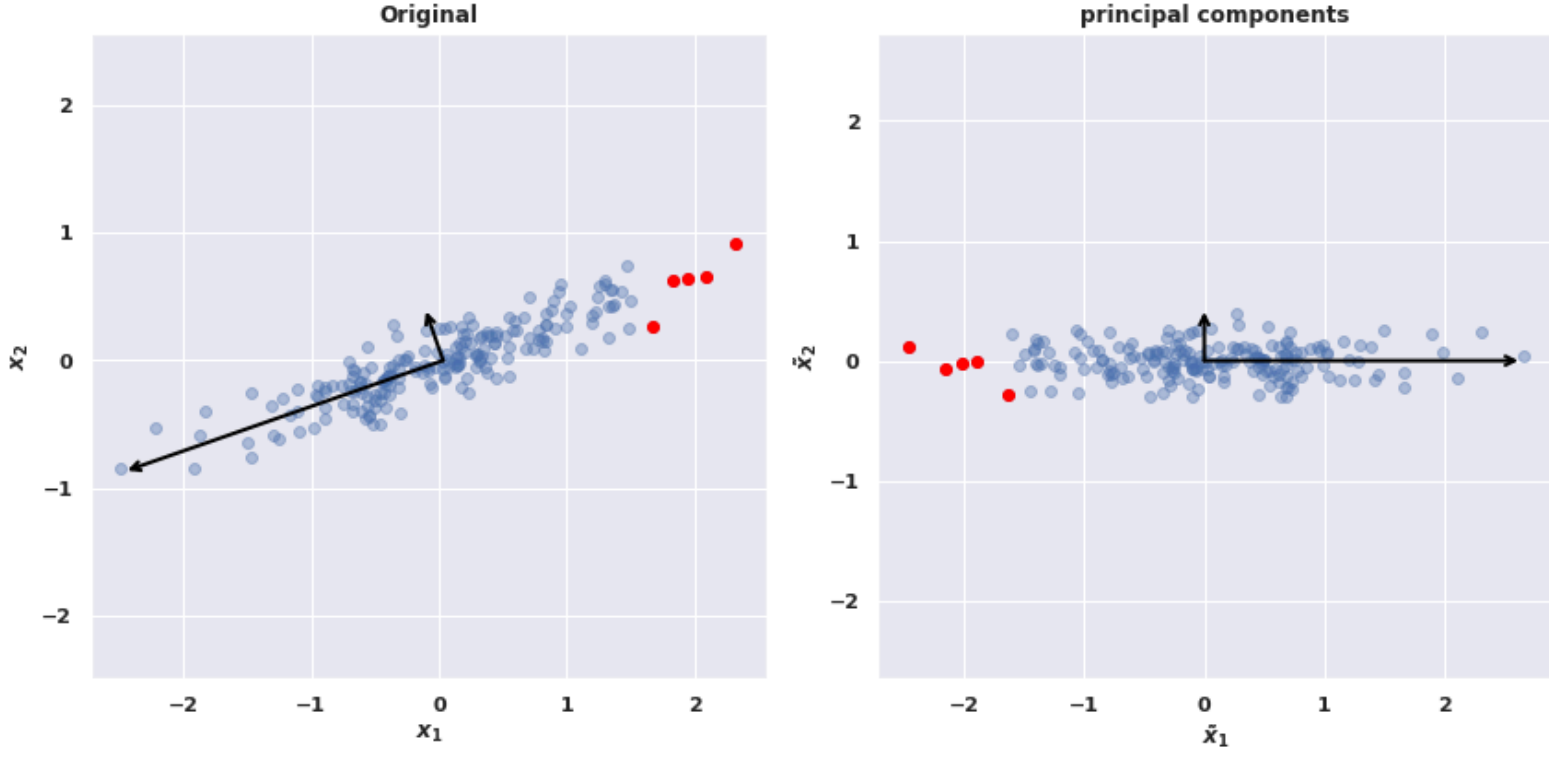
In the right plot: $\tilde{\mathbf{x}}_1$, $\tilde{\mathbf{x}}_2$ are

- 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

$$\mathbf{X} = \tilde{\mathbf{X}} V^T$$

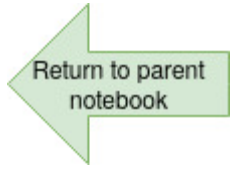
- The examples $\mathbf{X}$, expressed in the original basis
- Can be expressed as examples $\tilde{\mathbf{X}}$ in alternate basis
- V^T can transform coordinates in the alternate basis (V) back into coordinates in the original basis

Consider the example $\mathbf{x}$ (in original basis) and the same example $\tilde{\mathbf{x}}$ expressed in the alternate basis.

- the value of the features is different
- but they are just different encodings of the same example

Example $\mathbf{x}$ with feature values $\begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix}$

has feature values $\begin{pmatrix} \tilde{\mathbf{x}}_1 \\ \vdots \\ \tilde{\mathbf{x}}_n \end{pmatrix}$ in the alternate basis.


Return to parent
notebook

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 goal is to find

- a new set of basis vectors $\mathbf{V}^T$
- a way of expressing $\mathbf{X}$ in terms of $\mathbf{V}^T$
    - via loadings $\tilde{\mathbf{X}}$

$$\mathbf{X} = \tilde{\mathbf{X}}\mathbf{V}^T$$

We can view this expression as a factorization of matrix $\mathbf{X}$.

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

Moreover, the diagonal elements of Σ are in descending order of magnitude

$$\sigma_j > \sigma_{j'} \text{ for } j < j'$$

Given the SVD factorization of $\mathbf{X}$

$$\mathbf{X} = U\Sigma V^T$$

let us define $\tilde{\mathbf{X}}$ as

$$U\Sigma$$

so that

$$\mathbf{X} = \tilde{\mathbf{X}}\mathbf{V}^T$$

as required.

The definition of $\tilde{\mathbf{X}} = U\Sigma$ leads to an interesting interpretation of U.

- since Σ is a diagonal matrix
- the product $U\Sigma$
    - scales the columns of U by the diagonal elements of Σ

$$\begin{pmatrix} U_1^{(\mathbf{i})} * \sigma_1 \\ U_2^{(\mathbf{i})} * \sigma_2 \\ \vdots \\ U_n^{(\mathbf{i})} * \sigma_n \end{pmatrix}$$

by definition of matrix multiplication and the fact that all elements of column j of Σ are 0 except for row j.

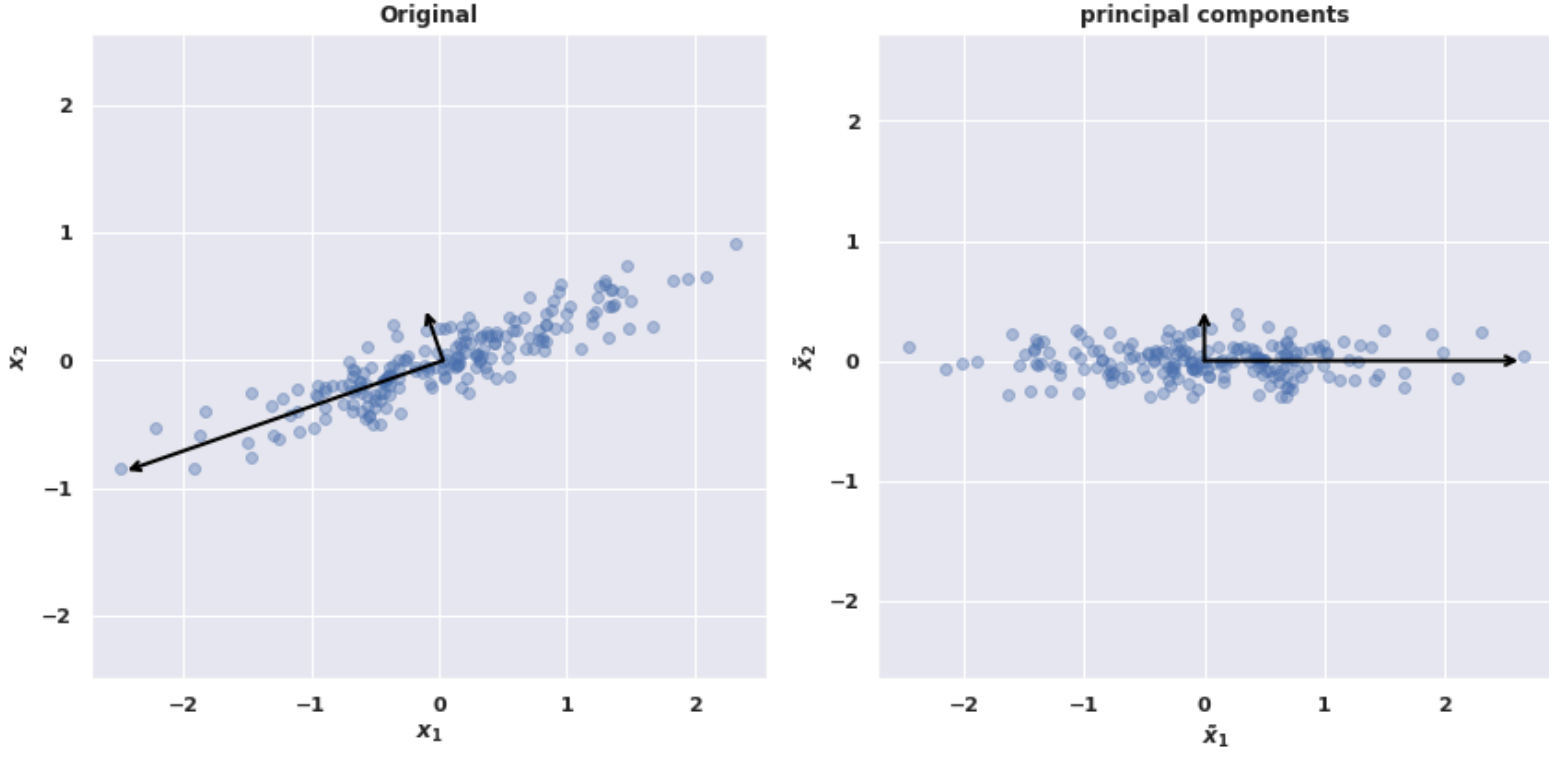
Thus

- 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 (right plot)

