Principal Component Analysis

AI+X
INSTITUTE FOR ARTIFICIAL INTELLIGENCE

Principal Component Analysis (PCA)

- PCA is an unsupervised learning method in the sense that it does not require labels/annotations for the training samples
- PCA is fundamentally a dimensionality reduction algorithm
  - a tool for visualization
  - noise filtering
  - feature extraction and engineering

Principal Component Analysis (PCA)

- PCA behavior is easier to visualize in two-dimensional data
- PCA is basically a rotation+translation transformation that concentrates information (variance) in the initial axes

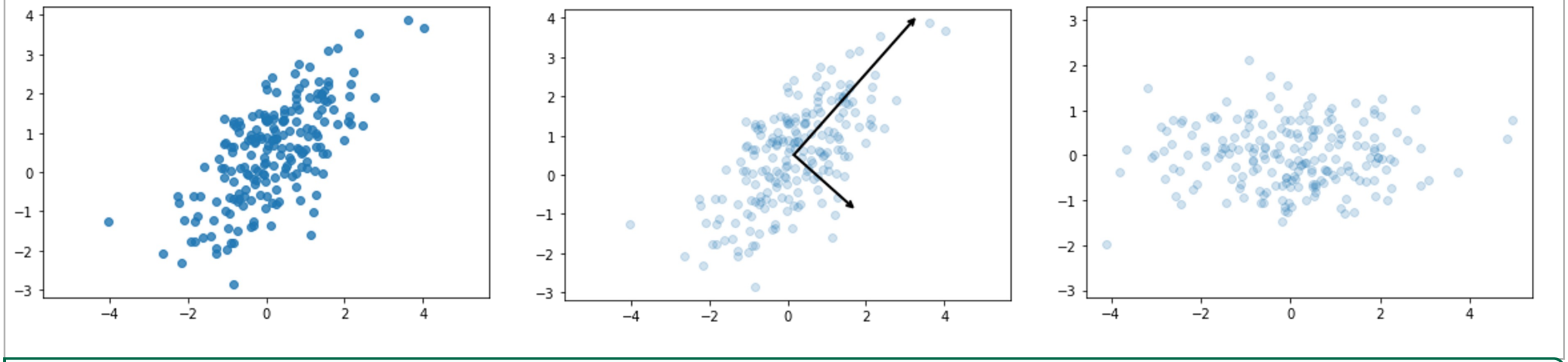

Figure 1. Original two-dimensional data (left), principal components extracted using PCA (center), and transformed data (right).

PCA Algorithm

1. Get some data $\mathbf{A} = \{\mathbf{a_1}, \ldots, \mathbf{a_N}\}$
2. Subtract the mean $\mathbf{A'} = \{\mathbf{a_1} - \bar{\mathbf{a}}, \ldots, \mathbf{a_N} - \bar{\mathbf{a}}\}$
3. Calculate the covariance matrix $\mathbf{C} = \mathbf{A'} \times \mathbf{A'^T}$
4. Calculate the eigenvectors $\mathbf{v_j}$ and eigenvalues $\boldsymbol{\lambda_j}$ of the covariance matrix
   - Sort the eigenvectors in decreasing order of eigenvalues, so that $\boldsymbol{\lambda_j} > \boldsymbol{\lambda_{j+1}} \forall \mathbf{j}$
5. Transform $\mathbf{a_i}$ into $\mathbf{w_i}$: $\mathbf{w_i} = \mathbf{V}(\mathbf{a_i} - \bar{\mathbf{a}})$
6. Reconstruct $\mathbf{a_i}$ from $\mathbf{w_i}$: $\hat{\mathbf{a}}_\mathbf{i} = \mathbf{w_i V} + \bar{\mathbf{a}}$

PCA for Dimensionality Reduction

- PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components
  - Lower-dimensional projection of the data that preserves the maximal data variance

