Linear Discriminant Analysis

AI+X
INSTITUTE FOR ARTIFICIAL INTELLIGENCE

Linear Discriminant Analysis (LDA)

- Can be seen as a supervised version of PCA
  - Requires class labels for the training samples
- LDA finds a linear combination of features that characterizes or separates two or more classes of objects or events

Linear Discriminant Analysis (LDA)

- PCA focuses on information conservation
- LDA focuses on class separation

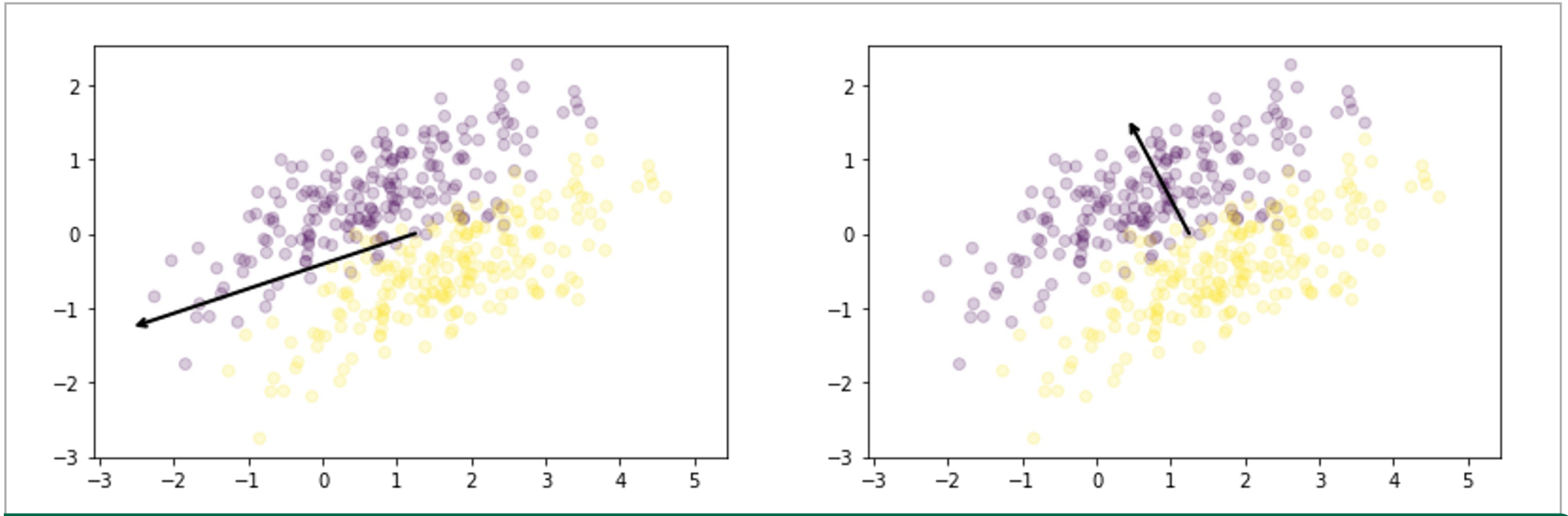

Figure 1. First component computed by PCA (left) and LDA (right).