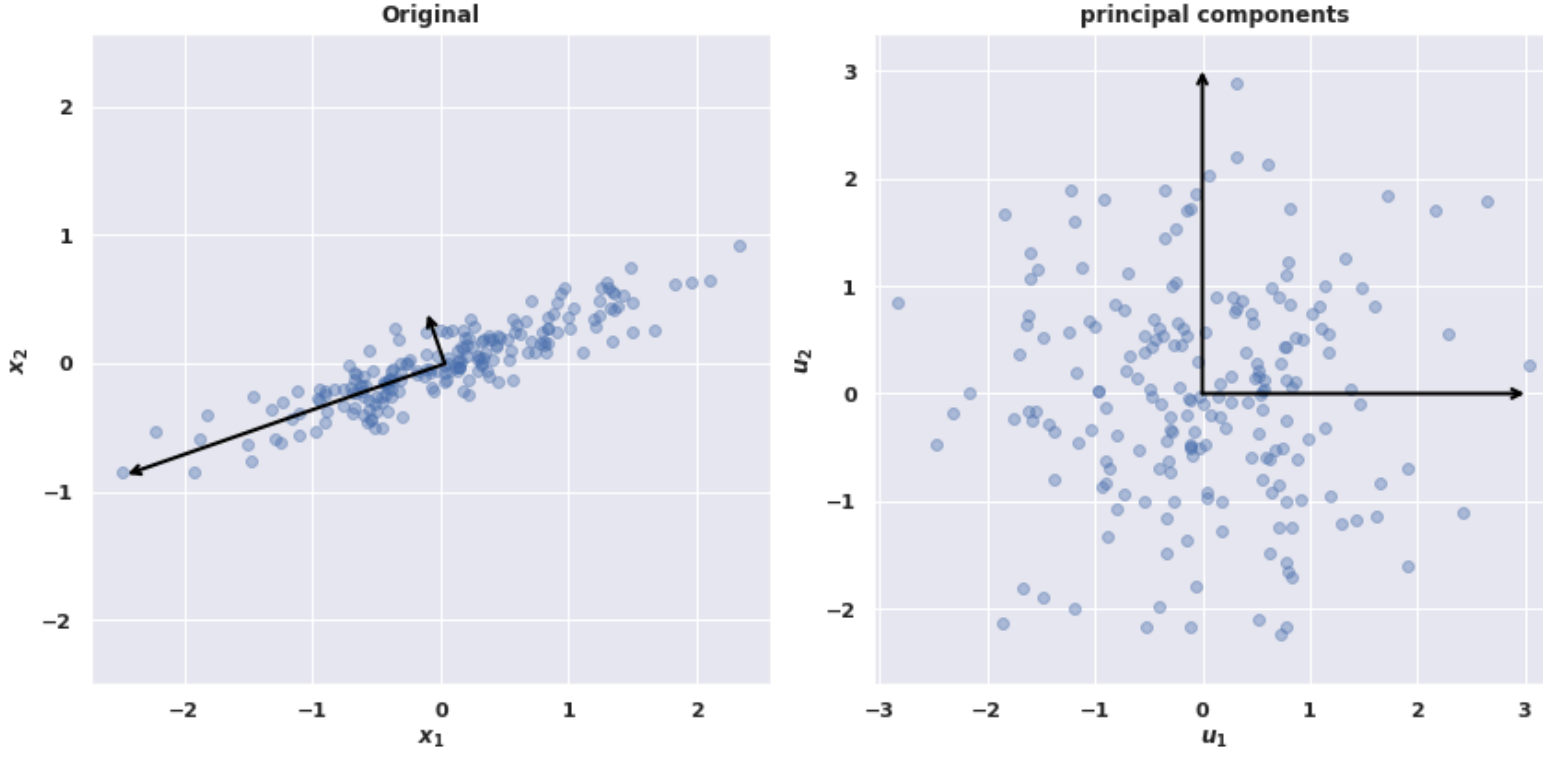
In [7]:
```
X = vp.create_data()

vp.show_2D(X)
```



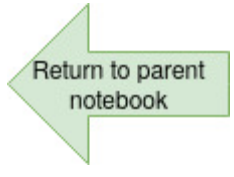
And here is the standardized plot of U (right plot)

- 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)
```



- 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


Return to parent
notebook

The new basis ("components") $\mathbf{V}^T$

It may be easier (for some) to view the matrix multiplication

$$\mathbf{X} = \tilde{\mathbf{X}}\mathbf{V}^T$$

as a simple sum and product.

We can do this by examining one row $\mathbf{X}^{(\mathbf{i})}$.

$$
\begin{aligned}
\mathbf{X} &= \tilde{\mathbf{X}}\mathbf{V}^T && \text{factorization of } \mathbf{X} \\
\mathbf{X}^{(\mathbf{i})} &= \tilde{\mathbf{X}}^{(\mathbf{i})}\mathbf{V}^T && \text{one row of the matrix mult} \\
&= [\tilde{\mathbf{X}}^{(\mathbf{i})} \cdot \mathbf{v}^{(1)}, \tilde{\mathbf{X}}^{(\mathbf{i})} \cdot \mathbf{v}^{(2)}, \ldots, \tilde{\mathbf{X}}^{(\mathbf{i})} \cdot \mathbf{v}^{(n)}] && \text{row } i \text{ column } j \text{ is the dot pr} \\
& && \text{and column } j \text{ of } \mathbf{V}^T \text{ is row }
\end{aligned}
$$

Thus

- the columns of $\mathbf{V}^T$ are the new basis vectors
- called the *components*
    - hence Principal Compoments
- with loadings $\tilde{\mathbf{X}}^{(\mathbf{i})}$

Dimensionality reduction

Thus far we have exactly replicated $\mathbf{X}$ via new basis 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

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

By setting

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

we zero out all features with index exceeding r

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

The dimensions of $\tilde{\mathbf{X}}'^{(\mathbf{i})}$ 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

$\mathbf{X}$ and $\mathbf{X}'$ have the *same* dimensions

$$(m \times n)$$

but the values in $\mathbf{X}'$ can only *approximate* the values in $\mathbf{X}$.

- because the alternate basis only has r basis vectors

$$\mathbf{V}^T : (r \times n)$$

- Note that each of the r vectors still has length n (as in the original basis)

So $\tilde{\mathbf{x}}$ is only of length r

- Hence, we achieve *dimensionality reduction*.

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

$$D = \{j \,|\, j > r\}$$

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

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

This is the feature-wise error $(\mathbf{x}_j^{(\mathbf{i})} - \mathbf{x}_j^{\prime(\mathbf{i})})^2$ of example i

- summed across all features j
- and all examples i

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 the coordinates $\tilde{\mathbf{x}}$ are unscaled coordinates (U) times scaling matrix Σ

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

and the relationship between $\mathbf{x}$ and $\tilde{\mathbf{x}}$ is given by

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

So, re-writing this relationship

$$\begin{aligned}
\mathbf{X}_j^{(\mathbf{i})} &= [U_1^{(\mathbf{i})} * \sigma_1, U_2^{(\mathbf{i})} * \sigma_2, \ldots, U_n^{(\mathbf{i})} * \sigma_n] \cdot (V^T)_j && \text{where } (V^T)_j \text{ is column} \\
&= \textstyle\sum_{k=1}^{n} U_k^{(\mathbf{i})} * \sigma_k * (V^T)_j^{(k)} && \text{multiply row } i \text{ of U by c} \\
&= \textstyle\sum_{k=1}^{n} \sigma_k * (U_k^{(\mathbf{i})} * (V^T)_j^{(k)})
\end{aligned}$$

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

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

Because the diagonal elements of Σ are in decreasing order of magnitude

$$\sigma_j > \sigma_{j'} \quad \text{for } j < j'$$

choosing D to be

$$D = \{j \,|\, j > r\}$$

results in dropping terms $U_k^{(\mathbf{i})} * (V^T)_j^{(k)}$ that are scaled by the $(n - r)$ *smallest* values of 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

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

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

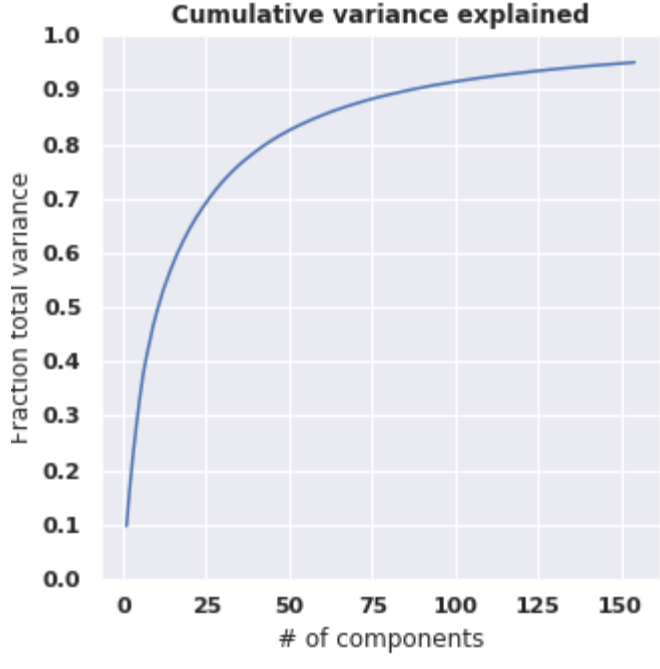
- As the r such that $s_r \geq T$
- Where T is a threshold fraction for explained variance
- For example

$$T = 95\%$$

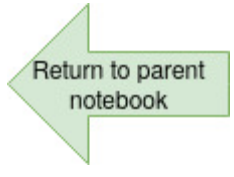
A detailed example of PCA performed on the MNIST digits follows in a later section.

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

PCA: MNIST digits, cumulative variance



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 synthetic features $\tilde{\mathbf{X}}$ to original features $\mathbf{X}$

$$\mathbf{X} = \tilde{\mathbf{X}}\mathbf{V}^T$$

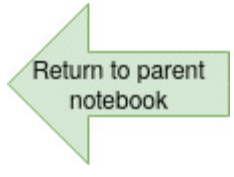
How about inverting the transformation: derive synthetic $\tilde{\mathbf{X}}$ from original $\mathbf{X}$?

Since

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

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

