

# COSC 325: Introduction to Machine Learning

Dr. Hector Santos-Villalobos



THE UNIVERSITY OF  
TENNESSEE  
KNOXVILLE

# Lecture 16: Feature Selection and Extraction



THE UNIVERSITY OF  
TENNESSEE  
KNOXVILLE



# Class Announcements

## Homework

Homework #5 due 11/06

Homework #6 due 11/13

## Course Project:

Midterm Report due 10/27

## Lectures:

Tuesday 10/22, In-Person

## Quizzes:

Weekly quiz as usual.

## Exams:

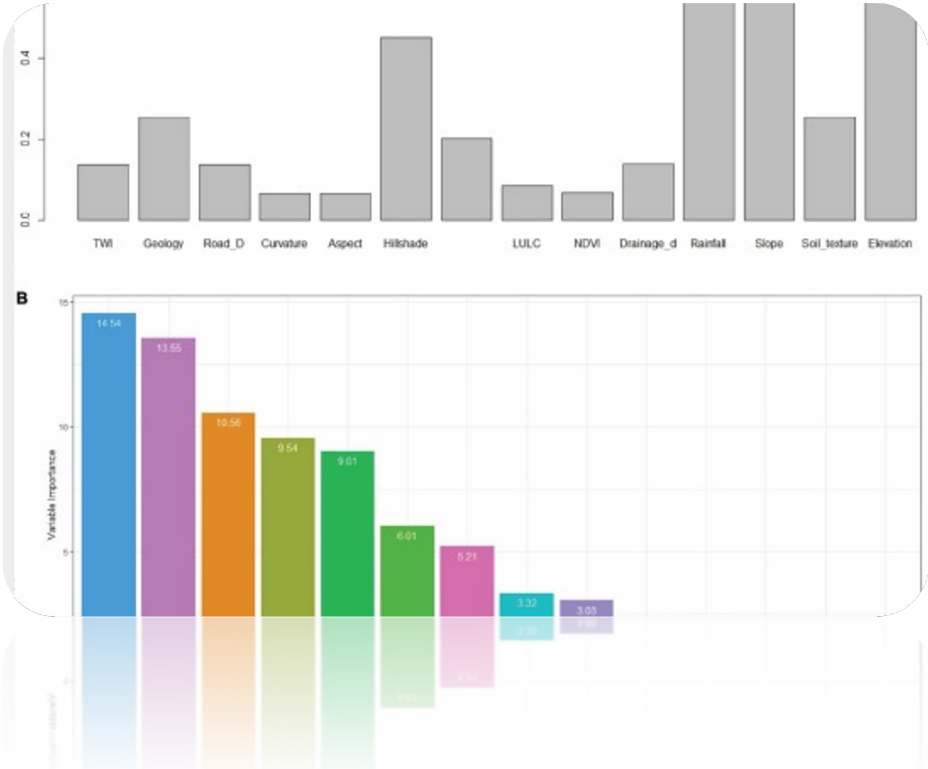
Next exam 11/21. Same format.

# Review

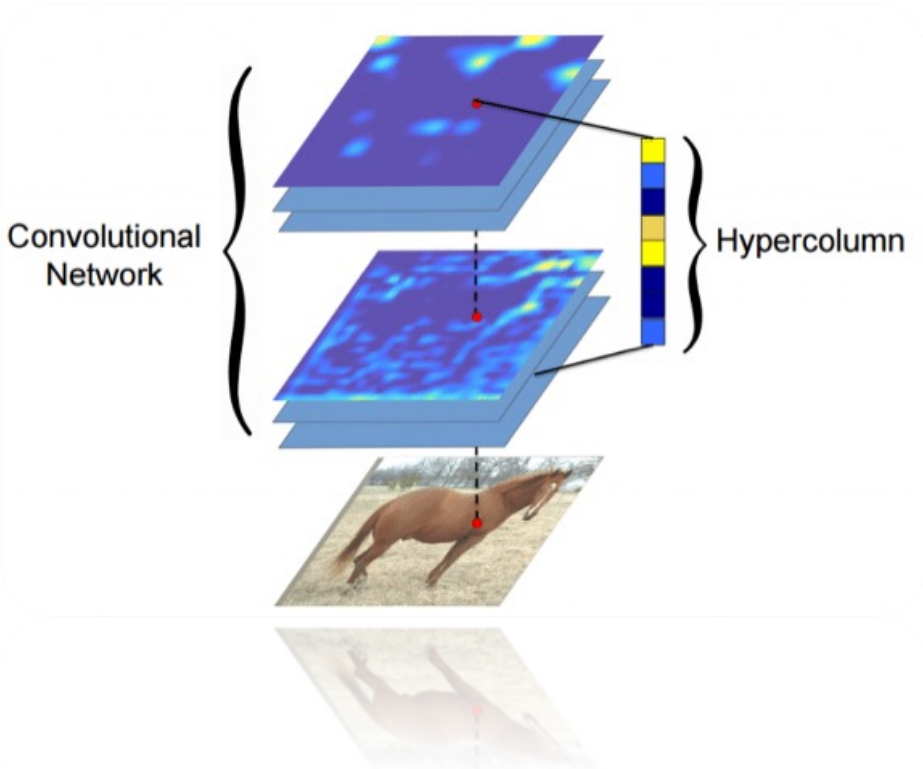
- Hyperparameter tuning
  - Grid Search
  - Random Search -> 3+ parameters
  - Scale search space (i.e., logscale)
- Model evaluation
  - Measuring performance
    - Holdout Method
    - Repeated Holdout (with Bootstrapping)
      - LOOB, OOBB
    - K-Fold Cross-Validation
      - LOOCV (n-Fold) (Small), 10-Fold (Avg), 5-Fold (Large)
  - Model evaluation
    - Accuracy metrics (Mean, Variance)
    - Confidence intervals
  - Sampling
    - Stratified sampling: ensure data splits class distributions match the original dataset class distributions.
    - Bootstrapping (w/ replacement, 34% not sampled)

# Today's Topics

## Feature Selection



## Feature Extraction



# Terminology Check

- **Exhaustive search:** The algorithm evaluates all possible combinations and guarantees an optimal solution. Computational expensive.
- **Greedy search:** makes locally optimal choices at each stage of a combinatorial search problem and generally yields a suboptimal solution. Computational efficient.

# Terminology Check

- **Feature selection:** keeping the best or most important features for machine algorithm training and execution performance. Besides feature scaling, denoising, and encoding, the original feature intent and nature remain.
- **Feature extraction:** transforming the original features into a new feature space that retains the essential information about the original dataset.

# Feature Selection



THE UNIVERSITY OF  
**TENNESSEE**  
KNOXVILLE





# Feature Selection

- **Lasso Regularization**
- Sequential Backward Selection
- Random Forest Feature Selection
- Permutation Feature Importance

# Lasso Regularization

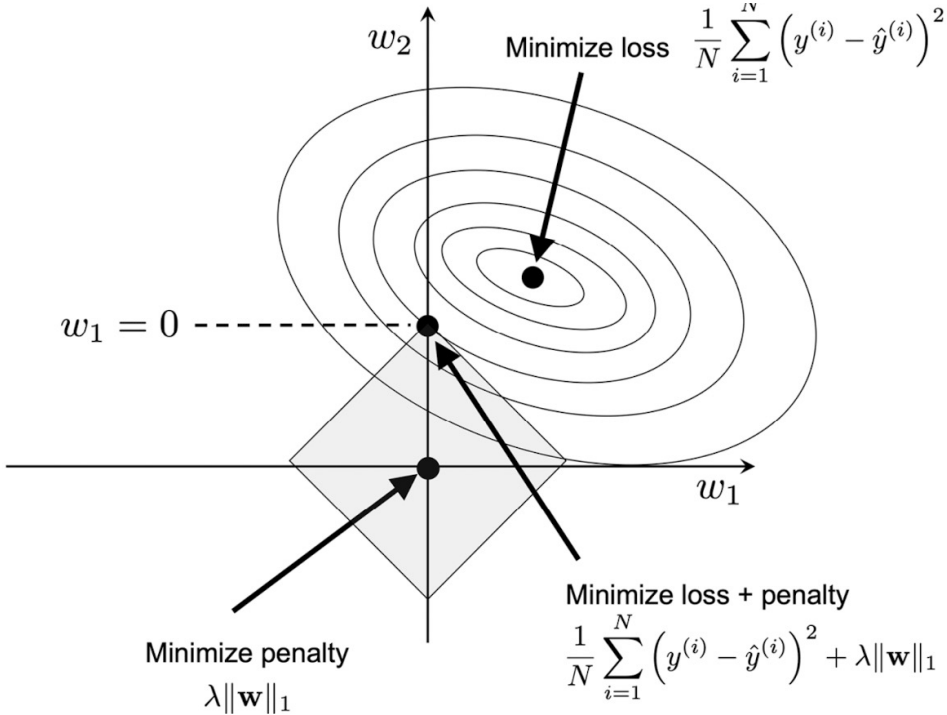
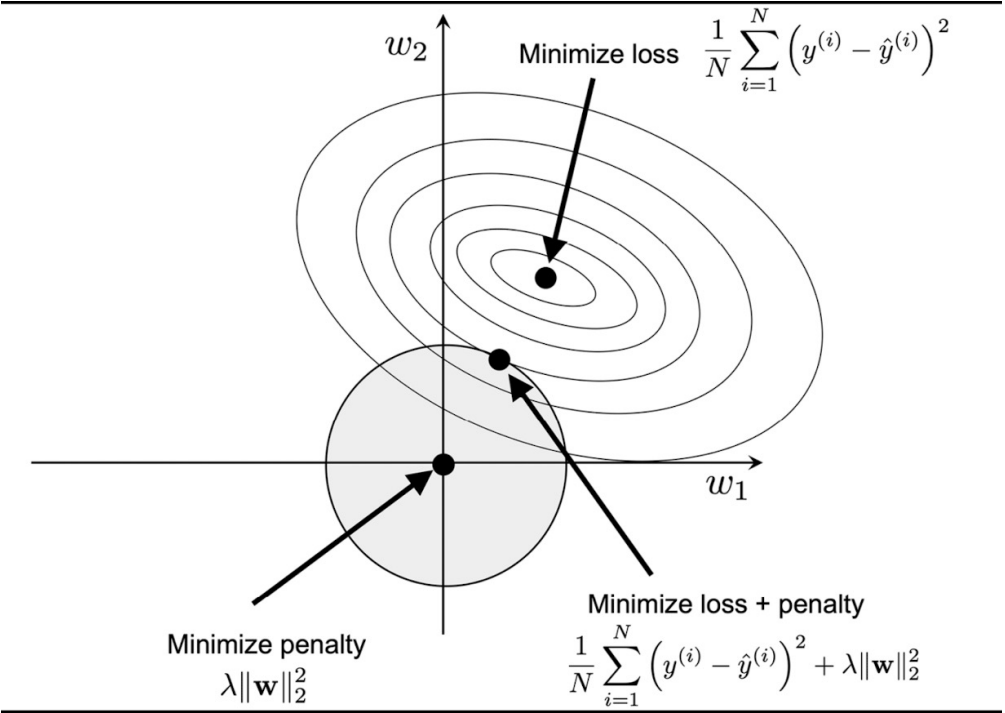
$$\text{Lasso (L1): } \|w\|_1 = \frac{1}{m} \sum_{j=1}^m |w_j| \quad \left| \quad \text{Ridge (L2): } \|w\|_2^2 = \frac{1}{m} \sum_{j=1}^m w_j^2$$

