

3.1 - Feature Engineering and Dimensionality Reduction

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Introduction to Machine Learning for Finance

Bologna - February-March, 2026

Lesson 2 — Learning Objectives

The goal of this lecture is to understand **why high-dimensional data are problematic in financial applications** and how to deal with this issue in a principled way.

By the end of this lecture, you should be able to:

- explain what the curse of dimensionality means in practice for financial data,
- identify the main sources of overfitting in high-dimensional settings,
- recognize why in-sample performance is unreliable in finance.

What You Will Be Able to Do After This Lecture

After completing this lecture, you should be able to:

- apply regularization, feature selection, and dimensionality reduction consciously,
- design machine learning pipelines that avoid information leakage,
- choose validation schemes appropriate for time-series financial data,
- evaluate models with a focus on robustness and stability rather than peak accuracy.

These skills are **prerequisites** for understanding and using supervised learning models correctly, which will be the focus of the next lecture.

The curse of dimensionality

The High-Dimensional Feature Problem

Managing complexity and robustness in financial ML

In many financial ML tasks, the feature space is large:

- Hundreds or thousands of predictors (technical indicators, fundamentals, macro, alternative data)
- Strong *redundancy* (many features encode similar information)
- Strong *collinearity* (features are highly correlated)

Key issue: model flexibility increases with dimensionality, but *generalization* typically worsens when

p is large relative to n ,

especially under noise and non-stationarity.

Why Too Many Features Hurt Out-of-Sample Performance

Managing complexity and robustness in financial ML

High dimensionality often leads to:

- **Overfitting:** the model fits noise rather than signal
- **Unstable estimates:** small data changes produce large parameter changes
- **Spurious relationships:** many predictors \Rightarrow many accidental correlations
- **Poor OOS performance:** training metrics look good, test metrics degrade

In practice, the marginal benefit of adding features quickly saturates, while variance can explode.

Why Finance Is a Worst-Case Scenario

Managing complexity and robustness in financial ML

Finance amplifies the high-dimensional problem because:

- Predictive signals are typically **weak** (low signal-to-noise ratio)
- Relationships are **non-stationary** (regime changes, structural breaks)
- Many features are **transformations of the same underlying information**
- Interpretability and robustness matter (risk management, regulation, model governance)

In finance, controlling model complexity is not optional: it is essential.

Three Strategies to Control Model Complexity

Managing complexity and robustness in financial ML

To address high-dimensional feature spaces, we can adopt three conceptually different strategies:

- ➊ **Regularization:** penalize model complexity without removing features
- ➋ **Feature Selection:** remove irrelevant or redundant features while keeping meaning
- ➌ **Dimensionality Reduction:** transform features into a lower-dimensional representation

These strategies solve the *same* problem (generalization under high dimensionality) in *different* ways. They are complementary, not mutually exclusive.

What Changes – and What Doesn't

Managing complexity and robustness in financial ML

Approach	# Features	Meaning Preserved?	Feature Space
Regularization	Same	Yes	Original
Feature Selection	Reduced	Yes	Original
Dimensionality Reduction	Reduced	Often no	Transformed

Choosing among these options is a **modeling decision** driven by objectives such as: *predictive accuracy, robustness, interpretability, and computational cost*.

Curse of Dimensionality: What Actually Breaks?

In quantitative finance, the typical failure mode is not model bias, but estimation noise and validation bias.

Let p be the number of features and n the number of training observations.

- When p grows at fixed n , the hypothesis class grows and **variance** increases.
- Many features are correlated, unstable, and weakly informative: the model can fit noise and still look good *in-sample*.
- **Non-stationarity** amplifies the problem: a feature that works in one regime may fail in the next.

Key message: In finance, p/n is often too large, so **your main job is to control variance and prevent leakage.**

A Minimal Quantitative Picture

Consider a linear model $y = X\beta + \varepsilon$ with i.i.d. noise $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

Training error typically decreases with p . But **test error** behaves like:

$$\mathbb{E}[\text{MSE}_{\text{test}}] \approx \text{Bias}^2(p) + \text{Var}(p) + \sigma^2$$

and **$\text{Var}(p)$ tends to increase** as p grows relative to n .

Operational implication:

- If you do not regularize or reduce dimension, out-of-sample performance degrades.
- In finance, you can get high in-sample R^2 with near-zero (or negative) out-of-sample R^2 .