```
# 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

Our original examples $\mathbf{X}$ can be approximated

- with reduced dimensionality $r < n$
- with loadings $\tilde{\mathbf{X}}$ on $\mathbf{V}^T$

$$\mathbf{X} \approx \tilde{\mathbf{X}}\mathbf{V}^T$$

where

$$\begin{aligned} \mathbf{X} &: (m \times n) \\ \tilde{\mathbf{X}} &: (m \times r) \\ \mathbf{V}^T &: (r \times n) \end{aligned}$$

The approximation becomes equality only when $r = n$ (no dimensionality reduction)

Using a single example $\mathbf{x} \in \mathbf{X}$

- with corresponding reduced dimensionality representation $\tilde{\mathbf{x}}$
- we will vary r
- in order to demonstrate
    - the increased quality of reconstruction as r increases

We will also visualize the principal components

- in an attempt to interpret them

Our examples will be (8×8) pixel grids

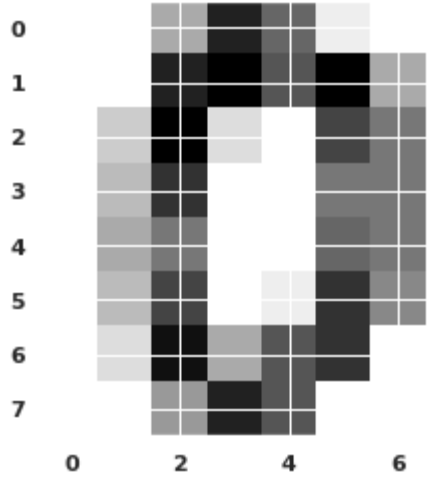
- representing a subset of the digits:
$$\{0, 4, 7, 9\}$$
- $n = 64$
- we choose $r = 8$

We choose a subset to aid the visualization.

Here is one example $\mathbf{x} \in \mathbf{X}$

In [12]:
```
fig0
```

Out[12]:



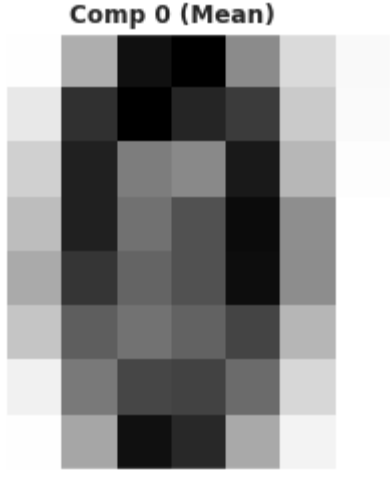
Recall

- The examples used in PCA should be **zero-mean**
- We apply a transformation to $\mathbf{X}$ to transform each feature to mean 0
- When reconstructing $\mathbf{x}$ from $\tilde{\mathbf{x}}$
    - we will have to *add back* the mean

Here is the mean of $\mathbf{X}$

In [13]:
```
figm
```

Out[13]:



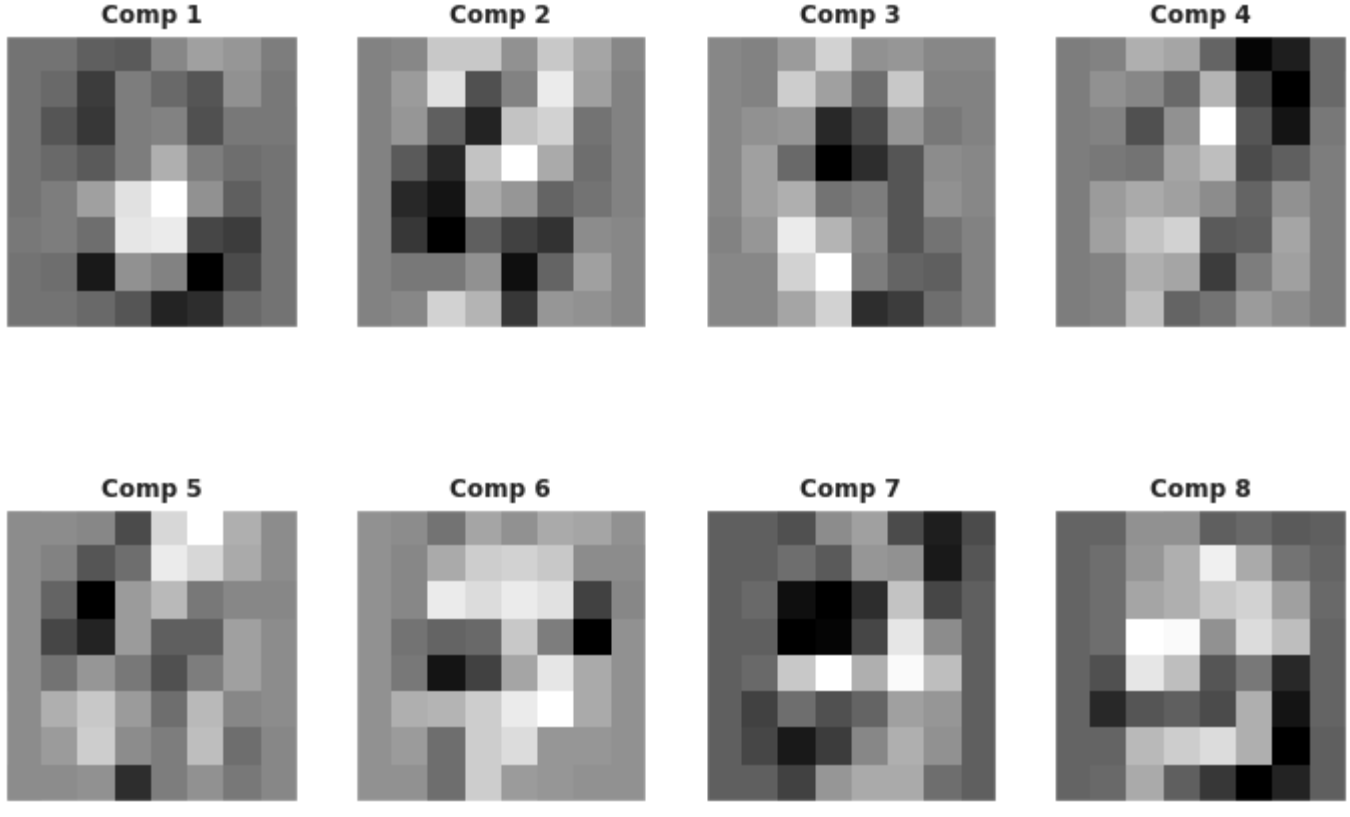
And here are the Principal Components (new basis)

- the rows of $\mathbf{V}^T$
  - $\mathbf{V}^T : (r \times n)$
  - r components, each of length n

visualized as (8×8) pixel grids

In [14]:
```
figc
```

Out[14]:



The components often correspond to "concepts"

- properties shared among subsets of examples

We can see that

- Component 1 is the concept corresponding to digit "0"
- Component 2 is the concept corresponding to digit "4"
- Components 3 and greater are concepts that we can't easily identify

It is not often easy to identify components other than the first few.