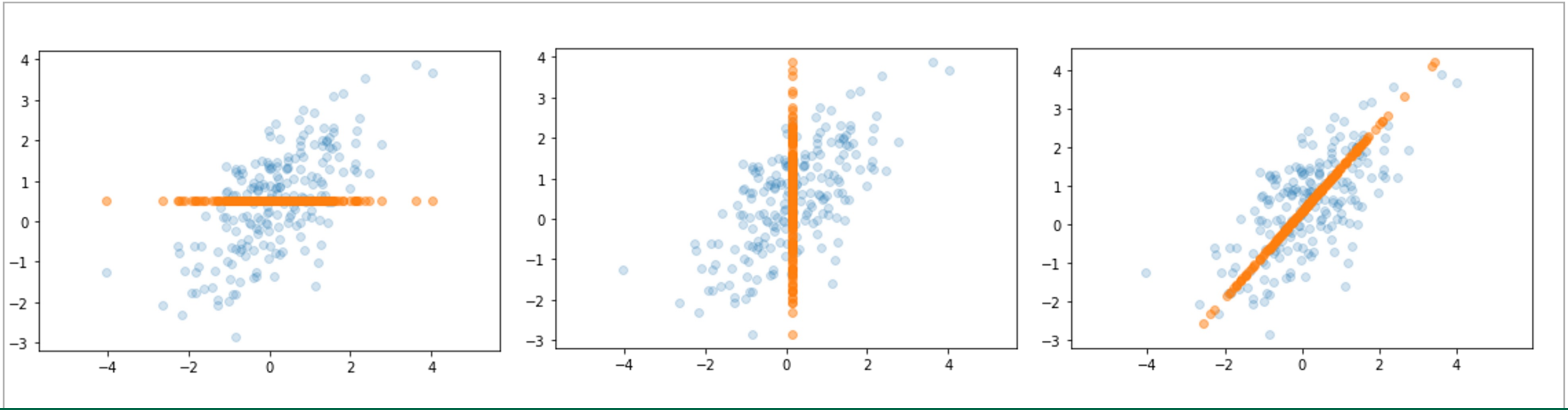

Figure 2. Reconstruction results if we discard the second axis of the original data (left), the first axis of the original data (center), and the second axis of the PCA-transformed data (right).

PCA for Dimensionality Reduction

- The information along the least important principal axis or axes is removed, leaving only the component(s) of the data with the highest variance
- The fraction of variance that is cut out is roughly a measure of how much "information" is discarded in this reduction of dimensionality
- This reduced-dimension dataset is in some sense "good enough" to encode the most important relationships between the points

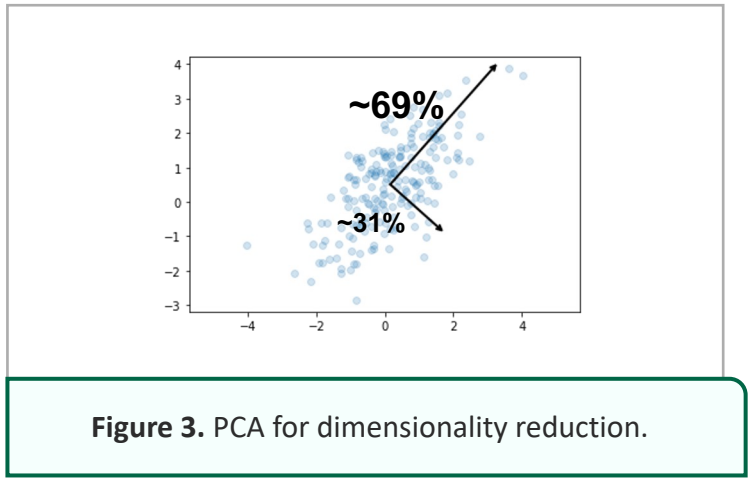

Figure 3. PCA for dimensionality reduction.

MNIST Dataset

- 10 classes
- 28x28 grayscale images
  - 784 pixels per image
  - Subset of 40 images per class

Figure 4. First 40 images from each MNIST class.