- Easily integrable in gradient-based ML methods via regularization term in the cost function.

$$\min_{w,b} J(w) = -\frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \lambda \text{Lasso}(w)$$

- Optimizes when weights are zero, resulting in **sparsity**
- Excellent choice for high-dimensional datasets

# Lasso and Feature Sparsity



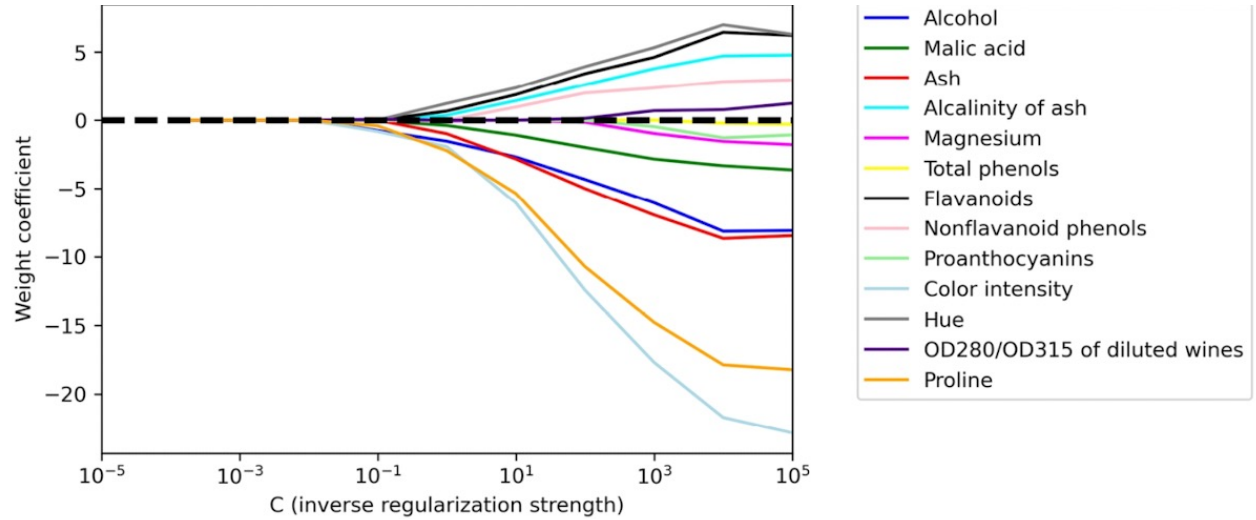
Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn".

# Regularization Coefficient vs Weight Sparsity

$$\lambda \propto \frac{1}{C}$$

```
>>> lr = LogisticRegression(penalty='l1',
...                          C=1.0,
...                          solver='liblinear',
...                          multi_class='ovr')
>>> # Note that C=1.0 is the default. You can increase
>>> # or decrease it to make the regularization effect
>>> # stronger or weaker, respectively.
>>> lr.fit(X_train_std, y_train)
>>> print('Training accuracy:', lr.score(X_train_std, y_train))
Training accuracy: 1.0
>>> print('Test accuracy:', lr.score(X_test_std, y_test))
Test accuracy: 1.0
```

Wine Dataset Logistic Regression Coefficients



$$\min_{w,b} J(w) = -\frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \lambda Lasso(w)$$

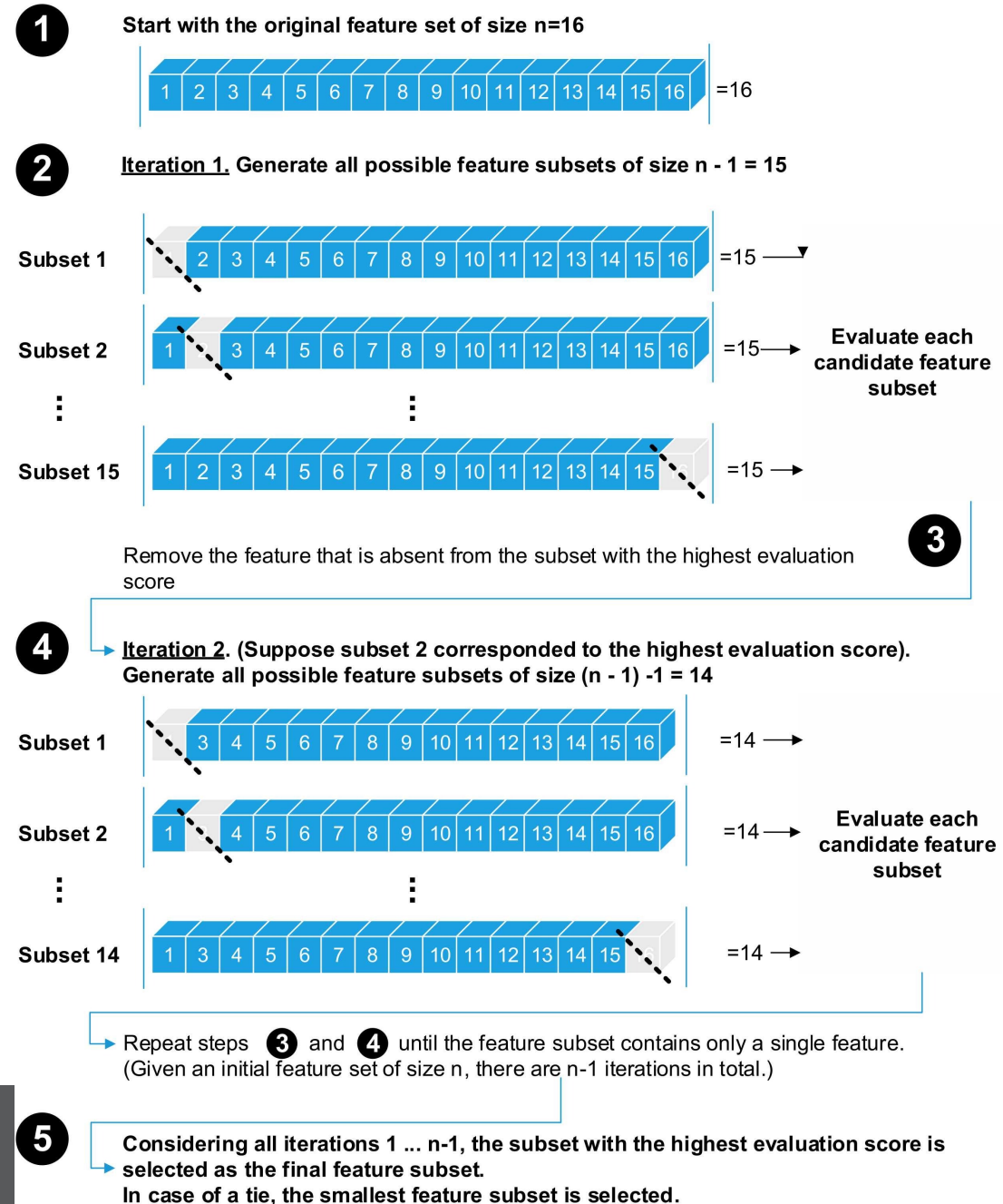
Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn"

# Feature Selection

- Lasso Regularization
- **Sequential Backward Selection**
- Random Forest Feature Selection
- Permutation Feature Importance

# Sequential Backward Selection (SBS)

- Classic heuristic technique to programmatically reduce the dimensionality of the original model parameters (i.e., features) space with minimal decay in model performance.
  - For overfitting models, SBS can improve performance.
- At each  $k$  trials, we remove the least important feature.



# Other techniques

- **Sequential Feature Selection (Forward):** conceptually similar to SBS. Starts with no features, then adds one feature at a time. (Homework 5)
- **Backward Elimination:** eliminates features based on the feature weights.
- **Tree-based methods:** selects features based by importance and statistical tests.

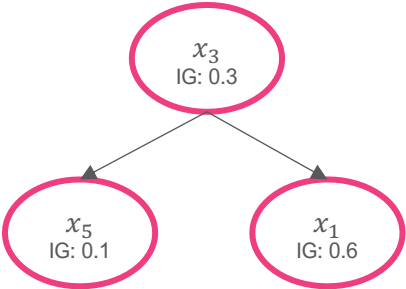
# Feature Selection

- Lasso Regularization
- Sequential Backward Selection
- **Random Forest Feature Selection**
- Permutation Feature Importance



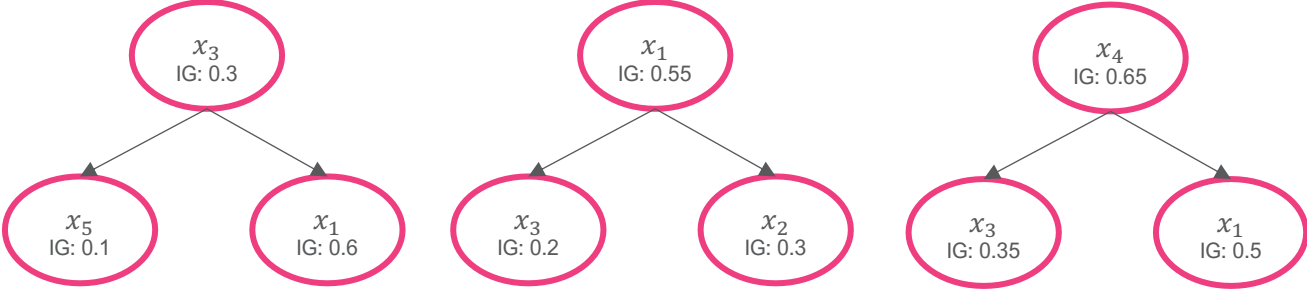
# Feature Reduction with Random Forests

- Random Forest
  - Fixed tree architecture
  - Randomly assign features to each internal node
  - Ensemble of hundreds of random trees



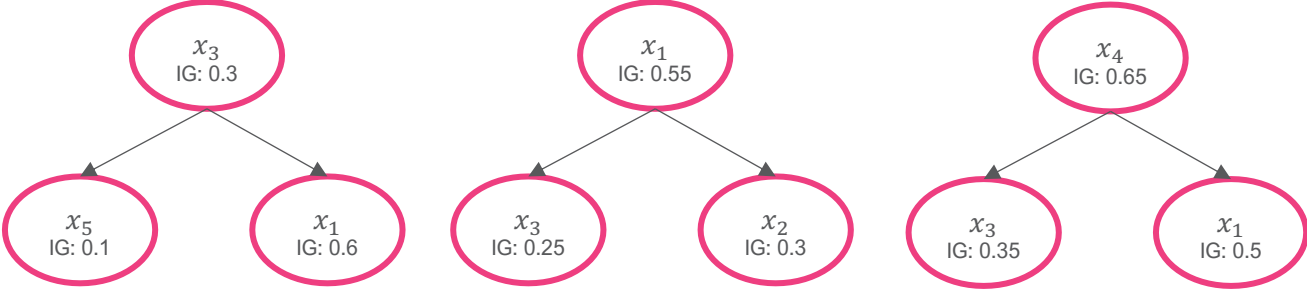
# Feature Reduction with Random Forests

- Random Forest
  - Fixed tree architecture
  - Randomly assign features to each internal node
  - Ensemble of hundreds of random trees
- Feature importance
  - Average impurity decrease (i.e., information gain)



# Feature Reduction with Random Forests

- Random Forest
  - Fixed tree architecture
  - Randomly assign features to each internal node
  - Ensemble of hundreds of random trees
- Feature importance
  - Average impurity decrease (i.e., information gain)