LDA Algorithm

1. Get some data $\mathbf{A} = \{\mathbf{a_1}, \ldots, \mathbf{a_N}\}$ and class labels $\mathbf{y} = \{\mathbf{y_1}, \ldots, \mathbf{y_N}\}$
2. Compute the mean $\bar{\mathbf{a}}_\mathbf{c}$ for each class label $\mathbf{c}$
3. Compute the covariance matrix per class $\mathbf{S_c} = \sum_{\mathbf{i} \mid \mathbf{y[i]=c}} (\mathbf{a_i} - \bar{\mathbf{a}}_\mathbf{c})(\mathbf{a_i} - \bar{\mathbf{a}}_\mathbf{c})^\mathbf{T}$
4. Compute the covariance matrix within classes $\mathbf{S_W} = \sum_\mathbf{c} \mathbf{S_c}$
5. Compute the global mean $\bar{\mathbf{a}}$
6. Calculate the covariance matrix between classes $\mathbf{S}_B = \sum_\mathbf{c} \mathbf{N_c} (\bar{\mathbf{a}}_\mathbf{c} - \bar{\mathbf{a}})(\bar{\mathbf{a}}_\mathbf{c} - \bar{\mathbf{a}})^\mathbf{T}$
7. Calculate the eigenvectors $\mathbf{v_j}$ and eigenvalues $\boldsymbol{\lambda_j}$ for $\mathbf{S_W^{-1} S_B}$
   - Sort the eigenvectors in decreasing order of eigenvalues, so that $\boldsymbol{\lambda_j} > \boldsymbol{\lambda_{j+1}} \forall \mathbf{j}$
8. Transform $\mathbf{a_i}$ into $\mathbf{w_i}$: $\mathbf{w_i} = \mathbf{V}(\mathbf{a_i} - \bar{\mathbf{a}})$
9. Reconstruct $\mathbf{a_i}$ from $\mathbf{w_i}$: $\hat{\mathbf{a}}_\mathbf{i} = \mathbf{w_i V} + \bar{\mathbf{a}}$

scikit-learn: LDA Implementation

https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

```
sklearn.discriminant_analysis.LinearDiscriminantAnalysis(
        solver='svd',
        shrinkage=None,
        priors=None,
        n_components=None,
        store_covariance=False,
        tol=0.0001,
        covariance_estimator=None
)
```

Parameters::

solver : *{'svd', 'lsqr', 'eigen'}, default='svd'*

Solver to use, possible values:

- 'svd': Singular value decomposition (default). Does not compute the covariance matrix, therefore this solver is recommended for data with a large number of features.
- 'lsqr': Least squares solution. Can be combined with shrinkage or custom covariance estimator.
- 'eigen': Eigenvalue decomposition. Can be combined with shrinkage or custom covariance estimator.

shrinkage : *'auto' or float, default=None*

Shrinkage parameter, possible values:

- None: no shrinkage (default).
- 'auto': automatic shrinkage using the Ledoit-Wolf lemma.
- float between 0 and 1: fixed shrinkage parameter.

This should be left to None if `covariance_estimator` is used. Note that shrinkage works only with 'lsqr' and 'eigen' solvers.

priors : *array-like of shape (n_classes,), default=None*

The class prior probabilities. By default, the class proportions are inferred from the training data.

n_components : *int, default=None*

Number of components (<= min(n_classes - 1, n_features)) for dimensionality reduction. If None, will be set to min(n_classes - 1, n_features). This parameter only affects the `transform` method.

Attributes::

coef_ : *ndarray of shape (n_features,) or (n_classes, n_features)*

Weight vector(s).

intercept_ : *ndarray of shape (n_classes,)*

Intercept term.

covariance_ : *array-like of shape (n_features, n_features)*

Weighted within-class covariance matrix. It corresponds to `sum_k prior_k * C_k` where `C_k` is the covariance matrix of the samples in class `k`. The `C_k` are estimated using the (potentially shrunk) biased estimator of covariance. If solver is 'svd', only exists when `store_covariance` is True.

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 explained variances is equal to 1.0. Only available when eigen or svd solver is used.

Figure 2. scikit-learn: LDA implementation.

scikit-learn: LDA Implementation

- Methods

`decision_function`(X)	Apply decision function to an array of samples.
`fit`(X, y)	Fit the Linear Discriminant Analysis model.
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_feature_names_out`([input_features])	Get output feature names for transformation.
`get_params`([deep])	Get parameters for this estimator.
`predict`(X)	Predict class labels for samples in X.
`predict_log_proba`(X)	Estimate log probability.
`predict_proba`(X)	Estimate probability.
`score`(X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X)	Project data to maximize class separation.

Figure 3. scikit-learn: LDA implementation.

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.