We can try to visualize how much information is lost via dimensionality reduction by

- reconstructing an example
- that has been reduced to r dimensions
- for increasing values of r

Let's progressively examine the approximation of $\mathbf{x}$

- using an increasing number of synthetic features r
  $\mathbf{x^{(i)}} \quad = \quad \sum_{j=1}^{r} \tilde{\mathbf{x}}^{\mathbf{(i)}} * \mathbf{V}^{(j)} \quad \mathbf{V}^{(j)} \text{ is column } j \text{ of } \mathbf{V}^T$

Note

- each summand
  $$\tilde{\mathbf{x}}^{\mathbf{(i)}} * \mathbf{V}^{(j)}$$
  is a **vector** (pair-wise product of two vectors $\tilde{\mathbf{x}}^{\mathbf{(i)}}$ and $\mathbf{V}^{(j)}$)
    - one component, weighted by by $\tilde{\mathbf{x}}^{\mathbf{(i)}}$
- so the sum $\mathbf{x}^{\mathbf{(i)}}$ is a weighted (by $\tilde{\mathbf{x}}^{\mathbf{(i)}}$) sum of r basis vectors $\mathbf{V}^{(j)}$

This equation says that we construct an approximation of $\mathbf{x^{(i)}}$

- By adding $(V^T)^{(j)}$, each of length n
- That are weighted by $\tilde{\mathbf{x}}_j^{\mathbf{(i)}}$

The resulting weighted sum is a vector of length n, as needed.

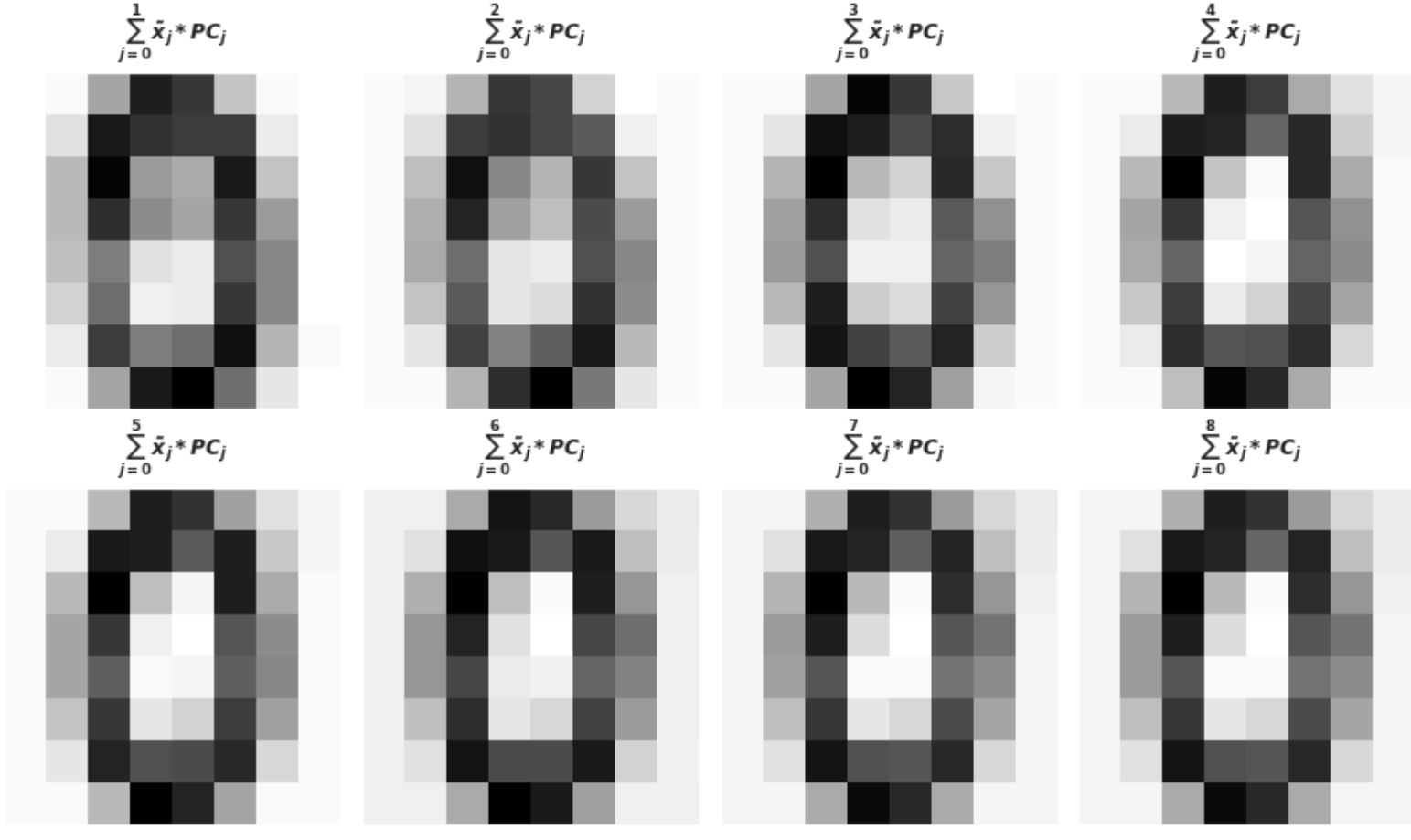
Recall

- $\mathbf{x}^{(\mathbf{i})}$ has been transformed to be *zero-mean*
- we will visualize the approximation of $\mathbf{x}$
    - by *adding back the mean*
    - modifying the summation
        - with element 0
        - having loading 1
        - on the mean $\mathbf{X}$

Here are the reconstructions as we vary r

In [16]:
```
figi
```

Out[16]:



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

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

This is not surprising since both $\mathbf{x}$ and the first component are instances of the concept 0.

You are encourage to play with the code

- to view the approximation of other examples from $\mathbf{X}$ that don't depict 0

Can we interpret the components ?

The components (alternative basis) often encode higher level "concepts"

- properties that are shared among a subset of examples

One objective is to demonstrate how we can try to interpret the components.

We previously tried to interpret the components by visualization.

An alternative:

- looking at $\tilde{\mathbf{x}}^{(\mathbf{i})}$ numerically
- here we look at $\tilde{\mathbf{x}}^{(\mathbf{i})}$ for an example of a digit "0"

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 =  [ 18.94187383   5.09552834 -11.11788275   6.39697251  -0.96294356
   3.23881823   2.53645528   0.04997342]
Largest feature at index 1
```

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

Remember:

- the example for which $\tilde{\mathbf{x}}$ is the vector has label "0".
- the first component looks very much like a "0"

Thus, it makes sense that this example "0" would have high loading on the concept 0.

We might try to confirm our interpretation of the first component as the concept 0

- by examining the loading on the first component
- of *other* examples in $\mathbf{X}$ that correspond to the digit "0"
    - By examining $\tilde{\mathbf{x}}^{(i')}$ for all i' where $\mathbf{y}^{(i')} = 0$ (assuming we have targets/labels)

In [19]:

```
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', '2.54', '0.05']
['10.94', '11.59', '-8.82', '8.34', '-6.50', '4.20', '4.22', '-4.47']
['15.16', '8.46', '-9.70', '2.85', '-3.39', '-5.13', '-3.83', '-7.17']
['21.68', '9.93', '-12.65', '3.27', '1.32', '4.84', '-0.86', '-4.05']
['13.36', '8.83', '-11.29', '4.24', '-0.35', '-1.18', '0.98', '-12.11']
['19.57', '7.25', '-9.87', '-3.35', '-1.75', '7.12', '4.04', '0.29']
['20.21', '9.81', '-7.81', '-2.10', '-2.19', '-1.38', '-0.83', '3.77']
['10.75', '12.33', '-9.70', '-1.01', '-4.82', '-7.07', '-4.91', '-12.88']
['17.96', '12.42', '-5.09', '-5.08', '2.47', '-8.20', '-2.90', '-11.07']
['22.48', '1.51', '-7.25', '-10.33', '3.19', '3.59', '0.18', '4.93']
```

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

This provides support for our theory that the first component corresponds to the concept 0.

We will re-visit the topic of interpreting the components in a little bit.

Taking advantage of reduced dimensionality

Reducing the dimensionality of examples from n features to $r < n$ features has several advantages

- large number of features are hard to manage, both logically and computationally
- collinearity between features may increase as n grows
    - numerical problems for some models: Linear Regression, Logistic Regression

Moreover, the new synthetic features $\tilde{\mathbf{x}}$ are *orthogonal*

- easier to ascribe a contribution of each feature to the target

But a great advantage is that we can plot in 2 and 3 dimensions.

This can be useful for analyzing whether

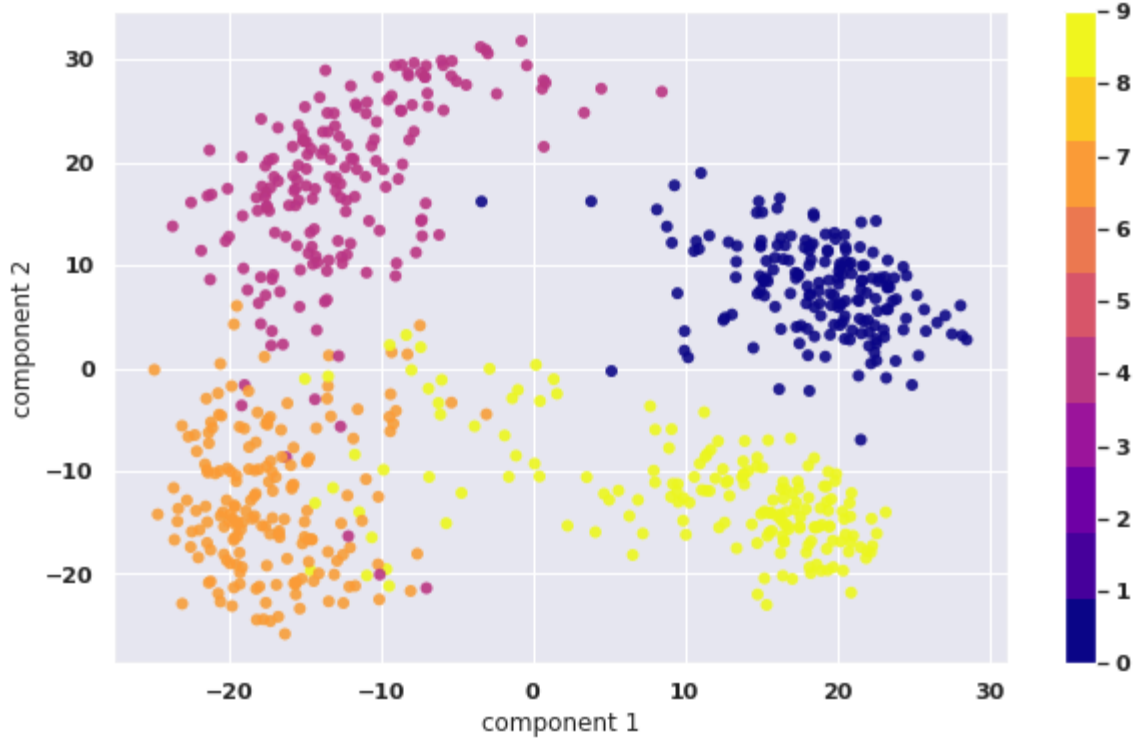
- examples expressing a common property
- form clusters

Let's plot the examples in our subset of 4 of the digits

- using just the first 2 components ($r = 2$)

In [20]:
```
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.