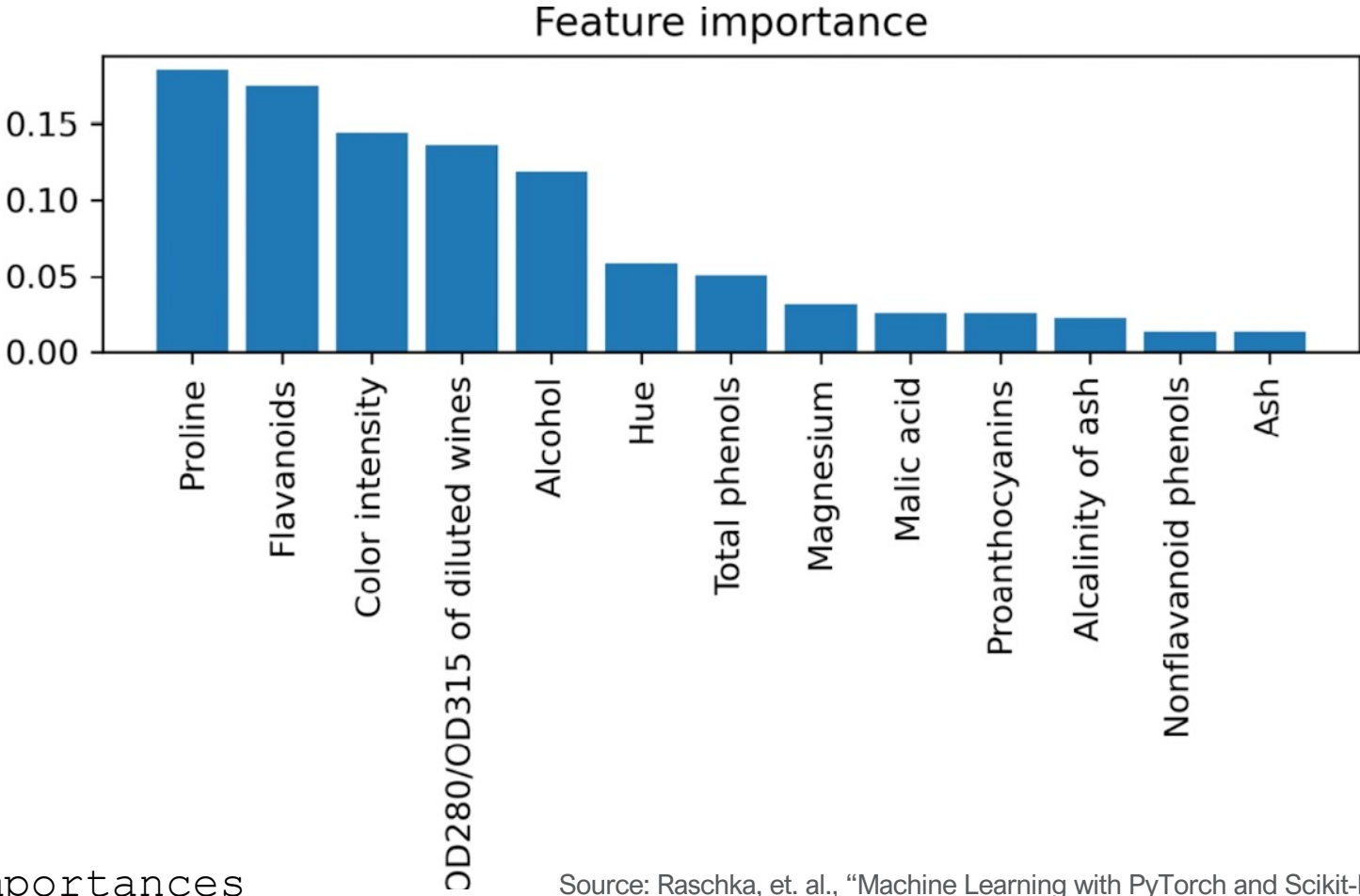


Average information gain:

- $\overline{IG}_{x_1} = 0.55$
- $\overline{IG}_{x_3} = 0.30$

# RF Feature Important Example

- Careful with high-dimensional data
  - Too many unique values
- Computed with training data
  - Optimistic metric
- Inverted importance for highly correlated features.



```
SKLearn RandomForestClassifier  
importances = forest.feature_importances_
```

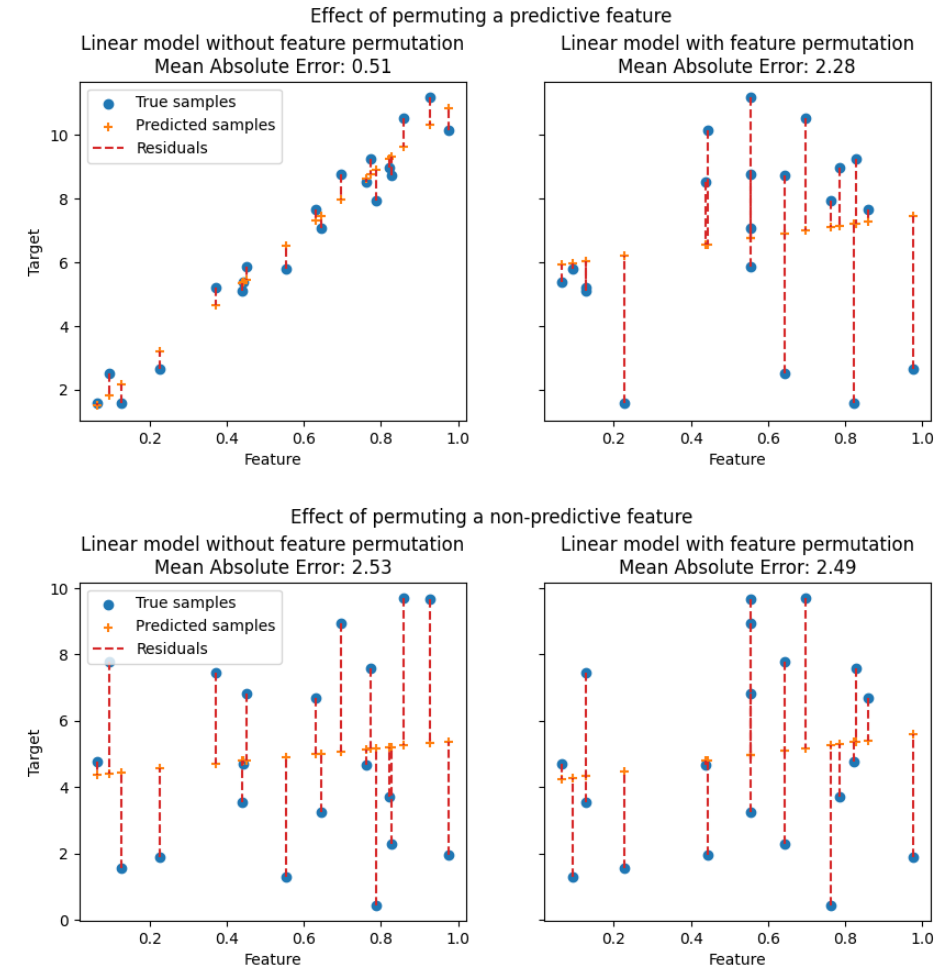
Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn"

# Feature Selection

- Lasso Regularization
- Sequential Backward Selection
- Random Forest Feature Selection
- **Permutation Feature Importance**

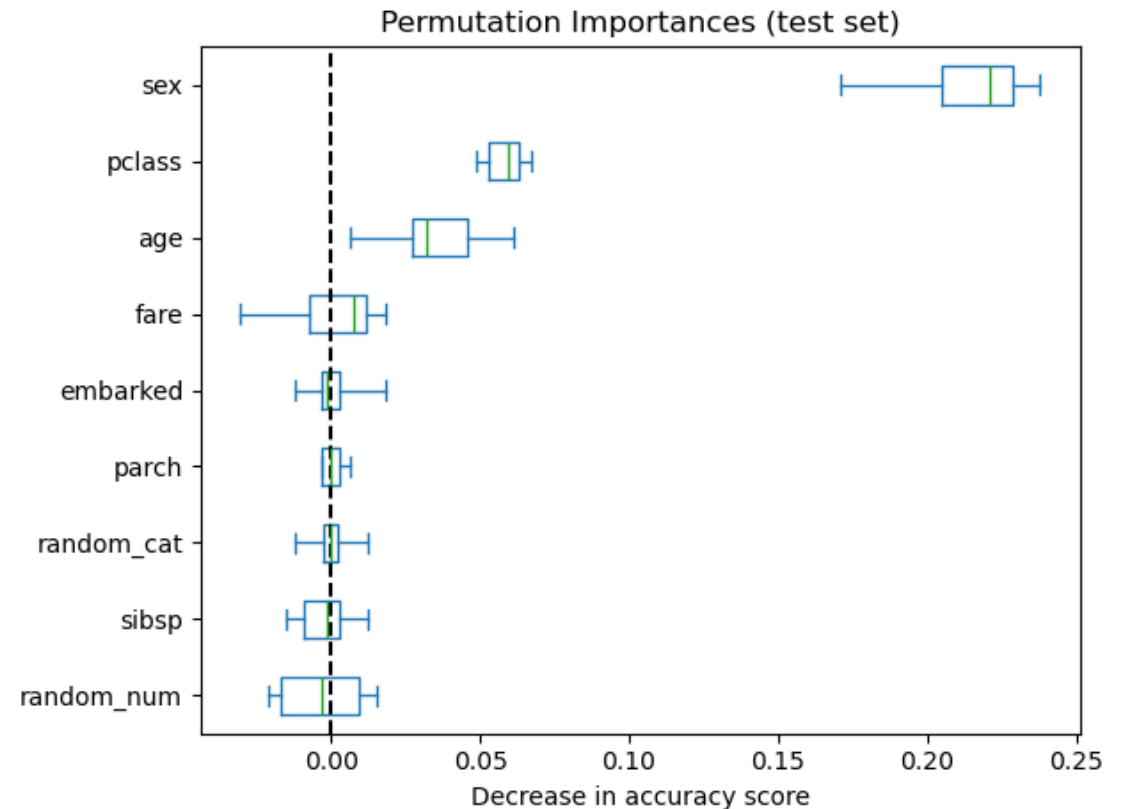
# Permutation Feature Importance

- **Operation:** Shuffle feature  $x_j$  and observe impact on model performance
- Works with tabular data
- Measured for training or validation set
- Useful for non-linear, black-box estimators
- Issues with highly correlated features



# Permutation Feature Importance

- **Operation:** Shuffle feature  $x_j$  and observe impact on model performance
- Works with tabular data
- Measured for training or validation set
- Useful for non-linear, black-box estimators
- Issues with highly correlated features



# Feature Selection Review

- Lasso Regularization
- Sequential Backward Selection
- Random Forest Feature Selection
- Permutation Feature Importance



# Pop Quiz

True or False. Feature Selection is the process of enhancing the original data to make it easier for the ML technique to select and process the features during training.