PCA/LDA for Visualization

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

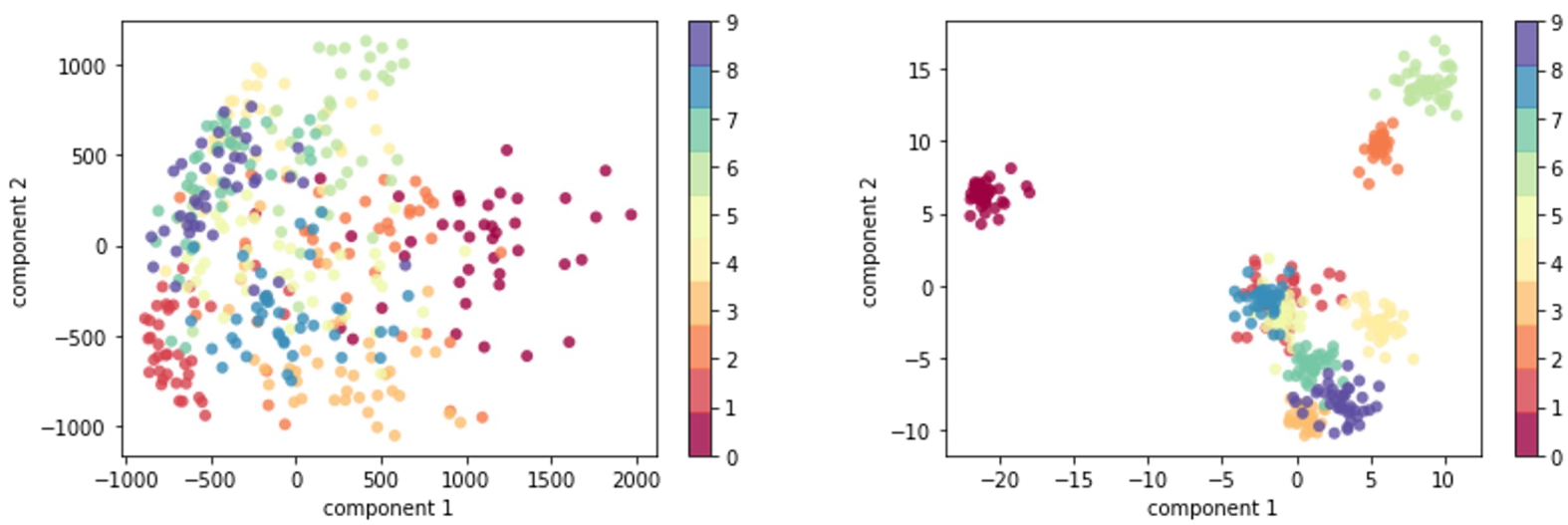

Figure 5. 2-dimensional projection of the MNIST images using PCA (left) and LDA (right).

LDA for Multi-class Classification

- LDA assumes all classes have the same covariance matrix
- Similarly to PCA, LDA can estimate the probability of a probe sample belonging to each class using those distributions
- The implementation of LDA in scikit-learn has a "predict" method that returns the class with the highest probability for a given probe sample

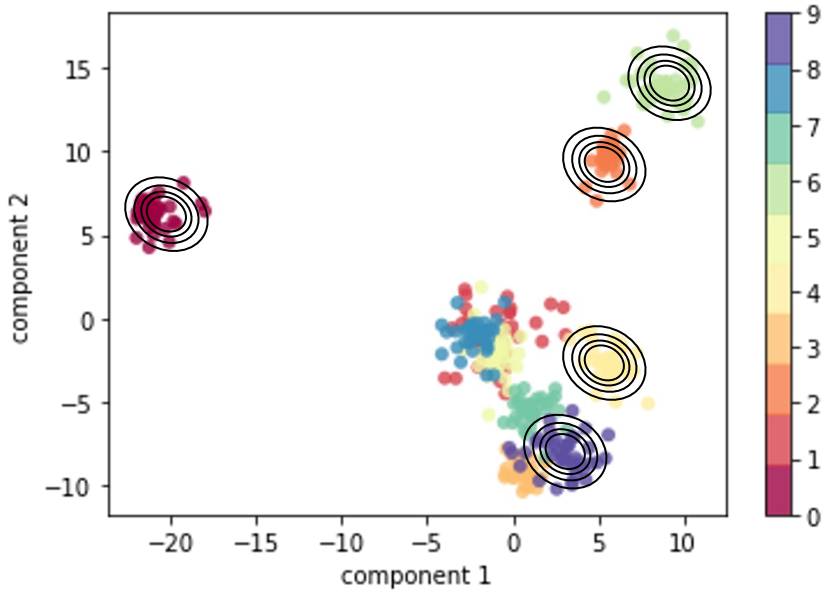

Figure 6. Illustration of the training distribution learned using LDA.