In the absence of colors (i.e., labels associated with each example)

- it would take some effort to determine the property common to each cluster

Fortunately, in this case

- our examples have labels
- which we use to color the points in the plot

We can see that the "common property" of each cluster is the label.

The "common property" might be something **other** than the label.

For example, had we used all 10 digits we might have various hypotheses for the common property

- digits with strong vertical visualization (1, 4, 7)
- open versus closed visualization: 3 versus 8, 7 versus 9

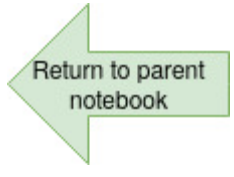
We can purse the interpretation of the components further.

For example, the 0's (upper right cluster)

- have high loadings $\tilde{\mathbf{x}}_1$
- but so do 9's (lower right cluster)
- 0's differ from 9's only in the value of their $\tilde{\mathbf{x}}_2$
    - does this suggest an interpretation for component 2 ?

Similarly

- 0's and 4's (upper half)
- have similar $\tilde{\mathbf{x}}_2$
- but different sign for $\tilde{\mathbf{x}}_1$
    - does this suggest and interpretation for component 1 ?


Return to parent
notebook

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 [21]:
```
ush = unsupervised_helper.PCA_Helper()

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

```
Retrieving MNIST_784 from cache
```

In [22]:
```
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[22]:
```
((70000, 784), (70000,))
```

Out[22]:
```
(52500, 784)
```

Perform PCA.

In [23]:
```
pca_mnist = ush.mnist_PCA(X_mnist_train)
```

In [24]:
```
pca_mnist.n_components_
X_mnist_train_reduced = ush.transform(X_mnist_train, pca_mnist)
X_mnist_train_reduced.shape
```

Out[24]: 154

Out[24]: (52500, 154)

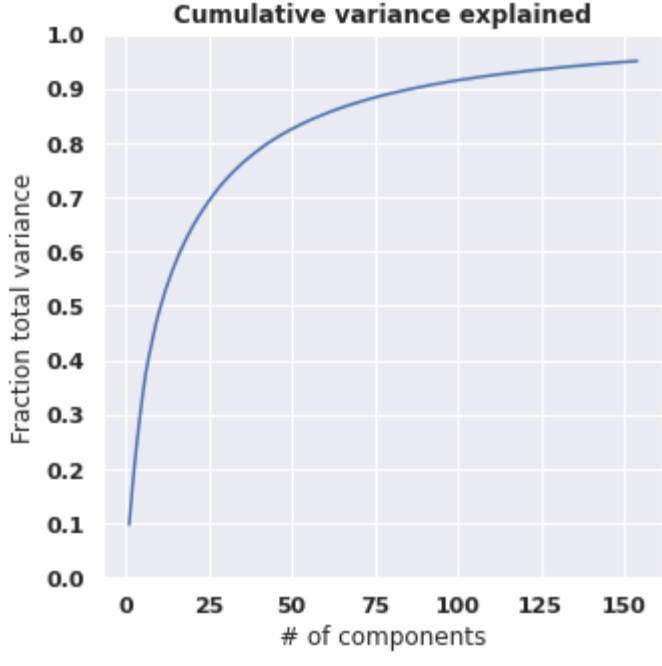
Let's plot the cumulative variance as a function of number of synthetic features.