A Practical Diagnostic: The p/n Ratio

In many financial applications:

$\frac{p}{n}$ is not small.

Rules of thumb (not laws):

- If $p/n \ll 1$: classical estimation may work (still watch collinearity).
- If $p/n \approx 0.1 - 1$: regularization and strict validation become essential.
- If $p/n > 1$: you **must** regularize / select / reduce dimension, and you must validate carefully.

Finance-specific twist: effective sample size can be *much smaller* than n due to autocorrelation, overlapping labels, and regime shifts.

Regularization

- Many regression techniques can over-fit, particularly when there are a large number of correlated features.
- Results for validation set may not then be as good as for training set
- Regularization is a way of avoiding overfitting controlling effective model complexity and variance. Alternatives:
 - **Ridge Regression**
 - **Lasso Regression**
 - **Elastic net**
- We must first scale feature values

Ridge Regression

- Ridge regression is a regularization technique where we change the function that is to be minimize;
- Reduce magnitude of regression coefficients by choosing a parameter λ and minimizing

$$\frac{1}{2N} \sum_{n=1}^N \left[h_{\theta} \left(x^{(n)} \right) - y^{(n)} \right]^2 + \lambda \sum_{j=1}^d b_j^2 \quad (1)$$

- This change has the effect of encouraging the model to keep the weights b_j as small as possibile;
- The Ridge regression should only be used for determining model parameters using the training set;
- What happens as λ increases?

Lasso Regression

- Lasso is short for *Least Absolute Shrinkage and Selection Operator*;
- Similar to ridge regression except we minimize

$$\frac{1}{2N} \sum_{n=1}^N \left[h_{\theta} \left(x^{(n)} \right) - y^{(n)} \right]^2 + \lambda \sum_{j=1}^d |b_j| \quad (2)$$

- This function cannot be minimized analytically and so a variation on the gradient descent algorithm must be used;
- Lasso regression also has the effect of simplifying the model. It does this by setting the weights of unimportant features to zero. When there are a large number of features, Lasso can identify a relatively small subset of the features that form a good predictive model.

Lasso Regression: Geometrical Explanation

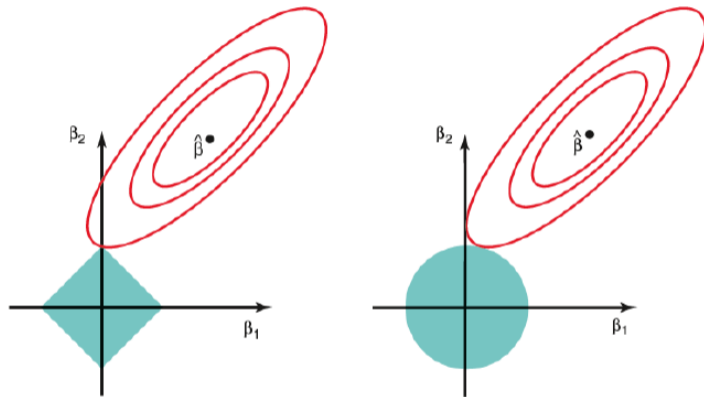


FIGURE 6.7. Contours of the error and constraint functions for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS.

Lasso Regression: Geometrical Explanation

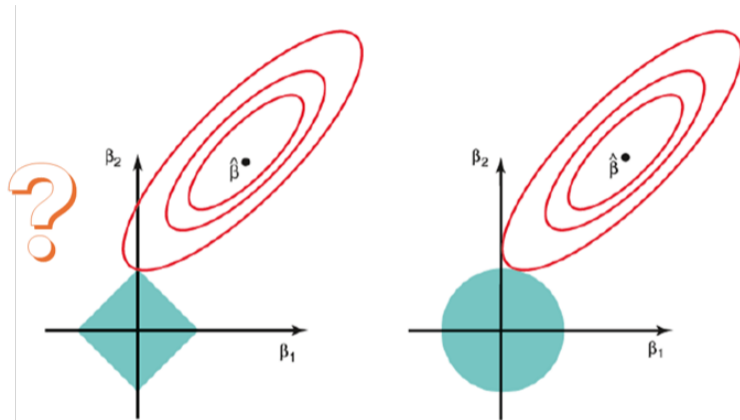


FIGURE 6.7. Contours of the error and constraint functions for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions, $|\beta_1| + |\beta_2| \leq s$ and $\beta_1^2 + \beta_2^2 \leq s$, while the red ellipses are the contours of the RSS.