A. True

B. False



# Feature Extraction



THE UNIVERSITY OF  
**TENNESSEE**  
KNOXVILLE



# Dimensionality Reduction

- Transforms original data onto a new feature subspace with lower dimensionality.
  - Other names: data summarization or compression.
- Benefits
  - Reduce storage space requirements
  - Increase computational efficiency of the ML algorithm
  - Improve predictive performance
    - Address the curse of dimensionality

# Curse of dimensionality

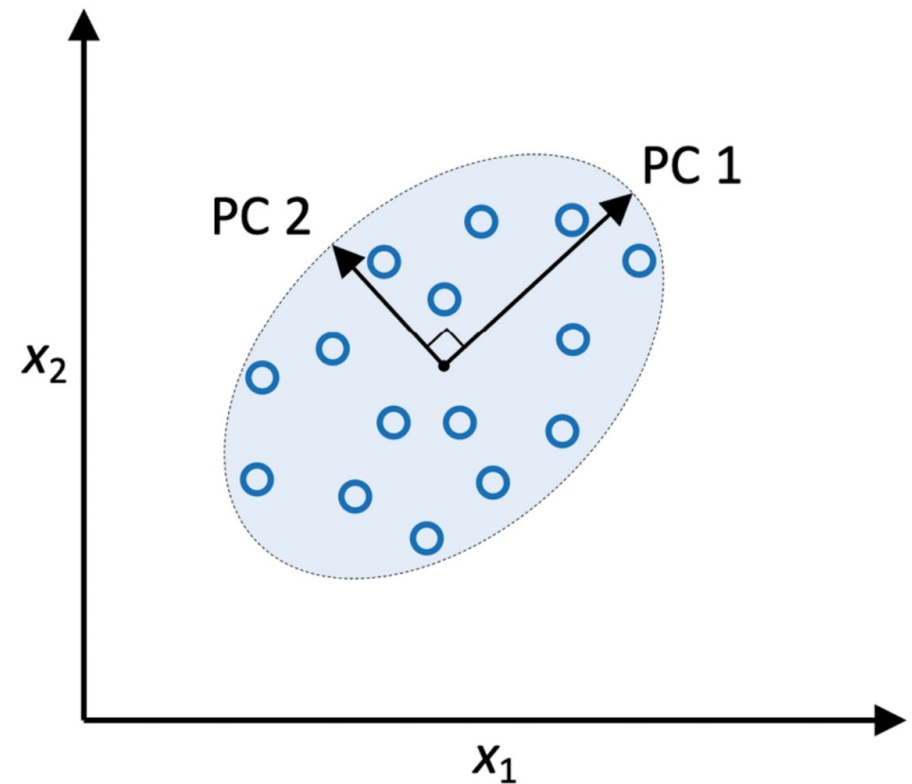
- High-dimensional data is typically sparse
- This leads to model overfitting and poor generalization

# Feature Extraction

- **Principal Component Analysis (PCA)**
- Linear Discriminant Analysis (LDA)
- t-distributed stochastic neighbor embedding (t-SNE)

# Principal component analysis (PCA)

- Unsupervised linear transformation technique that is widely used across different fields.
- Popular applications:
  - Dimensionality reduction
  - Feature extraction
  - Exploratory data analysis
  - Denoising of timeseries signals
  - Genome data and gene expression analysis



Source: Raschka, et. al., “Machine Learning with PyTorch and Scikit-Learn”

# Objective of PCA

- We have a dataset sample  $x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)}]$ ,  $x^{(i)} \in \mathbb{R}^m$

- We want to find a projection matrix  $W \in \mathbb{R}^{m \times k}$ , where
$$x^{(i)}W = z^{(i)}$$

- Resulting in the new vector

$$z^{(i)} = [z_1^{(i)}, z_2^{(i)}, \dots, z_k^{(i)}], \quad z^{(i)} \in \mathbb{R}^k$$

- The output vector is a lower dimensional feature space with  $k \ll m$

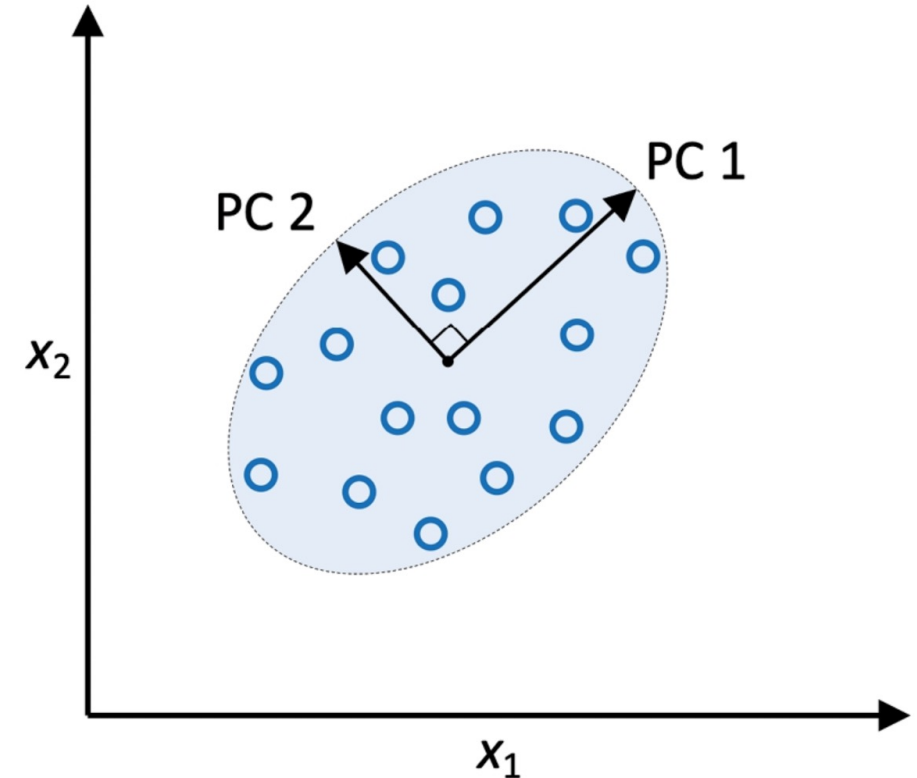
# PCA Algorithm

1. Standardize original feature data  $x$  (Note: Assumes normality)
2. Construct features covariance matrix  $\Sigma$
3. Decompose  $\Sigma$  in eigenvectors and eigenvalues
4. Select the  $k$  most important eigenvectors
  1. Sort eigenvalues in decreasing order
  2. Use eigenvalues order to rank eigenvector importance
5. Construct projection matrix  $W$
6. Transform original feature data  $x$  with  $W$  to generate  $z$



# Eigendecomposition Review

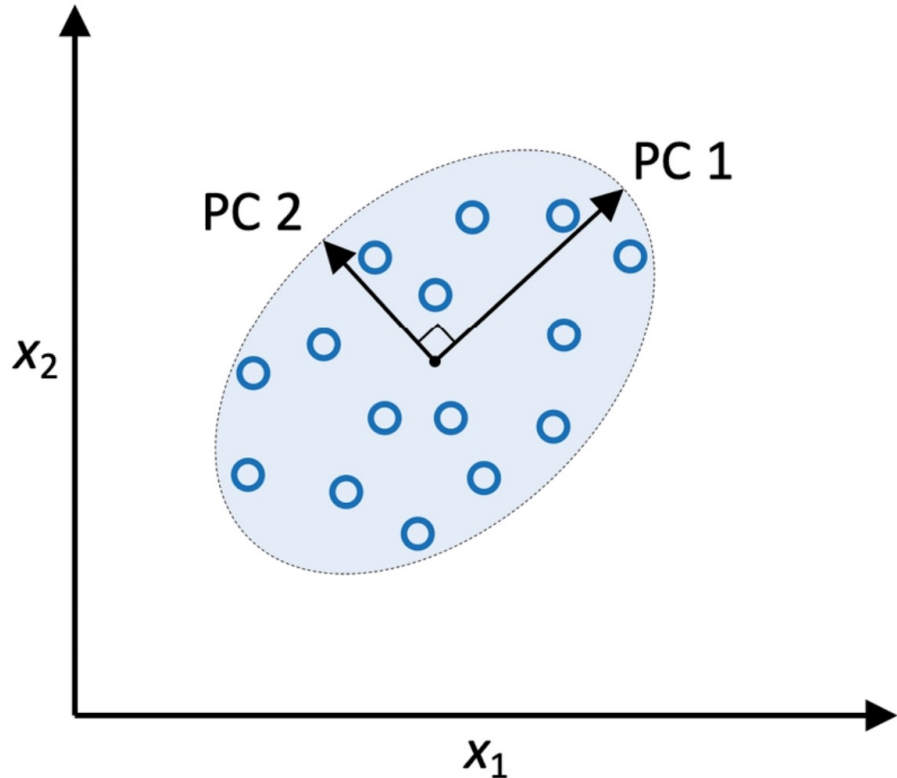
- Eigendecomposition is the factorization of a square matrix into eigenvalues and eigenvectors
  - Eigenvalues and eigenvectors come in pairs.
    - Eigenvalues: variance magnitude
    - Eigenvector: direction of variance (unchanged under linear transforms)



Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn"

# Eigendecomposition Review

- Eigendecomposition is the factorization of a square matrix into eigenvalues and eigenvectors
  - Eigenvalues and eigenvectors come in pairs.
    - Eigenvalues: variance magnitude
    - Eigenvector: direction of variance (unchanged under linear transforms). Also known as Principal Components (PC)
- The covariance matrix is **square** and **symmetric**,  $\Sigma = \Sigma^T$
- Such a matrix produces
  - Real eigenvalues
  - Orthogonal eigenvectors
- The eigenvector with the largest eigenvalue points in the direction of the largest variance in the data



Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn"

## Step 2: Covariance Construction

- The covariance between two feature vectors  $x_j$  and  $x_l$  is given by

$$\sigma_{jk} = \frac{1}{n-1} \sum_{i=1}^n \left\{ \left( x_j^{(i)} - \mu_j \right) \left( x_l^{(i)} - \mu_l \right) \right\},$$

- $\mu_j$  and  $\mu_l$  are the sample means for features  $j$  and  $l$ , respectively
  - Note that they are zero after standardization
- Example of a covariance matrix for dataset with three features

$$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}$$

# Step 3: Eigendecomposition

- We want to find the eigenvalues  $\lambda$  and eigenvectors  $v$  that satisfy the following equation

$$\Sigma v = \lambda v$$

- Computing these values and vectors is beyond this course. For those interested in the theory, please visit the following 3Blue1Brown video tutorial, “Eigenvectors and eigenvalues”
  - Link: <https://www.youtube.com/watch?v=PFDu9oVAE-g>

# Step 3: Eigendecomposition

- We want to find the eigenvalues  $\lambda$  and eigenvectors  $v$  that satisfy the following equation

$$\Sigma v = \lambda v$$

- Computing these values and vectors is beyond this course. For those interested in the theory, please visit the following 3Blue1Brown video tutorial, “Eigenvectors and eigenvalues”
  - Link: <https://www.youtube.com/watch?v=PFDu9oVAE-g>
- We will use the Numpy library to compute eigenvalues and eigenvectors  
`eigen_vals, eigen_vectors = np.linalg.eig(cov_mat)`