Converting Images into Arrays

- A 3x3 image becomes an array of 9 values
- A 28x28 image becomes an array of 784 values

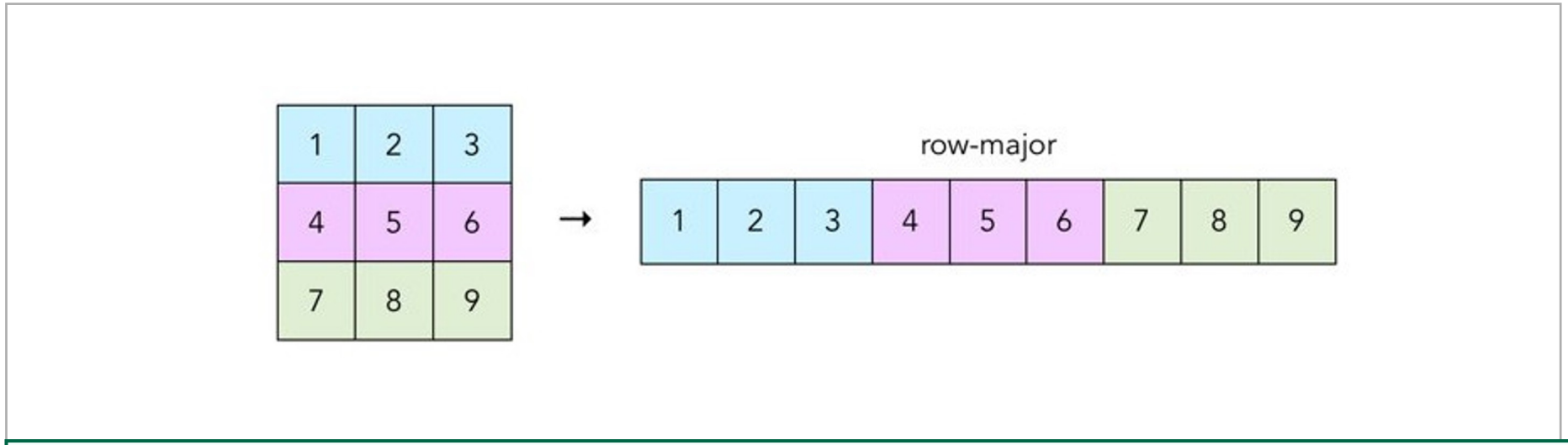

Figure 5. Converting images to arrays through row concatenation.

PCA for Visualization

- Apply PCA to project data to a more manageable number of dimensions
  - Convert a 784-dimensional image into a 2-dimensional point