L1 Norm Constraint as a Cross Polytope

Level Sets of the Objective Function

- **L1 Norm Constraint as a Cross Polytope:**

- In 2D, the L1 norm constraint $\|\beta\|_1 \leq t$ (for some $t > 0$) forms a diamond shape (a square rotated by 45 degrees) with four corners at $(\pm t, 0)$ and $(0, \pm t)$.
- In higher dimensions, this shape generalizes to a cross polytope with $2n$ corners in n -dimensional space.

- **Level Sets of the Objective Function:**

- In many Lasso explanations, we visualize level sets (contours) of the unconstrained objective—often ellipses in the case of squared-error loss—expanding outward from a central point (e.g., the ordinary least squares solution).
- These ellipses will eventually intersect the L1 diamond or polytope.

Corners Promote Sparsity

Why It May Seem Less Obvious in 2D

Corners Promote Sparsity:

- The “sparsity” effect arises because corners of the L1 region correspond to solutions where one or more coefficients are zero (i.e., the solution “hits” an axis).
- In higher dimensions, there are many more corners (since there are $2n$ vertices in an n -dimensional cross polytope), which increases the likelihood that the outward-expanding contours will meet the L1 ball at a corner, thereby setting some coefficients to zero.

Corners Promote Sparsity

Why It May Seem Less Obvious in 2D

Why It May Seem Less Obvious in 2D:

- In two dimensions, the diamond only has four vertices. Depending on the orientation of the contours (ellipses), it may not always appear as "corner-favoring"
- Even in 2D, as you vary the penalty parameter t , you can often see the solution snap to a corner (meaning one coordinate goes to zero).
- In higher dimensions, this effect is magnified by the large number of corners, making it more likely that some coefficients become exactly zero.

Elastic Net Regression

- Middle ground between Ridge and Lasso
- Minimize

$$\frac{1}{2N} \sum_{n=1}^N \left[h_{\theta} \left(x^{(n)} \right) - y^{(n)} \right]^2 + \lambda_1 \sum_{j=1}^d b_j^2 + \lambda_2 \sum_{j=1}^d |b_j| \quad (3)$$

- In Lasso some weights are reduced to zero but others may be quite large. In Ridge, weights are small in magnitude but they are not reduced to zero. The idea underlying Elastic Net is that we may be able to get the best of both by making some weights zero while reducing the magnitude of the others.

Feature Selection

Introduction to Feature Selection

- Feature selection is the process of identifying and retaining only the most relevant variables for a machine learning model.
- The goal is to improve model performance by reducing noise and redundancy in the dataset.
- High-dimensional datasets often contain irrelevant or correlated features that do not contribute meaningful information for prediction.
- Unlike feature extraction, which transforms existing features into new ones (e.g., PCA, LDA), feature selection retains a subset of the original features.
- Feature selection enhances interpretability, reduces training time, and prevents overfitting, leading to a more efficient and effective model.

Why Feature Selection is Important?

- **Reduces Overfitting:** By removing irrelevant or redundant features, the model focuses only on meaningful variables, reducing the risk of learning noise instead of true patterns.
- **Improves Accuracy:** By eliminating uninformative features, the model generalizes better to unseen data, leading to improved prediction performance.
- **Speeds Up Training:** Fewer features mean lower computational complexity, reducing the time required to train and optimize machine learning models.

How do we select features?

- A feature selection procedure combines a search technique with an evaluation method. The search technique proposes new feature subsets, and the evaluation measure determines the how good the subset is.
- In a perfect world, a feature selection method would evaluate all possible subsets of feature combinations and determine which one results in the best performing machine learning model.
- However, computational cost inhibits such a practice in reality. In addition, the optimal subset of features varies between machine learning models. A feature subset that optimizes one model's performance won't necessarily optimize another's.

Methods of Feature Selection

- 1 **Filter Methods:** These methods apply statistical techniques to evaluate the importance of each feature independently of the model. Common techniques include correlation analysis, Chi-square test, and mutual information.
- 2 **Wrapper Methods:** These methods iteratively evaluate subsets of features using machine learning models, selecting the best-performing combination. Examples include Recursive Feature Elimination (RFE), Forward Selection, and Backward Elimination.
- 3 **Embedded Methods:** These methods integrate feature selection within the model training process. Techniques such as Lasso (L1) regression and Decision Tree-based feature importance are commonly used.