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

In [25]:

```
_ = 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.

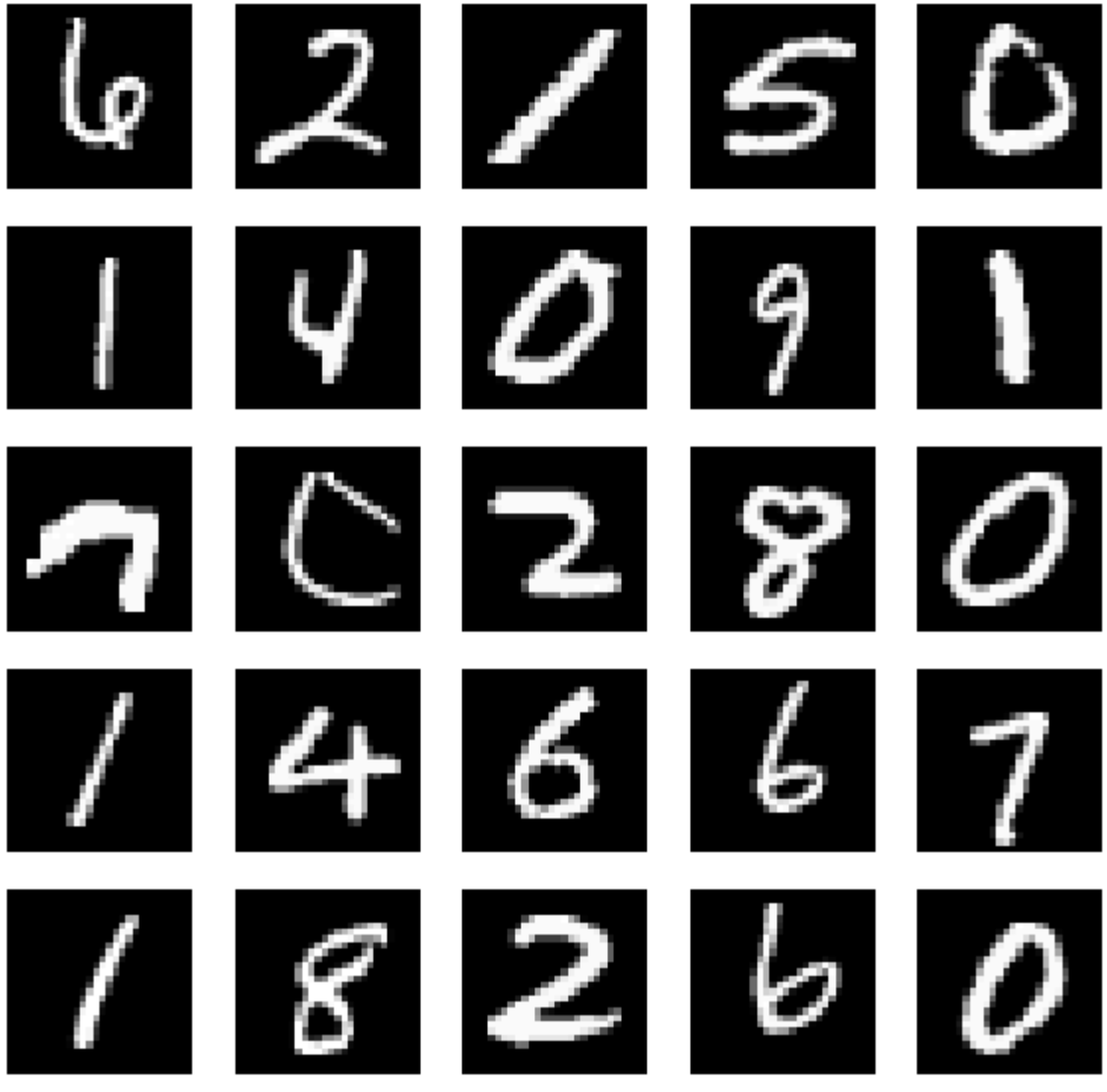
That is, we can see what the digits look like when reconstructed from only 154 synthetic features.

First, let's look at the original:

In [26]:
```
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, title="Original")
```

Out[26]:
```
(52500, 154)
```

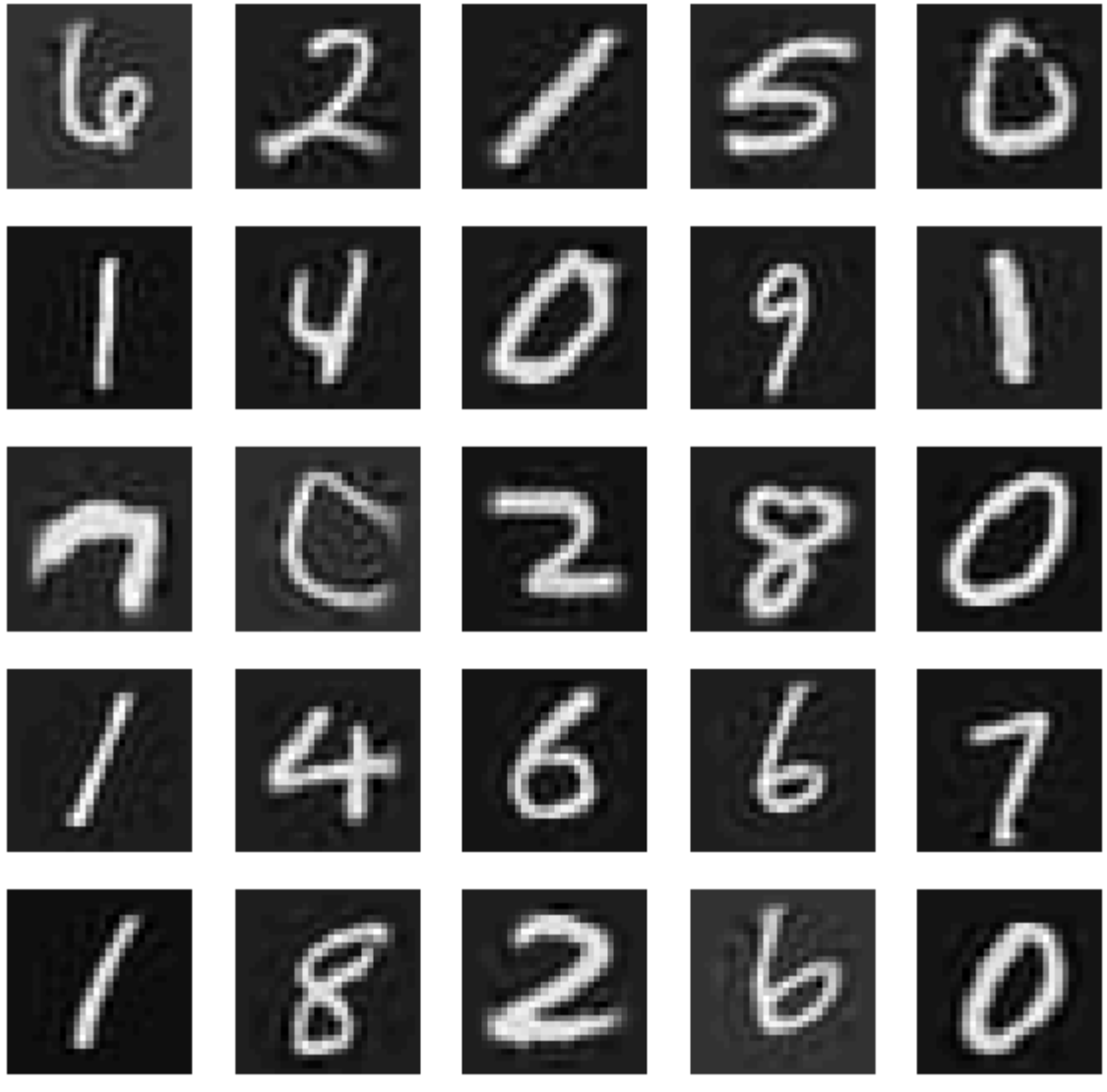
Original



Next, the reconstructed

In [27]:
```
X_mnist_train_reconstruct = ush.inverse_transform(X_mnist_train_reduced, pca_mnist)
_ = ush.mnh.visualize(X_mnist_train_reconstruct, y_mnist_train, title="Reconstructed")
```

Reconstructed

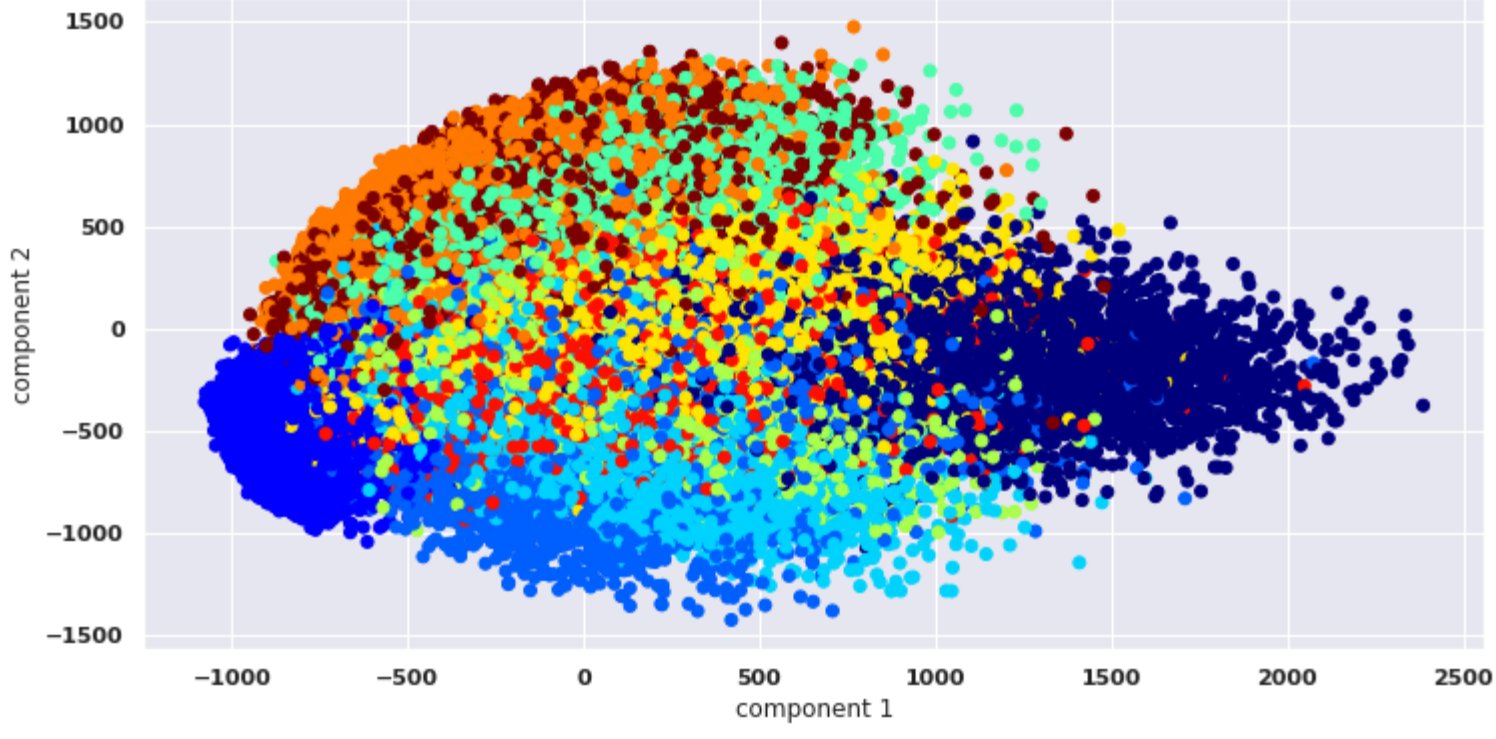


A little fuzzy, but pretty good.

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 [28]:
```
_ = 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 [29]:

```
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 16.89%, about 132.4 pixels
```

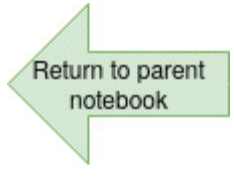
Is 17% good ? You bet !

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).


Return to parent
notebook

PCA in Finance

PCA of yield curve

Litterman Scheinkman (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 Treasury bonds with many maturities.
- A common goal in Fixed Income Finance is to *immunize* (hedge) a portfolio to changes in the Yield Curve.
- The ideal hedge for a Treasury bond
    - is a bond with similar maturity date

But there are lots of maturity dates !

A more parsimonious way of hedging is to

- define a small number $(n = 14)$ of *benchmark* maturities
- for each bond in the portfolio
    - compute the sensitivity of the price of the bond
    - to a change in yield
    - for each benchmark

So each bond in the portfolio is represented as $n = 14$ features

- the sensitivity to each of the benchmark bonds

By summing up the sensitivities of all bonds in the portfolio

- we have a single vector of length $n = 14$
- describing the total value of benchmark bonds
- that we need to *go short*
- in order to hedge our portfolio against changes in the yield curve

But even $n = 14$ hedging instruments is large

- transaction costs associated with each

The solution ?

PCA !