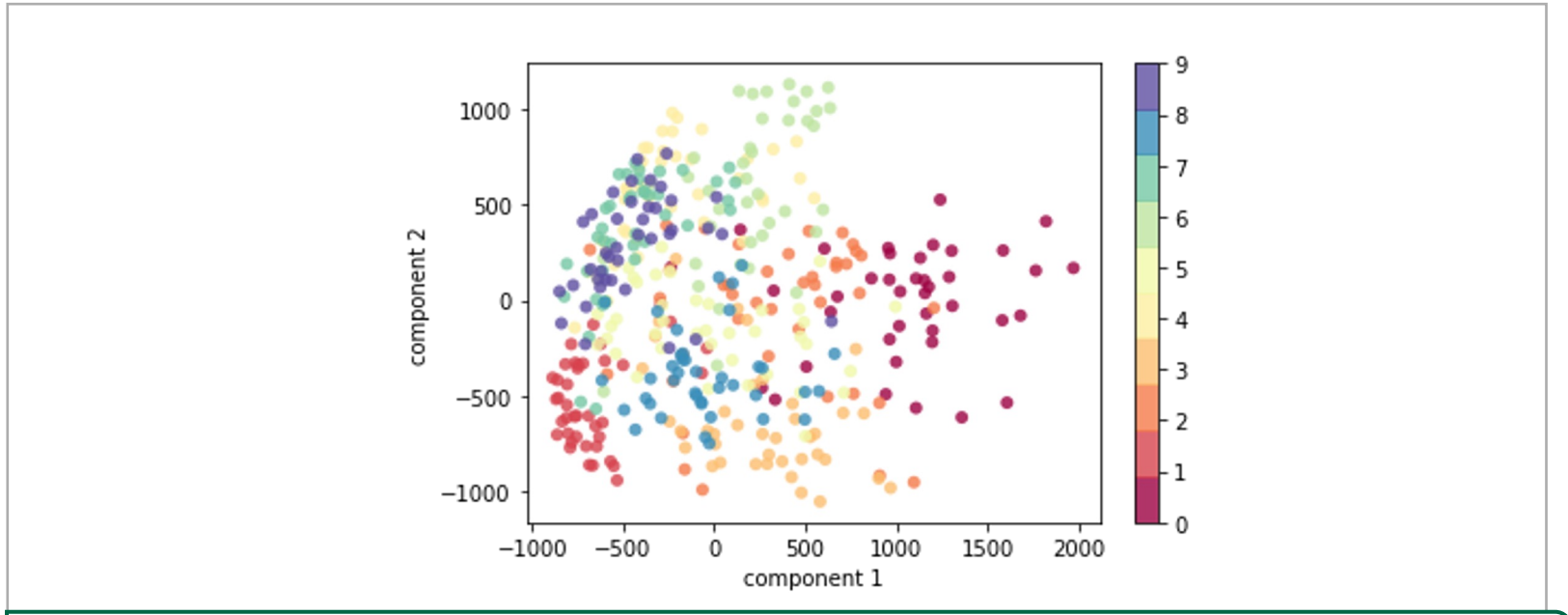

Figure 6. 2-dimensional projection of the MNIST images.

PCA for Visualization

- Apply PCA to project data to a more manageable number of dimensions
  - Convert a 784-dimensional image into a 2-dimensional point

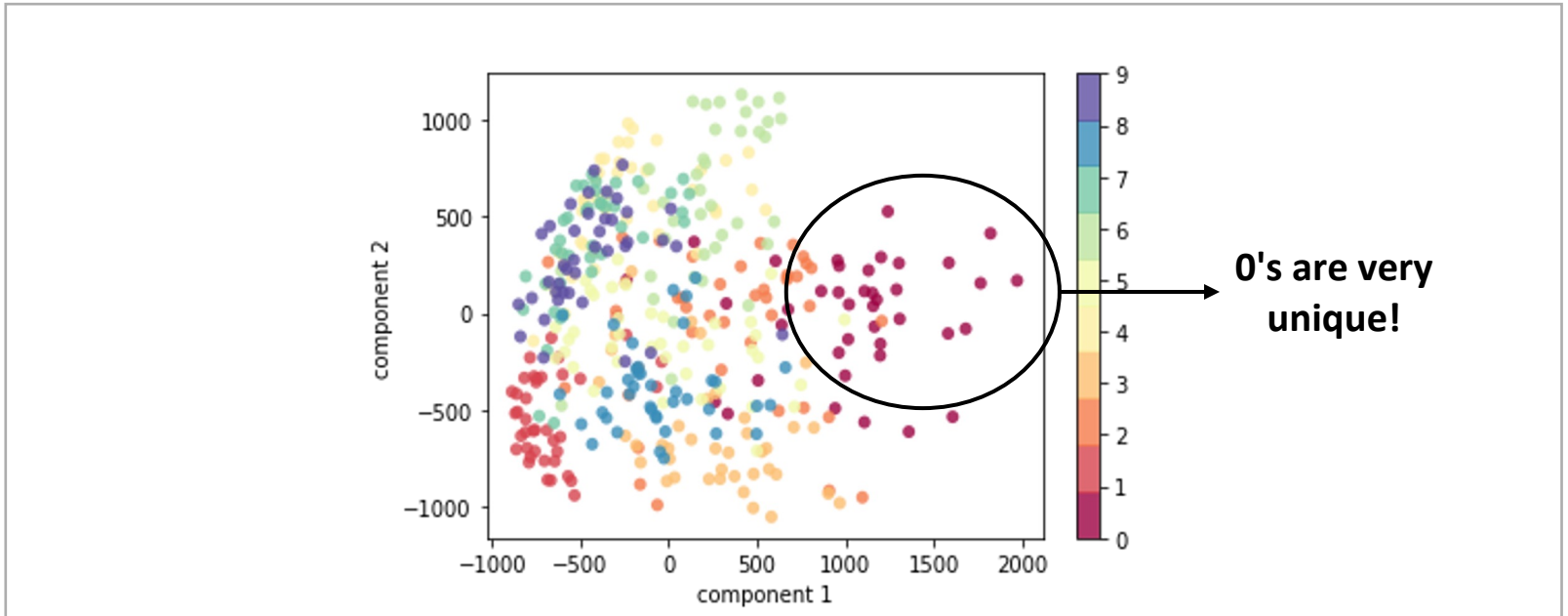

Figure 6. 2-dimensional projection of the MNIST images.