# PCA Algorithm

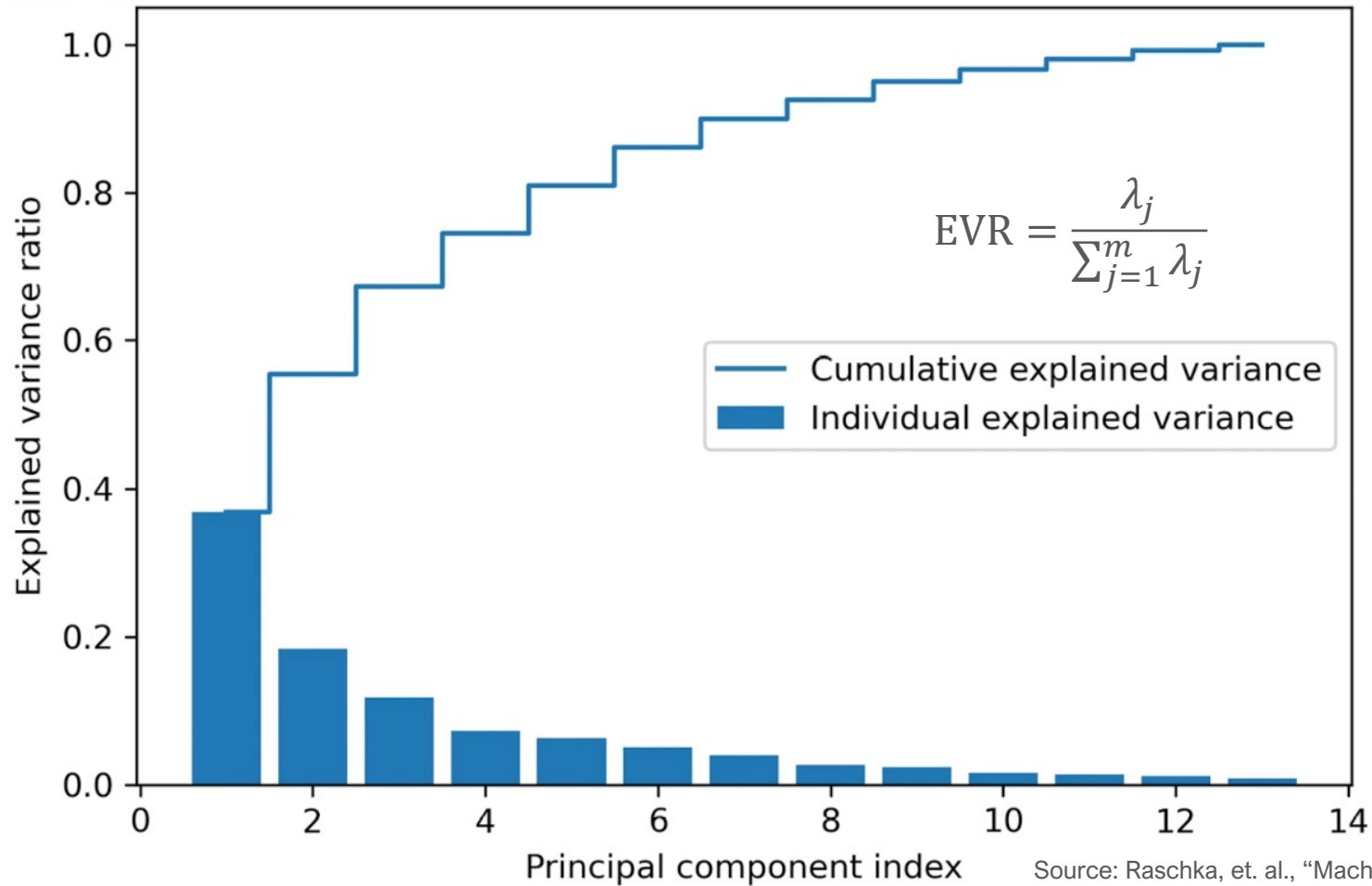
1. Standardize original feature data  $x$  (Note: Assumes normality)
2. Construct features covariance matrix  $\Sigma$
3. Decompose  $\Sigma$  in eigenvectors and eigenvalues
4. **Select the  $k$  most important eigenvectors**
  1. **Sort eigenvalues in decreasing order**
  2. **Use eigenvalues order to rank eigenvector importance**
5. Construct projection matrix  $W$
6. Transform original feature data  $x$  with  $W$  to generate  $z$

# Step 4: Select the $k$ most important eigenvectors

- Select a subset of  $k$  eigenvectors that contains most of the information (variance) (i.e., the corresponding  $k$ -largest eigenvalues)
- How do we pick  $k$ ?
  - Hyperparameter tuning
  - **Better way:** By using Total and Explained Variance
    - Compute the Explained Variance Ratio (EVR) =  $\frac{\lambda_j}{\sum_{j=1}^m \lambda_j}$
    - Compute the cumulative sum of the EVR (Use Numpy cumsum function on EVR)
    - Select the number of eigenvectors that reach the desired cumulative value.

# Step 4: Select the $k$ most important eigenvectors: EVR

SKLearn Wine dataset.



Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn"



## Step 4: Select the $k$ most important eigenvectors: Sort eigenvalues/vector

```
# Create tuples of eigenvalues and eigenvectors
eigen_data = [(np.abs(eigen_vals[i]), eigen_vecs[:,i])
               for i in range(len(eigen_vals))]

# Sort eigenvalues/vectors tuples from high to low
eigen_data.sort(key=lambda k: k[0], reverse=True)
```

# Step 5: Construct projection matrix $W$

- You have  $k$  eigenvectors  $v_1, v_2, \dots, v_k$  where  $v_j \in \mathbb{R}^m$
- The matrix  $W$  should end up with shape  $m \times k$
- Therefore,  $W = [v_1 \quad v_2 \quad \dots \quad v_k]$ 
  - You can use `np.hstack()` function

```
# Construct W with first two eigenvectors
W = np.hstack((eigen_data[0][1][:, np.newaxis],
               eigen_data[1][1][:, np.newaxis]))
```

# PCA Algorithm

1. Standardize original feature data  $x$  (Note: Assumes normality)
2. Construct features covariance matrix  $\Sigma$
3. Decompose  $\Sigma$  in eigenvectors and eigenvalues
4. Select the  $k$  most important eigenvectors
  1. Sort eigenvalues in decreasing order
  2. Use eigenvalues order to rank eigenvector importance
5. Construct projection matrix  $W$
6. **Transform original feature data  $x$  with  $W$  to generate  $z$**

# Step 6: Transform original feature data $x$ with $W$ to generate $z$

- Sample transformation:

$$z^{(i)} = x^{(i)}W$$

- Whole dataset transformation:

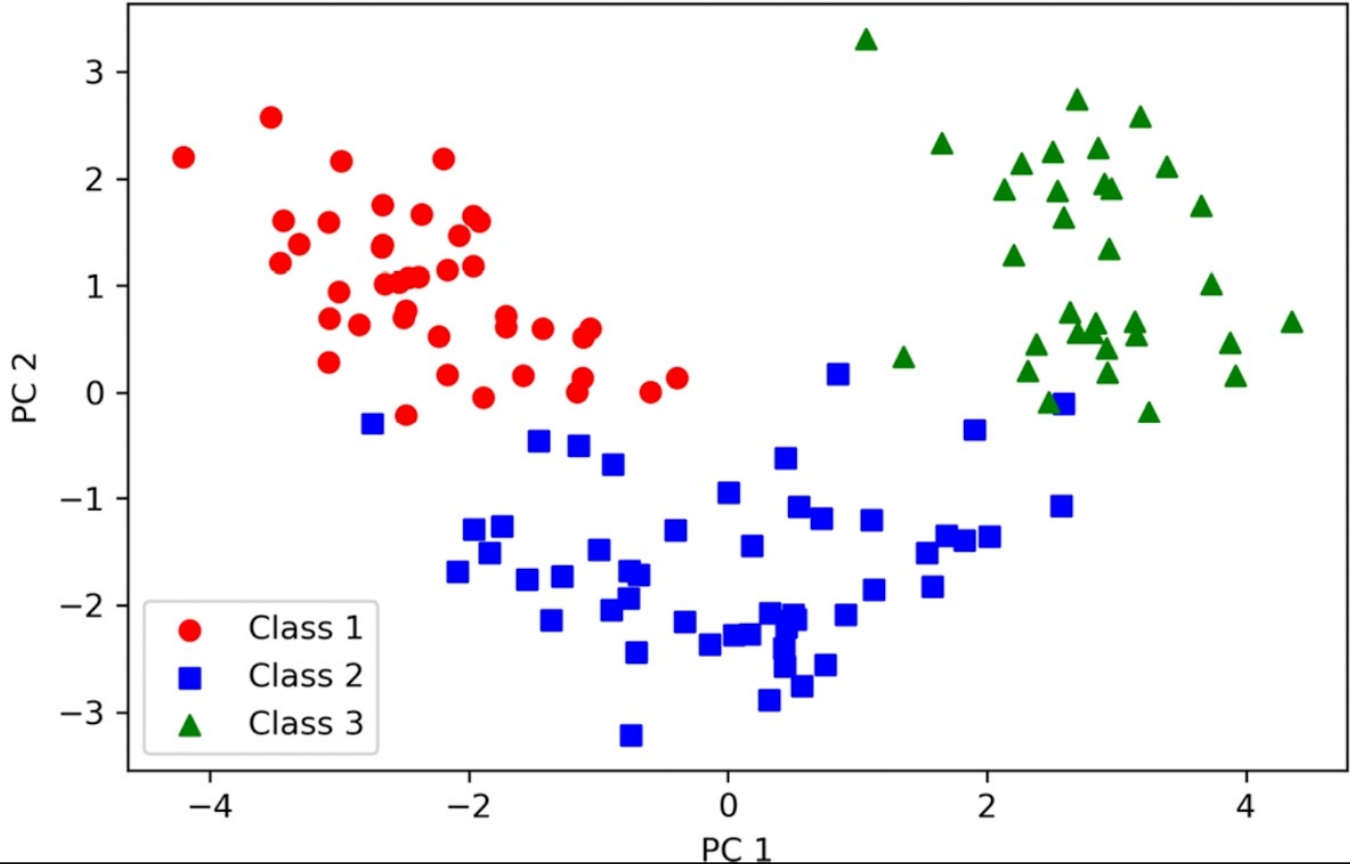
$$X_{PCA} = XW$$

- Your ML algorithm ingests  $X_{PCA}$  instead of  $X$

# SkLearn Wine Raw Features



# SkLearn Wine Dataset PC1 and PC2



```
from sklearn.decomposition import PCA

pca = PCA(n_components=None)
X_train_pca = pca.fit_transform(X_train_std)
pca.explained_variance_ratio_
```

Figure 5.3: Data records from the Wine dataset projected onto a 2D feature space via PCA

Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn"

# Pop Quiz

If we split a dataset  $X$  in  $X_{\text{train}}$ ,  $X_{\text{val}}$ , and  $X_{\text{test}}$  sets, what set do we use to compute the PCA projection matrix  $W$ ?