- replace $n = 14$ benchmark bonds
- with $r < n$ synthetic benchmarks

Taking the PCA

We will obtain many examples of the Yield Curve.

Each example (daily yield curve)

- is a vector of $n = 14$ yields of the benchmarks, on a particular day

Here are a few examples

- sampled at a end-of-month

In [30]:
```
ych = unsupervised_helper.YieldCurve_PCA()

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

Out[30]:

	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 [31]:
```
data_yc.shape
```

Out[31]:
```
(287, 14)
```

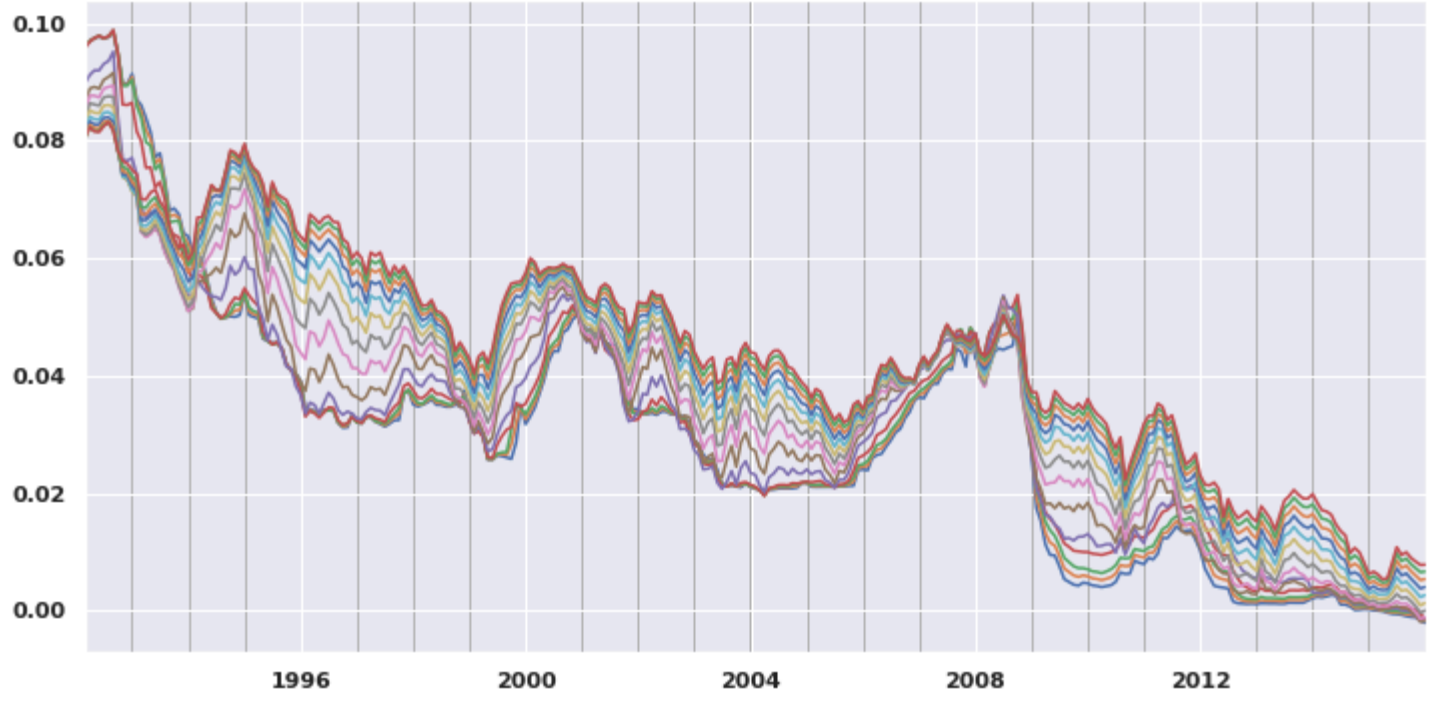
Each example (row) has 14 features

- the end-of-month Yield Curve
- the yields for 14 maturity points on a given date.

Let's plot the history of yield curves

In [32]:
```
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)
```



In hedging

- we are concerned with **changes** in yield
- rather than the *level* of the yield
- we compute the sensitivity of a bond with respect to a *change in yield* of the hedging instrument

So we will perform PCA

- on **changes** in yield
- rather than the level of yield

of the 14 benchmarks.

Our goal is to reduce dimensionality from n to $r < n$.

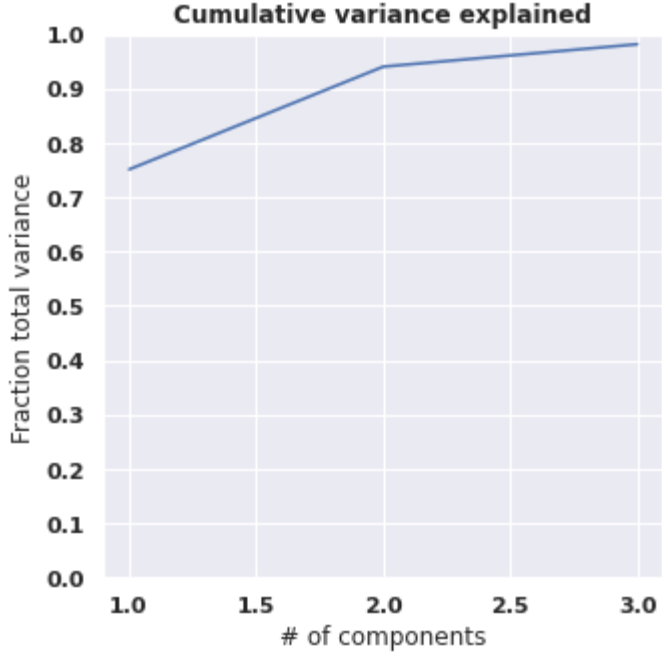
What value of r is "good enough" ?

As usual

- we plot the cumulative variance
- as a function of the number of reduced dimensions
- choosing the r
- that meets a chosen threshold (e.g, 95%) of cumulative variance

Let's look at the plot.

In [33]:
```
pca_yc, df_pca_yc = ych.doPCA(data_yc, doDiff=True)
_ = ych.plot_cum_variance(pca_yc)
```



Wow !

You'll notice that we don't display more than 3 dimensions

- that's because *almost 100 percent* of cumulative variance
- is captured by the first 3 components

So, we can hope to hedge

- with at most 3 synthetic bonds

It gets even better !

By examining

- the influence of each component
- on the $n = 14$ benchmark yield changes (original features)

we can *interpret* what the components represent

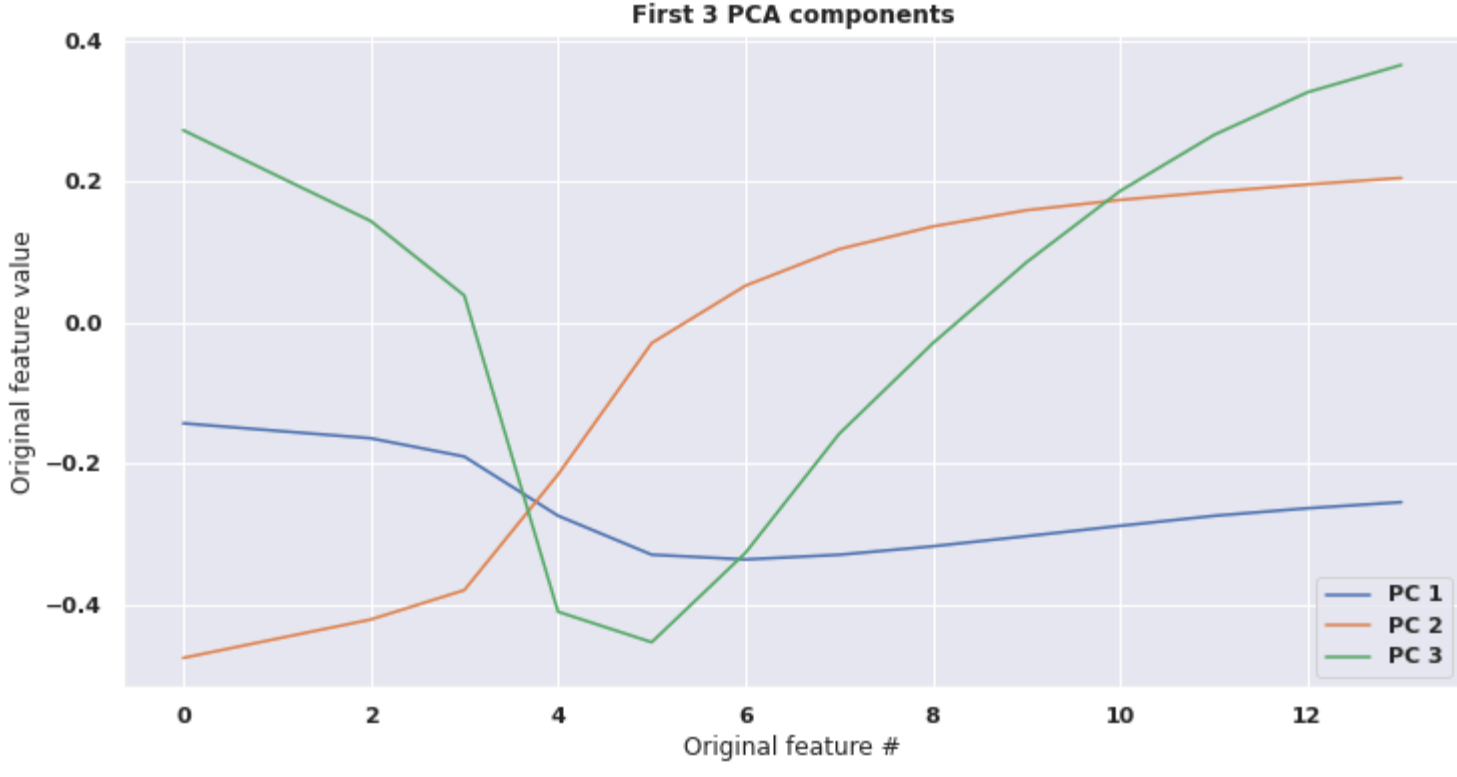
For each component

- shock the component by 1 standard deviation
    - see below for how to shock
- compute the change (measured in standard deviations) of each of the $n = 14$ yield changes (original features)

We plot a line for each component

- arranging benchmark maturities by increasing value as the horizontal axis
- Component 1: blue line
- Component 3: orangle line
- Component 3: green line

In [34]:
```
ych.plot_components(pca_yc)
```



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

- $(j = 1)$: results in a roughly equal yield change across the $n = 14$ benchmarks
    - 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

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

Aside

We are measuring changes in units of standard deviation.

Recall that

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

- U are the *standardized* features (unit variance)
- that are scaled by diagonal matrix Σ of variances

So we are graphing

- U rather than $\tilde{\mathbf{X}}$

This also mean we can't compare the levels of the 3 separate curves

- since each component has a different standard deviation

$$\sigma_1 > \sigma_2 > \sigma_3$$

That is

- a unit standard deviation change in component j
- represents a large change
- than a unit standard deviation change in component $j' > j$

Computing the effect on original features of change in synthetic feature

How do we obtain

- the effect of a small change in synthetic feature $\tilde{\mathbf{x}}_j$
- on each *original* feature $\mathbf{x}_{j'}, 1 \leq j' \leq n$

Recall

$$\begin{aligned} \mathbf{X} &= \tilde{\mathbf{X}}\mathbf{V}^T && \text{definition} \\ &= U\Sigma\mathbf{V}^T && \text{since} \tilde{\mathbf{X}} = U\Sigma \end{aligned}$$

where

$$\begin{aligned} \mathbf{X}: &\quad (m \times n) \\ U: &\quad (m \times r) \\ \Sigma: &\quad (r \times r) \\ \mathbf{V}^T: &\quad (r \times n) \end{aligned}$$

Consider example i : $\mathbf{x}^{(\mathbf{i})}_{j'}$

$$\mathbf{x}^{(\mathbf{i})} \quad = \quad U^{(\mathbf{i})}\,\Sigma\,\mathbf{V}^T \quad \text{one row of the matrix multiplication}$$

A one standard deviation increment in $\tilde{\mathbf{x}}_j$

- increases $U^{(\mathbf{i})}_j$ by 1
- which results in an increase in $\mathbf{x}^{(\mathbf{i})}$ by
  $$\Delta\mathbf{x}^{(\mathbf{i})}(j) = \mathrm{OHE}(j)\,\Sigma\,\mathbf{V}^T$$
  where $\mathrm{OHE}(j)$ is the OHE vector of length n with element j being non-zero
  $$\mathrm{OHE}(j)_k == \begin{cases} 0 & k \neq j \\ 1 & k = j \end{cases}$$

Observe that

- $\Delta\mathbf{x}^{(\mathbf{i})}(j)$ is the same for *all* examples i

  Write as $\Delta\mathbf{x}(j)$ (no superscript).

- Left multiplication by $\mathrm{OHE}(j)$ selects row j of the right multiplicand

$$\mathrm{OHE}(j) * \Sigma = \sigma_j * \mathrm{OHE}(j) \quad \mathrm{OHE}(j) \text{ selects row } j \text{ of } \Sigma$$

- Expanding $\Delta\mathbf{x}(j)$

$$\begin{aligned}
\Delta\mathbf{x}(j) &= \mathrm{OHE}(j)\,\Sigma\,\mathbf{V}^T \\
&= \sigma_j * \mathrm{OHE}(j)\,\mathbf{V}^T && \text{since above shows } \mathrm{OHE}(j) * \Sigma = \sigma_j * \mathrm{OH} \\
&= \sigma_j * \mathbf{V}_j && \mathrm{OHE}(j) \text{ selects row } j \text{ of } \mathbf{V}^T \\