PCA for Visualization

- Apply PCA to project data to a more manageable number of dimensions
  - Convert a 784-dimensional image into a 2-dimensional point

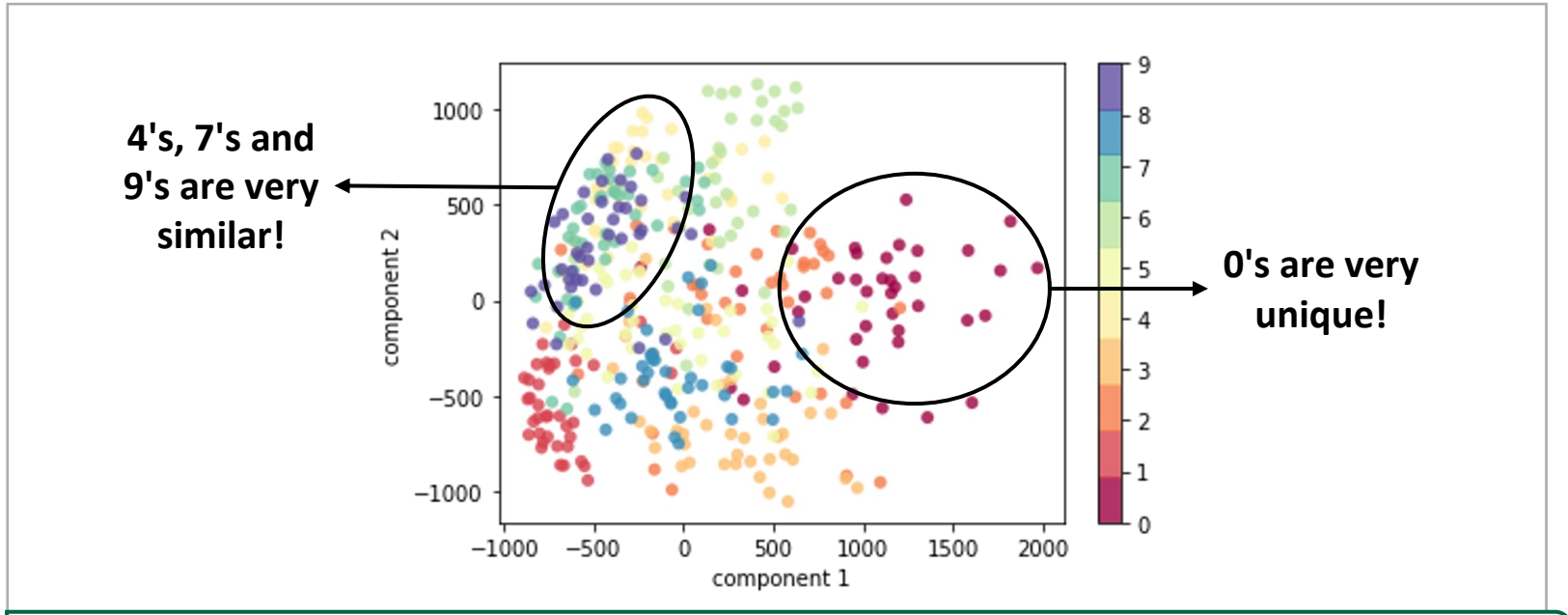

Figure 6. 2-dimensional projection of the MNIST images.