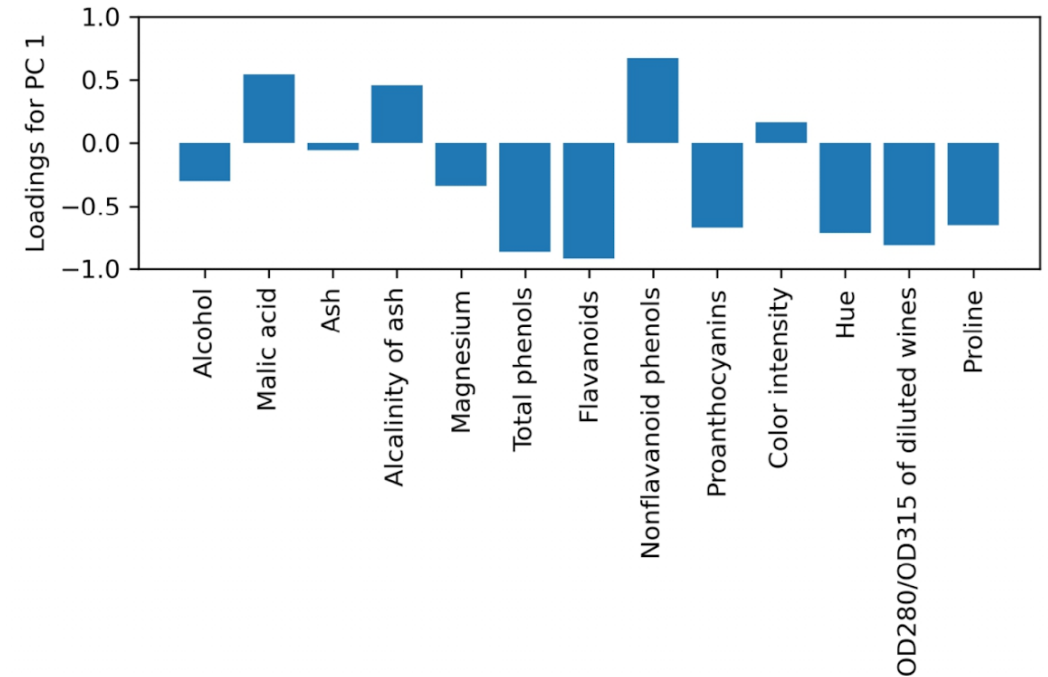
- A. The whole dataset  $X$
- B. The training set  $X_{\text{train}}$
- C. The validation set  $X_{\text{val}}$
- D. The test set  $X_{\text{test}}$
- E. The training and validation sets  $X_{\text{train}}$  and  $X_{\text{val}}$

**Can we learn anything about the raw features  
from the Principal Components?**



# Loadings

- Principal components are a combination of original features
- How can we assess the contributions of the original features?
  - Compute loadings  
 $loadings = eigenvectors * np.sqrt(eigenvals)$
  - The loadings measures the correlation between the original features and the principal components.



Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn"

# Feature Extraction

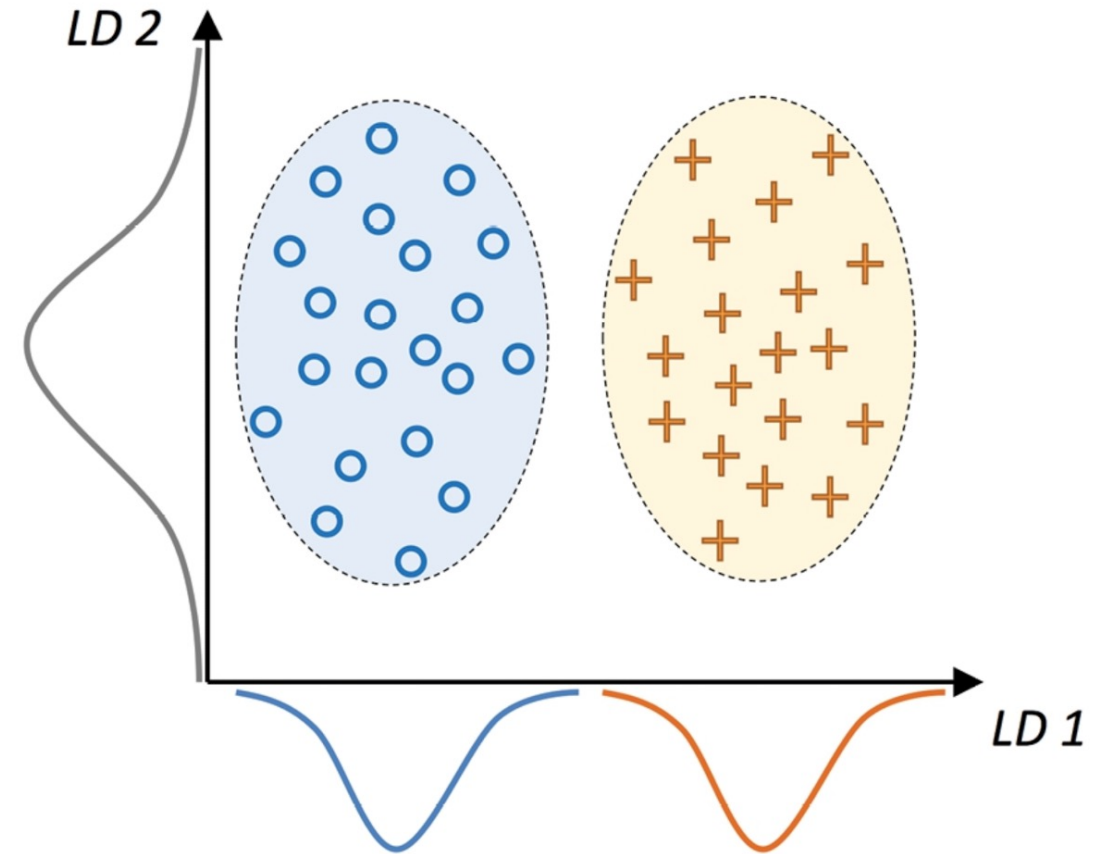
- Principal Component Analysis (PCA)
- **Linear Discriminant Analysis (LDA)**
- t-distributed stochastic neighbor embedding (t-SNE)

# Linear Discriminant Analysis (LDA)

- Also known as Fisher's LDA (Ronald A. Fisher 1936)
- As PCA, the high-level goal is to reduce dimensionality
- Supervised technique (i.e., requires labels)
- Increase computational efficiency
- Reduce overfitting (Curse of dimensionality)
- **PCA Objective:** find the orthogonal component axes of maximum variance in a dataset
- **LDA Objective:** find the feature subspace that optimizes class separability.

# Two-Class LDA Problem

- LD2: Good principal component
- LD1: Good LDA component
- Assumptions:
  - Data is normally distributed.
  - Classes have identical covariance matrices
  - Training examples are statistically independent of each other
- LDA typically works well even when one or more assumptions are violated.



Source: Raschka, et. al., "Machine Learning with PyTorch and Scikit-Learn"

# LDA Algorithm

1. Standardize original feature data  $x$  (Note: Assumes normality) with  $m$  features.
2. For each class, compute the  $m$ -dimensional mean vector  $\bar{v}$
3. Construct the between-class scatter matrix  $S_B$  and within class scatter matrix  $S_W$
4. Compute the eigenvectors and corresponding eigenvalues of the matrix  $S_W^{-1} S_B$
5. Sort eigenvalues in decreasing order. This also sorts eigenvectors.
6. Construct projection matrix  $W$  with the  $k$  largest eigenvalues (columns).  $W$  has shape  $m \times k$ .
7. Transform original feature data  $x$  with  $W$  to generate  $z$

# Step 2: $m$ -dimensional mean vector

- For a dataset with  $C$  classes and  $m$  features, we will generate  $C$   $m$ -dimensional mean vectors, each with  $m$  elements.
- General  $m$ -dimensional vector

- $\bar{v}_i = \begin{bmatrix} \mu_{i,x_1} \\ \mu_{i,x_2} \\ \vdots \\ \mu_{i,x_m} \end{bmatrix}$ , where  $\mu_{i,x_j}$  is the average feature  $j$  value for class  $i$ .

$i \in \{1, 2, \dots, C\}$

calculation of the scatter matrix and  $m_i$  stores the mean

This results in the

These mean vectors are used to calculate one mean vector for

```
>>> np.set_printoptions(precision=2)
>>> mean_vecs = {}
>>> for label, counts in class_counts.items():
...     mean_vecs[label] = (1/counts.sum()) * np.sum(class_means[label] * counts, axis=0)
...     print(label, mean_vecs[label])
```

# *m*-dimensional Mean Vector Example

Sample ID	$x_1$	$x_2$	Label
1	1	3	0
2	2	3	0
3	3	2	0
4	4	3	1
5	4	2	1
6	5	3	1
7	2	5	2
8	3	4	2
9	4	5	2

$$\bar{v}_0 = \begin{bmatrix} \frac{(1 + 2 + 3)}{3} \\ \frac{(3 + 3 + 2)}{3} \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

# *m*-dimensional Mean Vector Example

Sample ID	$x_1$	$x_2$	Label
1	1	3	0
2	2	3	0
3	3	2	0
4	4	3	1
5	4	2	1
6	5	3	1
7	2	5	2
8	3	4	2
9	4	5	2

$$\bar{v}_0 = \begin{bmatrix} \frac{(1 + 2 + 3)}{3} \\ \frac{(3 + 3 + 2)}{3} \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix}$$

$$\bar{v}_1 = \begin{bmatrix} \frac{(4 + 4 + 5)}{3} \\ \frac{(3 + 2 + 3)}{3} \end{bmatrix} = \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix}$$

$$\bar{v}_2 = \begin{bmatrix} \frac{(2 + 3 + 4)}{3} \\ \frac{(5 + 4 + 5)}{3} \end{bmatrix} = \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix}$$



# Step 3: Construct Scatter Matrices

- Compute within scatter matrix  $S_W$  of size  $m \times m$

$$S_W = \sum_{i=1}^C S_i, \text{ where}$$

$$S_i = \sum_{x \in D_i} (x - \bar{v}_i)(x - \bar{v}_i)^T$$

- This is tracking the within class variance.

# Step 3: Construct Scatter Matrices

Sample ID	$x_1$	$x_2$	Label
1	1	3	0
2	2	3	0
3	3	2	0
4	4	3	1
5	4	2	1
6	5	3	1
7	2	5	2
8	3	4	2
9	4	5	2

- Compute within scatter matrix  $S_W$  of size  $m \times m$

$$S_W = \sum_{i=1}^C S_i, \text{ where}$$
$$S_i = \sum_{x \in D_i} (x - \bar{v}_i)(x - \bar{v}_i)^T$$

- This is tracking the within class variance.
- Continuing with our example

$$\bar{v}_0 = \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \quad \bar{v}_1 = \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix} \quad \bar{v}_2 = \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix}$$

$$S_0 = \sum_{x \in D_0} (x - \bar{v}_0)(x - \bar{v}_0)^T$$
$$= \left( \begin{bmatrix} 1 \\ 3 \end{bmatrix} - \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \right) \left( \begin{bmatrix} 1 \\ 3 \end{bmatrix} - \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \right)^T + \left( \begin{bmatrix} 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \right) \left( \begin{bmatrix} 2 \\ 3 \end{bmatrix} - \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \right)^T$$
$$+ \left( \begin{bmatrix} 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \right) \left( \begin{bmatrix} 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \right)^T = \begin{bmatrix} 2 & -1 \\ -1 & 0.6667 \end{bmatrix}$$

# Step 3: Construct Scatter Matrices

Sample ID	$x_1$	$x_2$	Label
1	1	3	0
2	2	3	0
3	3	2	0
4	4	3	1
5	4	2	1
6	5	3	1
7	2	5	2
8	3	4	2
9	4	5	2

- Compute within scatter matrix  $S_W$  of size  $m \times m$

$$S_W = \sum_{i=1}^C S_i, \text{ where}$$

$$S_i = \sum_{x \in D_i} (x - \bar{v}_i)(x - \bar{v}_i)^T$$

- Continuing with our example

$$\bar{v}_0 = \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \quad \bar{v}_1 = \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix} \quad \bar{v}_2 = \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix}$$

$$\begin{aligned}
S_1 &= \sum_{x \in D_1} (x - \bar{v}_1)(x - \bar{v}_1)^T \\
&= \begin{pmatrix} [4] \\ [3] \end{pmatrix} - \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix} \begin{pmatrix} [4] \\ [3] \end{pmatrix} - \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix}^T + \begin{pmatrix} [4] \\ [2] \end{pmatrix} - \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix} \begin{pmatrix} [4] \\ [2] \end{pmatrix} - \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix}^T \\
&+ \begin{pmatrix} [5] \\ [3] \end{pmatrix} - \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix} \begin{pmatrix} [5] \\ [3] \end{pmatrix} - \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix}^T = \begin{bmatrix} 0.6667 & 0.3333 \\ 0.3333 & 0.6667 \end{bmatrix}
\end{aligned}$$