Knowledge Check 1

LDA is more computationally expensive than PCA. For this reason, some applications use PCA first to reduce the dimensionality of the training data and then apply LDA over the obtained PCA projections to reorganize them focusing on class separation. This solution can speed up LDA, but at the cost of:

A. Losing the ability to reconstruct the original sample

B. Losing discriminative information discarded by PCA

C. Losing the ability to use more than two classes

D. Losing noise information discarded by PCA

How to Evaluate the Results?

- Classification
  - Accuracy: number of correctly classified samples / total number of samples
    - Okay if classes are balanced
  - F-score: harmonic mean of precision and recall
    - F-score = (2×precision×recall)/(precision+recall)
    - Compute F-score for each class and report the average F-score

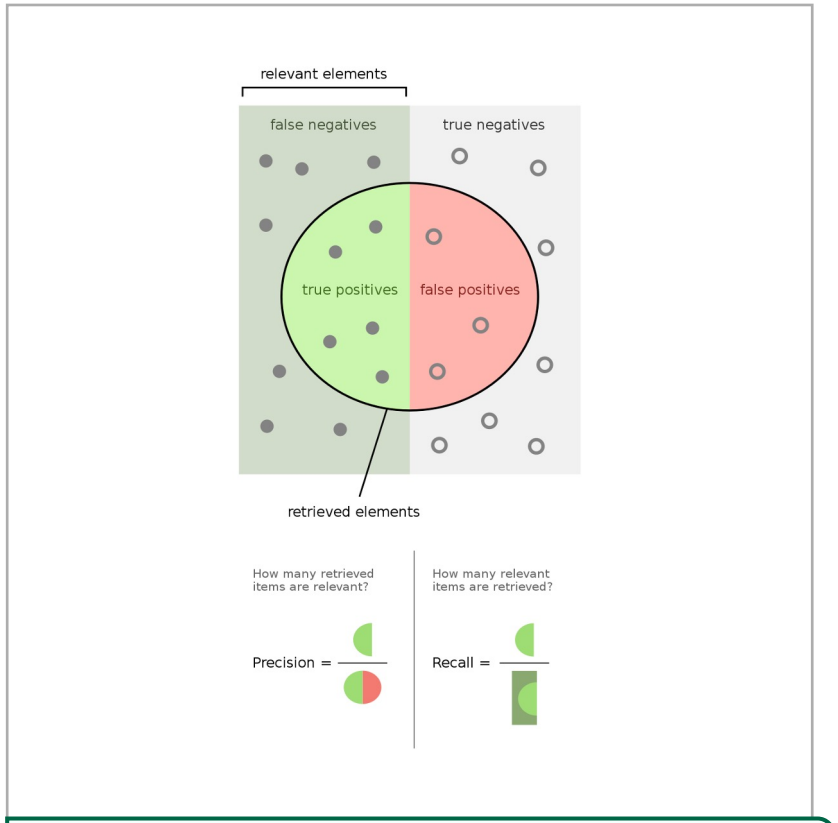

Figure 7. Precision and recall.

You have reached the end of the lecture.

Image/Figure References

Figure 1. First component computed by PCA (left) and LDA (right). Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 2. scikit-learn: LDA implementation. Source: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html
Figure 3. scikit-learn: LDA implementation. Source: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html
Figure 4. First 40 images from each MNIST class. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 5. 2-dimensional projection of the MNIST images using PCA (left) and LDA (right). Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 6. Illustration of the training distribution learned using LDA. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.
Figure 7. Precision and recall. Source: Russell & Norvig, Artificial Intelligence: A Modern Approach, 4th edition, Pearson, 2021.