PCA for Noise Filtering

- Any components with variance much larger than the effect of the noise should be relatively unaffected by the noise
- If we reconstruct the data using just the largest principal components, we will potentially keep the signal and throw out the noise

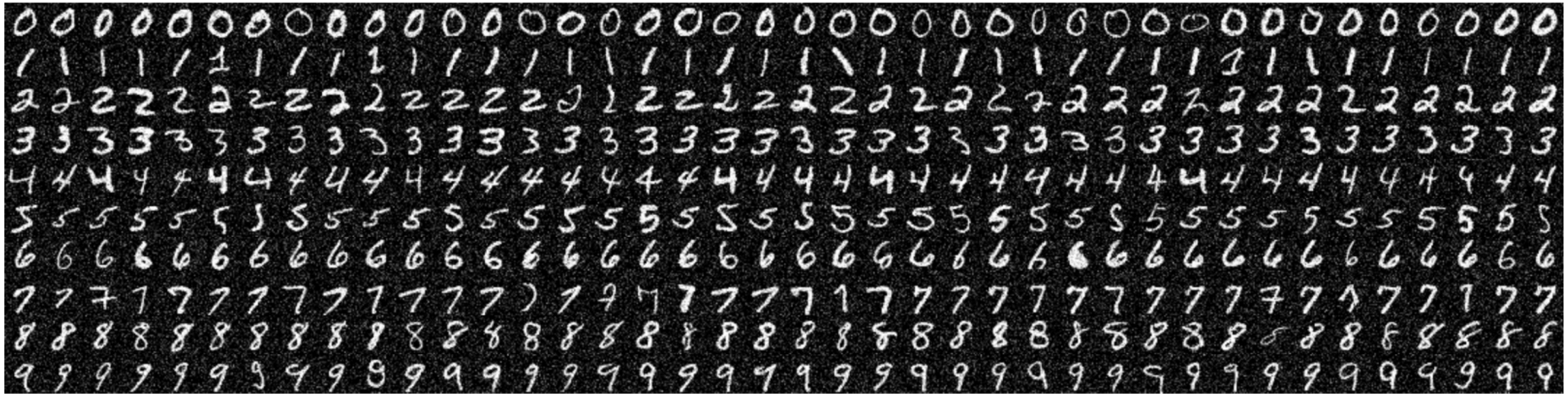

Figure 7. MNIST images with random noise.

PCA for Noise Filtering

- Let's train a PCA on the noisy data, requesting that the projection preserve 50% of the variance
  - 26 principal components

Figure 8. Filtered MNIST images with random noise.

PCA for Single-class Classification

- Use Mahalanobis distance to decide whether a probe sample is close enough to the training distribution or not
  - Given a probe sample **p**, project it to the training distribution using PCA to get **w**
  - $\mathbf{d(w) = \left(\sum_j w[j]^2 \,/\, \lambda_j\right)^{1/2}}$
  - If **d(w) < threshold**, **p** belongs to the training distribution

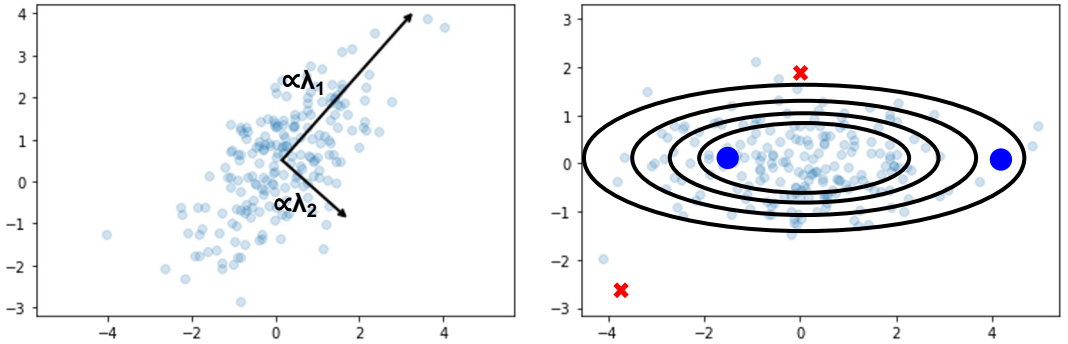

Figure 9. Illustration of the training distribution learned using PCA.

How to Evaluate the Results?