# Step 3: Construct Scatter Matrices

Sample ID	$x_1$	$x_2$	Label
1	1	3	0
2	2	3	0
3	3	2	0
4	4	3	1
5	4	2	1
6	5	3	1
7	2	5	2
8	3	4	2
9	4	5	2

- Compute within scatter matrix  $S_W$  of size  $m \times m$

$$S_W = \sum_{i=1}^C S_i, \text{ where}$$

$$S_i = \sum_{x \in D_i} (x - \bar{v}_i)(x - \bar{v}_i)^T$$

- Continuing with our example

$$\bar{v}_0 = \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \quad \bar{v}_1 = \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix} \quad \bar{v}_2 = \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix}$$

$$S_2 = \sum_{x \in D_2} (x - \bar{v}_2)(x - \bar{v}_2)^T$$

$$= \left( \begin{bmatrix} 2 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix} \right) \left( \begin{bmatrix} 2 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix} \right)^T + \left( \begin{bmatrix} 3 \\ 4 \end{bmatrix} - \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix} \right) \left( \begin{bmatrix} 3 \\ 4 \end{bmatrix} - \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix} \right)^T$$

$$+ \left( \begin{bmatrix} 4 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix} \right) \left( \begin{bmatrix} 4 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix} \right)^T = \begin{bmatrix} 2 & 0 \\ 0 & 0.6667 \end{bmatrix}$$

# Step 3: Construct Scatter Matrices

- Compute within scatter matrix  $S_W$  of size  $m \times m$

$$S_W = \sum_{i=1}^C S_i, \text{ where}$$

$$S_i = \sum_{x \in D_i} (x - \bar{v}_i)(x - \bar{v}_i)^T$$

- Continuing with our example

$$S_0 = \begin{bmatrix} 2 & -1 \\ -1 & 0.6667 \end{bmatrix}, \quad S_1 = \begin{bmatrix} 0.6667 & 0.3333 \\ 0.3333 & 0.6667 \end{bmatrix}, \quad S_2 = \begin{bmatrix} 2 & 0 \\ 0 & 0.6667 \end{bmatrix}$$

$$S_W = S_0 + S_1 + S_2 = \begin{bmatrix} 4.667 & -0.667 \\ -0.667 & 2.0 \end{bmatrix}$$

# Step 3: Construct Scatter Matrices

- Compute within scatter matrix  $S_W$  of size  $m \times m$

$$S_W = \sum_{i=1}^C S_i, \text{ where}$$
$$S_i = \sum_{x \in D_i} (x - \bar{v}_i)(x - \bar{v}_i)^T$$

- Continuing with our example

$$S_0 = \begin{bmatrix} 2 & -1 \\ -1 & 0.6667 \end{bmatrix}, \quad S_1 = \begin{bmatrix} 0.6667 & 0.3333 \\ 0.3333 & 0.6667 \end{bmatrix}, \quad S_2 = \begin{bmatrix} 2 & 0 \\ 0 & 0.6667 \end{bmatrix}$$

$$S_W = S_0 + S_1 + S_2 = \begin{bmatrix} 4.667 & -0.667 \\ -0.667 & 2.0 \end{bmatrix}$$

When classes are unbalanced, normalize each scatter matrix.

$$S_W = \sum_{i=1}^C \frac{S_i}{n_i}$$

# Step 3: Construct Scatter Matrices

- Compute **between** scatter matrix  $S_B$  of size  $m \times m$

$$S_B = \sum_{i=1}^C n_i (\bar{v}_i - \bar{v})(\bar{v}_i - \bar{v})^T,$$

where  $\bar{v}$  is the overall mean including examples from all  $C$  classes.

$$\bar{v}_0 = \begin{bmatrix} 2 \\ 8 \\ 3 \end{bmatrix} \quad \bar{v}_1 = \begin{bmatrix} 13 \\ 3 \\ 8 \\ 3 \end{bmatrix} \quad \bar{v}_2 = \begin{bmatrix} 3 \\ 14 \\ 3 \end{bmatrix}$$

Sample ID	$x_1$	$x_2$	Label
1	1	3	0
2	2	3	0
3	3	2	0
4	4	3	1
5	4	2	1
6	5	3	1
7	2	5	2
8	3	4	2
9	4	5	2

$$\bar{v} = \begin{bmatrix} 3.11 \\ 10 \end{bmatrix}$$

# LDA Algorithm

1. Standardize original feature data  $x$  (Note: Assumes normality) with  $m$  features.
2. For each class, compute the  $m$ -dimensional mean vector  $\bar{v}$
3. Construct the between-class scatter matrix  $S_B$  and within class scatter matrix  $S_W$
4. Compute the eigenvectors and corresponding eigenvalues of the matrix  $S_W^{-1} S_B$
5. Sort eigenvalues in decreasing order. This also sorts eigenvectors.
6. Construct projection matrix  $W$  with the  $k$  largest eigenvalues (columns).  $W$  has shape  $m \times k$ .
7. Transform original feature data  $x$  with  $W$  to generate  $z$

Similar to PCA



**Step 4: Compute the eigenvectors and corresponding eigenvalues of the matrix  $S_W^{-1}S_B$**

```
eigen_vals, eigen_vecs =\
| np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
```

**Step 5: Sort eigenvalues and pick eigenvectors corresponding to  $k$ -largest eigenvalues.**

# Step 6: Construct projection matrix $W$

- Construct  $W$  as we did for PCA
- Expect at most  $C - 1$  linear discriminants
- Discriminability Ratio is equivalent to PCA Explained Variance Ratio

SKLearn Wine dataset.

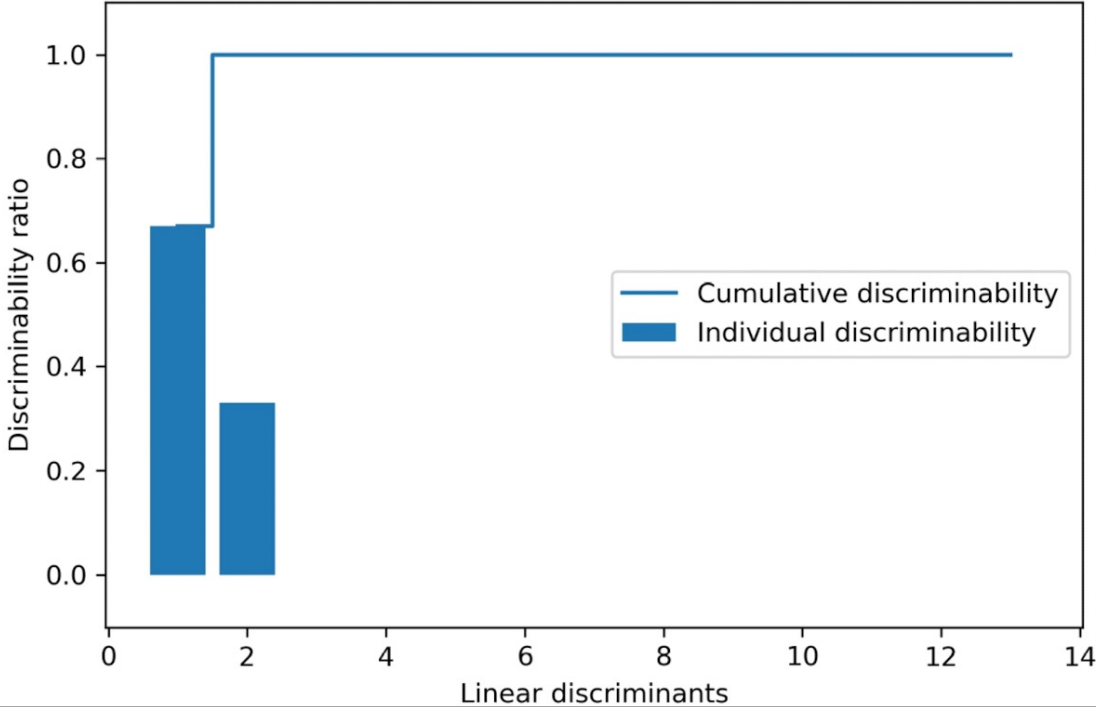


Figure 5.9: The top two discriminants capture 100 percent of the useful information

# Step 7: Transform original feature data $x$ with $W$ to generate $z$

- Same projection step as with PCA,  $Z = XW$

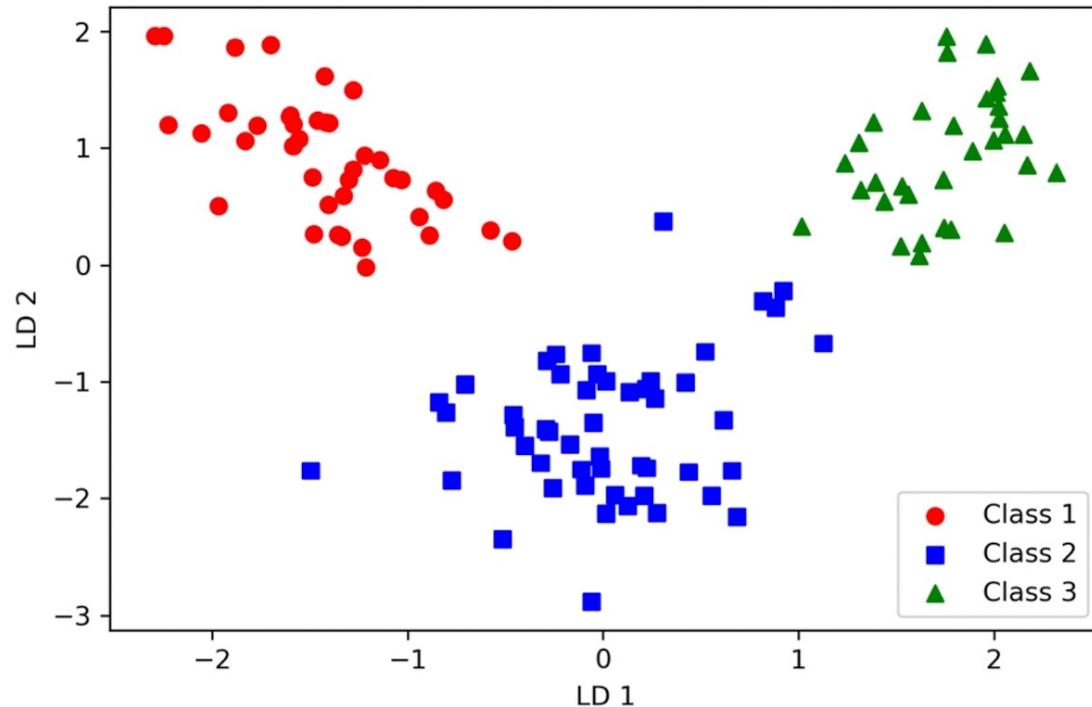


Figure 5.10: Wine classes perfectly separable after projecting the data onto the first two discriminants