& && \text{equivalently: column } j \text{ of } \mathbf{V}
\end{aligned}$$

Thus

- $\Delta\mathbf{x}(j) = \sigma_j * \mathbf{V}_j$
- is the vector of length n
- such that element j' : $\Delta\mathbf{x}(j)_{j'}$
- is the effect on original feature j' (independent of example)
- of a one standard deviation change in synthetic feature j

In other words

- $\mathbf{V}_j$ is the effect of a unit change in $\tilde{\mathbf{x}}_j$
- $\sigma_j * \mathbf{V}_j$ is the effect of a unit change in $\mathbf{u}_j$, where $\mathbf{u}$ is the standardized $\tilde{\mathbf{x}}$
    - equivalent to a one standard deviation change in $\tilde{\mathbf{x}}_j$

Constructing the hedges (synthetic bonds)

The good news:

- we can construct a hedge portfolio using only 3 synthetic instruments (the components)
- and lose almost no information

The bad news:

- each synthetic bond (feature) is a linear combination of $n = 14$ real bonds.
- although we have $r = 3$ synthetic components
- each is a vector length $n = 14$

So, in order to use fewer than $n = 14$ real bonds

- we will have to *approximate* the components
- using many fewer instruments

We

- choose one benchmark as our "proxy" for a parallel yield curve change
    - a very liquid maturity: 10 year
- construct a proxy for "slope change" using two benchmark bonds
    - long a long-maturity bond, short a short-maturity bond
    - both liquid: 2 year/30 year
- construct a proxy for the "butterfly change" using 3 benchmark bonds
    - long short and long-term maturities, short intermediate-maturity
    - 2 year/5 year/10 year butterfly

Using Linear Regression

- we can project the 14 benchmarks onto the 3 proxies

So rather than using *true* components (and true loadings on them)

- we use approximations
- that are consistent with our interpretation

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 $\frac{\partial P_b}{\partial y_b}$
        - 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.

Recall our "factor model" of stock returns from our introduction

$$\mathbf{X}_1 = \beta_{1,\text{idx}} * \tilde{\mathbf{X}}_{\text{index}} + \beta_{1,\text{size}} * \tilde{\mathbf{X}}_{\text{size}} + \epsilon'_1$$

Using PCA

- we can discover common factors (i.e., the components)

The components are the common factors.

These are

- statistical artifacts
    - composed of the original large n stocks
- rather than easy-to-interpret factors
    - Index, Size, Industry

But perhaps, like the PCA of the Yield Curve

- we can interpret the components
- as expressing economically meaningful factors

The interpretation of the components may change with time.

But one thing that is fairly constant

- the first ("most important") component
- is an almost equally weighted combination of all stocks in the universe
- the "parallel shift" factor
    - affecting all stocks
    - is an equally-weighted market index

The higher components vary much more.

But they often represent dichotomies.

Some common themes (hypotheses for you to investigate in trying to interpret them)

- Economic cycle
    - cyclical versus non-Cyclical stocks
- Size
    - Large versus small cap stocks
- Industry
    - in one particular industry versus not in that industry

Interpretation is not easy

- not as simple as the "labels"
- require economic intuition

Interpreting the PC's

We have previously illustrated several tools to interpret the components.

One way was via clustering

- hypothesize a common property of a cluster
- interpret the components by how
    - a change in component
    - changes the common property
        - e.g., moving from "open curve (3)" to "closed curve (8)"

Another way

- hypothesize the common property among a subset of examples
- examine the loadings of each example on the various components

That is how, in the (8×8) pixel grid of digits

- we can to interpret the first component
- as the concept of being 0

For PCA of the Yield Curve

- we *implicitly* hypothesized that the components
- could affect subsets of maturities differently

We were able to visualize the subsets by plotting

- sensitivity of original feature
- to unit change in component

But this plotting *only worked*

- because we (luckily) chose to arrange the horizontal access by maturity
- the subsets depended on the *relative order* of maturities
    - e.g., Components 2: long versus short

Had the subsets been defined differently

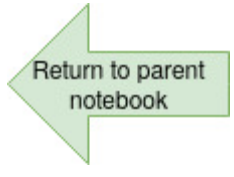
- the plots would not been revealing

Suggestion

- come up with an hypothesis
- arrange your examples along the horizontal access according to the intensity of an example on this hypothesis
    - e.g., arrange all Cyclical stocks before all Non-Cyclical stocks
    - e.g., arrange all Tech stocks before all Non-Tech stocks
- vertical axis
    - how much the *original* feature (e.g., return of the stock)
    - is affected by a unit change in the component

If your hypothesis is true

- similar examples, with similar sensitivities to the compoment
- will appear adjacent to one another
- rather than randomly scattered


Return to parent
notebook

In [35]:
```
print("Done")
```

```
Done
```