- Classification
  - Accuracy: number of correctly classified samples / total number of samples
    - Biased towards the most frequent class
  - F-score: harmonic mean of precision and recall
    - F-score = (2×precision×recall)/(precision+recall)

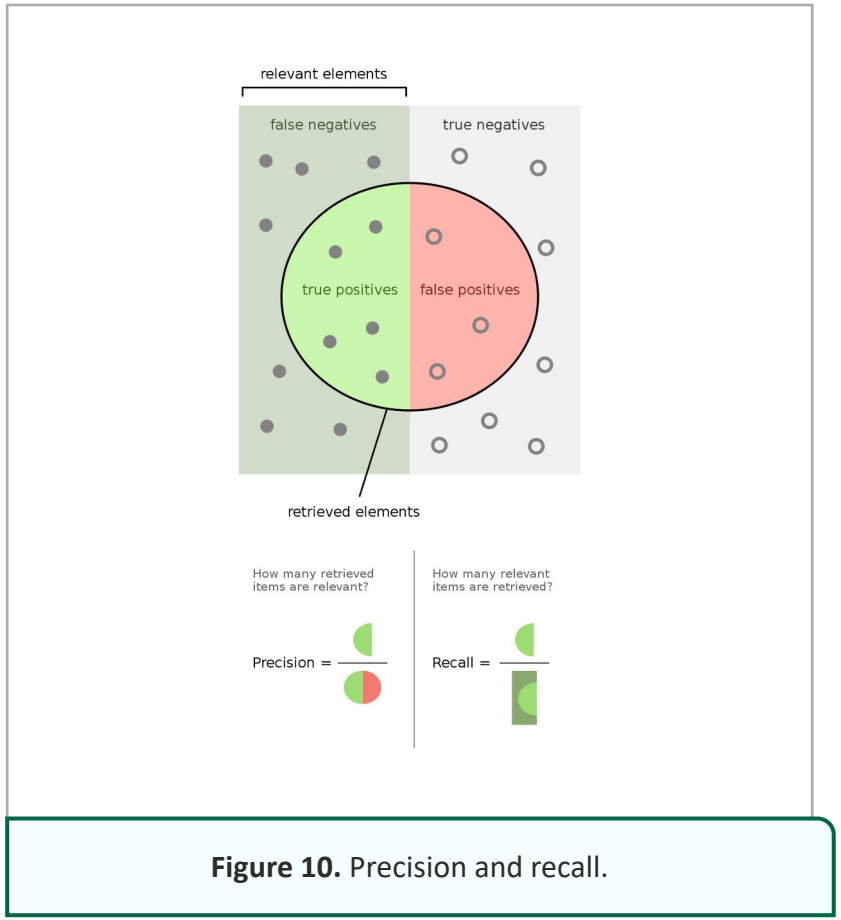

Figure 10. Precision and recall.

Knowledge Check 1

You trained a PCA model to learn the distribution of images with the digit 0. Then you tested this model using 100 images of 0's and 100 images with other digits. You observed that:

I) 87 out of 100 images of 0's were correctly classified by the model

II) 5 out of the 100 images with other digits were misclassified as 0 by the model

What is the precision and recall of the model?

A. 87% and 94.6%

B. 94.6% and 87%

C. 88% and 95%

D. 95% and 88%

How to Do a Proper Evaluation?

- Split available data into training, validation and test sets
- Use training set for model creation
- Use validation set for hyperparameter tuning
  - Example: use it to find the best number of principal components for PCA
- Use test set to report results in unknown data