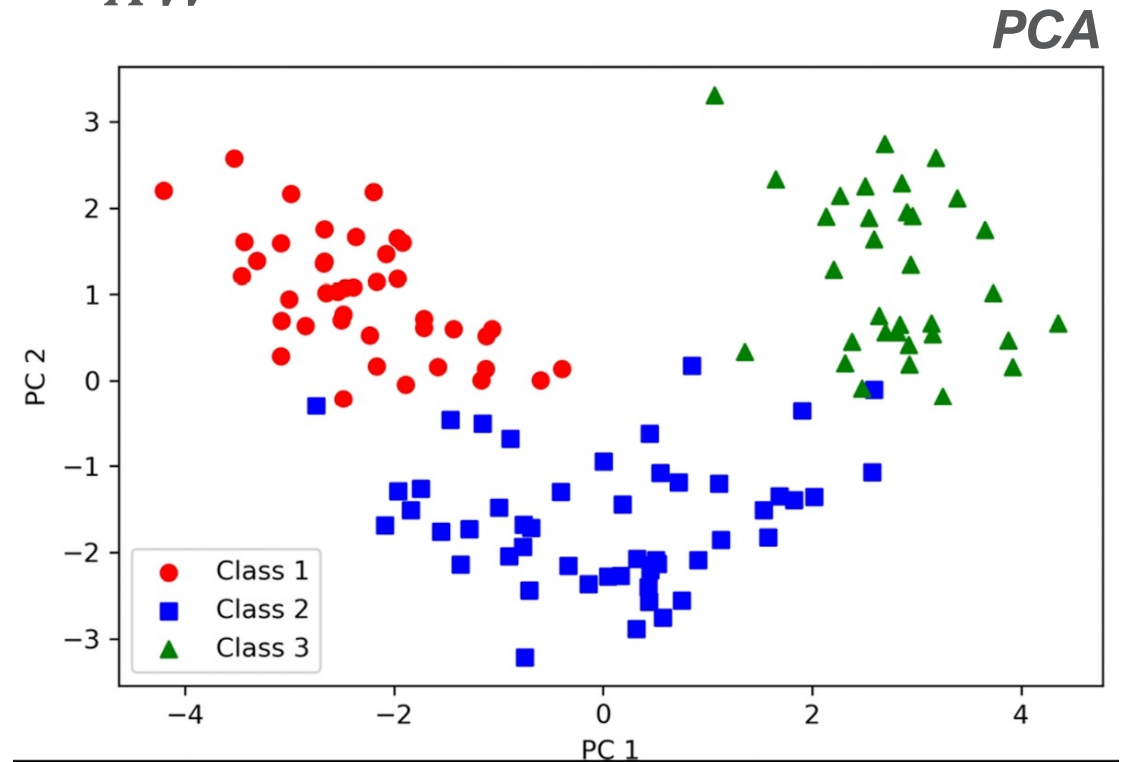
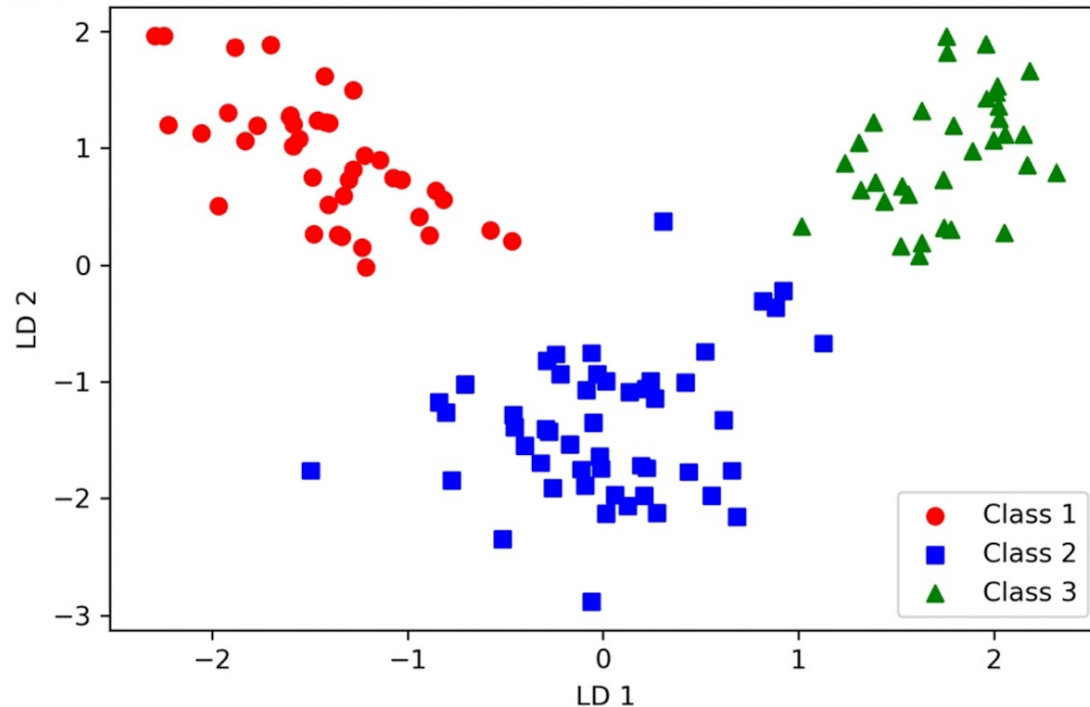


Figure 5.3: Data records from the Wine dataset projected onto a 2D feature space via PCA

# Step 7: Transform original feature data $x$ with $W$ to generate $z$

- Same projection step as with PCA,  $Z = XW$



```
# LDA Library
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# Constructor
lda = LDA(n_components=2)

# Fit and Transform
X_train_lda = lda.fit_transform(X_train_std, y_train)
```

Figure 5.10: Wine classes perfectly separable after projecting the data onto the first two discriminants

# Pop Quiz

X finds eigenvectors that best separate the classes, while Y finds the eigenvectors in the direction of the highest variance in the data independent of the class.

- A. X is PCA and Y is LDA
- B. X is LDA and Y is PCA
- C. None of the above

# Feature Extraction

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- **t-distributed stochastic neighbor embedding (t-SNE)**

# t-distributed stochastic neighbor embedding (t-SNE)

- Popular for data visualization
- Projects m-dimensional space to 2D/3D
- It is a non-linear dimensionality reduction technique also known as manifold learning
  - Manifold: lower-dimensional topological space embedded in a high-dimensional space
- Other sklearn non-linear techniques for dimensionality reduction can be found at <http://scikit-learn.org/stable/modules/manifold.html>

# T-SNE

*MNIST Digits Dataset*

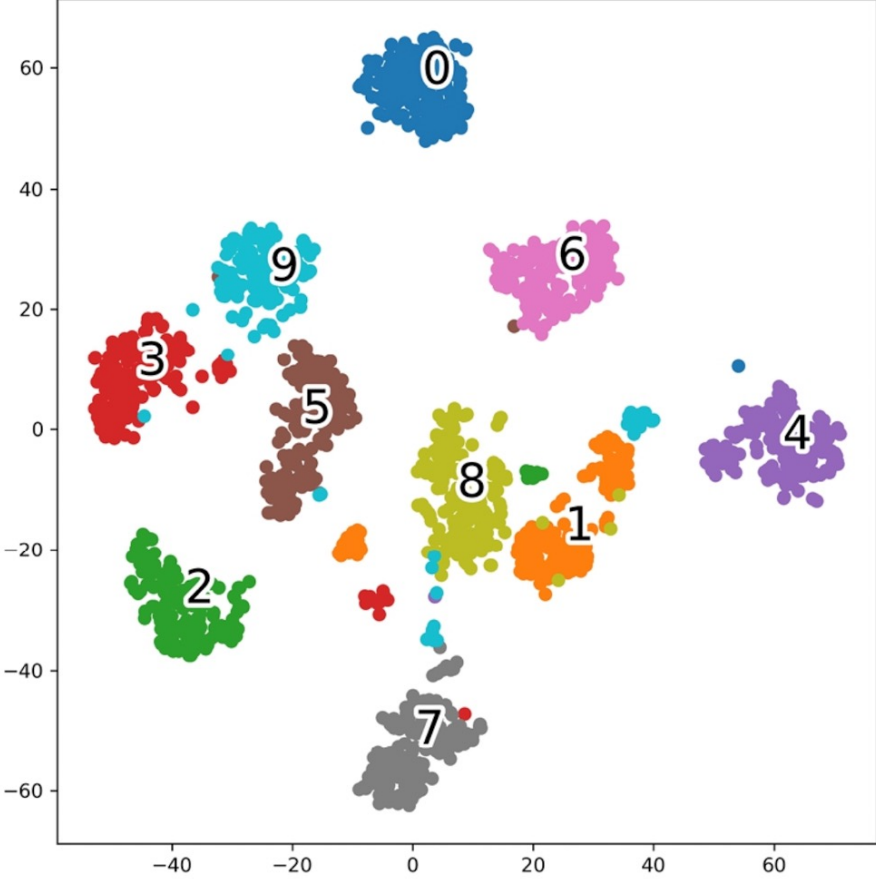
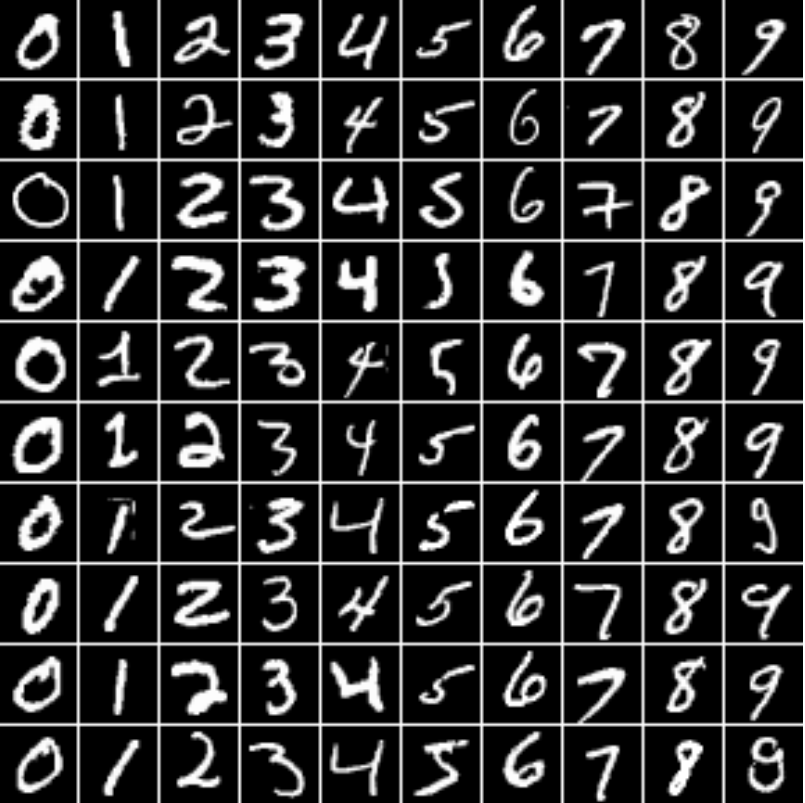


Figure 5.16: A visualization of how t-SNE embeds the handwritten digits in a 2D feature space



# Dimensionality Reduction



# Feature Extraction Review

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- t-distributed stochastic neighbor embedding (t-SNE)

# Review

- Feature Selection
  - Lasso Regularization
  - Sequential Backward Selection
  - Random Forest Feature Selection
  - Permutation Feature Importance
- Feature Extraction
  - Principal Component Analysis (PCA)
  - Linear Discriminant Analysis (LDA)
  - t-distributed stochastic neighbor embedding (t-SNE)



# Next Lecture

- Shapley Values for XAI
- Unsupervised learning



# Helper Slides