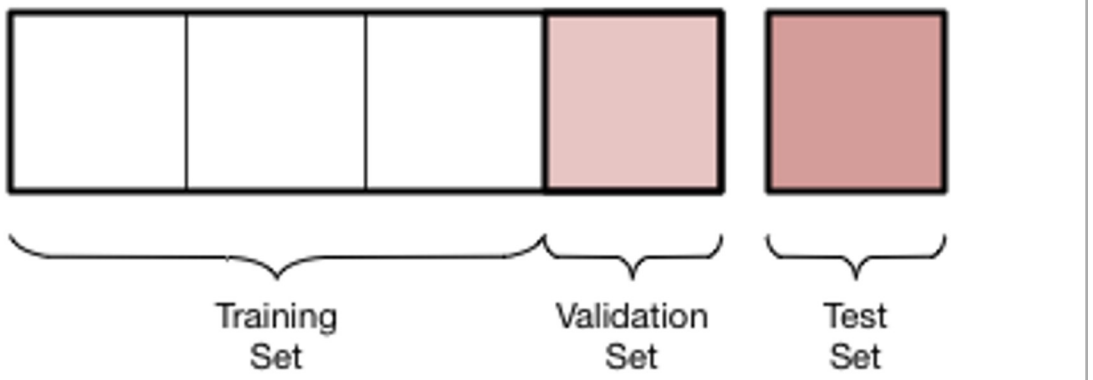

Figure 12. Process for proper evaluation.

Scikit-learn: Data Manipulation

We can use NumPy!

scikit-learn: PCA Implementation

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

```
sklearn.decomposition.PCA(
        n_components=None,
        copy=True,
        whiten=False,
        svd_solver='auto',
        tol=0.0,
        iterated_power='auto',
        n_oversamples=10,
        power_iteration_normalizer='auto',
        random_state=None
)
```

Parameters::

n_components : *int, float or 'mle', default=None*

Number of components to keep. if n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

If `n_components == 'mle'` and `svd_solver == 'full'`, Minka's MLE is used to guess the dimension. Use of `n_components == 'mle'` will interpret `svd_solver == 'auto'` as `svd_solver == 'full'`.

If `0 < n_components < 1` and `svd_solver == 'full'`, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

If `svd_solver == 'arpack'`, the number of components must be strictly less than the minimum of n_features and n_samples.

Hence, the None case results in:

```
n_components == min(n_samples, n_features) - 1
```

Attributes::

components_ : *ndarray of shape (n_components, n_features)*

Principal axes in feature space, representing the directions of maximum variance in the data. Equivalently, the right singular vectors of the centered input data, parallel to its eigenvectors. The components are sorted by `explained_variance_`.

explained_variance_ : *ndarray of shape (n_components,)*

The amount of variance explained by each of the selected components. The variance estimation uses `n_samples - 1` degrees of freedom.

Equal to n_components largest eigenvalues of the covariance matrix of X.

New in version 0.18.

explained_variance_ratio_ : *ndarray of shape (n_components,)*

Percentage of variance explained by each of the selected components.

If `n_components` is not set then all components are stored and the sum of the ratios is equal to 1.0.

Figure 13. scikit-learn: PCA implementation.

scikit-learn: PCA Implementation

- Methods

`fit`(X[, y])	Fit the model with X.
`fit_transform`(X[, y])	Fit the model with X and apply the dimensionality reduction on X.
`get_covariance`()	Compute data covariance with the generative model.
`get_feature_names_out`([input_features])	Get output feature names for transformation.
`get_params`([deep])	Get parameters for this estimator.
`get_precision`()	Compute data precision matrix with the generative model.
`inverse_transform`(X)	Transform data back to its original space.
`score`(X[, y])	Return the average log-likelihood of all samples.
`score_samples`(X)	Return the log-likelihood of each sample.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X)	Apply dimensionality reduction to X.

Figure 14. scikit-learn: PCA implementation.

You have reached the end of the lecture.

Image/Figure References

Figure 1. Original two-dimensional data (left), principal components extracted using PCA (center), and transformed data (right). Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 2. Reconstruction results if we discard the second axis of the original data (left), the first axis of the original data (center), and the second axis of the PCA-transformed data (right). Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 3. PCA for dimensionality reduction. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 4. First 40 images from each MNIST class. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 5. Converting images to arrays through row concatentation. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 6. 2-dimensional projection of the MNIST images. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 7. MNIST images with random noise. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 8. Filtered MNIST images with random noise. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 9. Illustration of the training distribution learned using PCA. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 10. Precision and recall. Source: https://en.wikipedia.org/wiki/Precision_and_recall#/media/File:Precisionrecall.svg
Figure 11. MNIST images. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 12. Process for proper evaluation. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 13. scikit-learn: PCA implementation. Source: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
Figure 14. scikit-learn: PCA implementation. Source: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html