- A typical filter algorithm consists of two steps: it ranks features based on certain criteria and then chooses the highest-ranking features to train the machine learning models.
- Filter methods are generally univariate, so they rank each feature independently of the rest. Because of this, the filter methods tend to ignore any interactions that occur between features. Thus, redundant variables will not necessarily be eliminated by filter methods.
- However, some multivariate filter selection methods exist as well. These consider features in relations to others in the data set, making them naturally capable of handling redundant features. Their selection criteria scans for duplicated features and correlated features and provide simple but powerful methods to quickly remove redundant information.

Constant, quasi-constant, and duplicated features

- The most basic and intuitive methods for feature selection consist of removing constant, quasi-constant, or duplicated features.
- Constant features only show one value for all the observations in the data set. That is, they show absolutely no variability.
- Quasi-constant features are similar; if most observations share the same value, then we'd label that feature quasi-constant. In practice, quasi-constant features typically refer to those variables where more than 95 to 99 percent of the observations show the same value.
- Duplicated features, as the name indicates, are those that are in essence, identical. That is, for every observation, they show the same value.

Constant, quasi-constant, and duplicated features

- Although it sounds obvious and overly simple, many datasets contain a lot of constant, quasi-constant, and duplicated features.
- In fact, duplicated features often arise when generating new features by one-hot encoding of categorical variables.
- Removing these features is an easy but effective way to reduce the dimension of the feature space, without losing any significant information.

Correlation

- Correlation measures the linear association between two or more variables.
- The higher the correlation, the more linearly associated the variables are.
- The central hypothesis is that good feature sets contain features that are highly correlated with the target, yet uncorrelated with each other.
- If two variables are correlated, we can predict one from the other.
- Therefore, if two features are correlated, the model only really needs one of them, as the second one does not add additional information.

Mutual Information

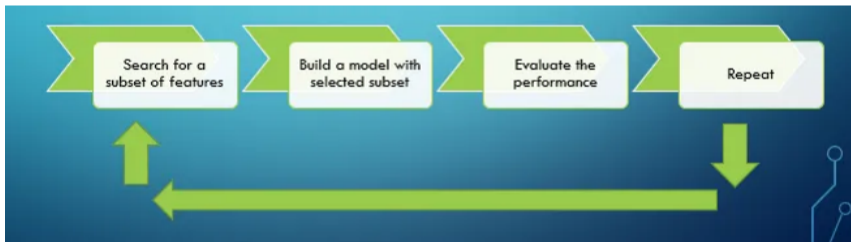
- Mutual information measures the mutual dependence between two variables, in this case, the feature and the target. Mutual information is similar to correlation, but more general; it doesn't strictly represent linear association. It measures how much knowing one of these variables reduces uncertainty in the other.

Fisher Score

- Fisher score uses the Chi-Square distribution to measure the dependency between two variables and works with categorical or discrete variables.
- Fisher score essentially compares the actual frequencies of the values of a variable with the expected frequencies if it had no relation to the target.

Wrapper Methods

- In comparison to filter methods, wrapper methods tend to be more computationally expensive, but select a better set of features.
- Wrapper methods use a specific machine learning algorithm in the selection process and choose the best subset of features tailored for that algorithm.
- This subset of features may not be optimal for a different machine learning model. In other words, **wrapper methods are not model agnostic**.



Example: RFE with Logistic Regression

Wrapper Methods

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
rfe = RFE(model, n_features_to_select=5)
rfe.fit(X_train, y_train)
selected_features = X_train.columns[rfe.support_]
print("Selected_Features:", selected_features)
```

- Feature selection occurs during model training.
- Uses model-specific techniques.
- Examples:
 - Lasso (L1) Regression
 - Decision Trees and Random Forests

Comparison of Methods

Method	Pros	Cons
Filter	Fast, scalable	Ignores interactions
Wrapper	Better selection	Expensive
Embedded	Model-optimized	Model-dependent

Feature Selection in Finance: The Stability Problem

In finance, many predictors are:

- weak (low signal-to-noise)
- correlated (redundant information)
- non-stationary (regime-dependent)

Result: the set of “selected features” may change dramatically with small changes in the sample.

So, selection should be evaluated not only by accuracy, but also by **stability over time**.

A Simple Stability Check (Rolling Selection Frequency)

Perform feature selection on multiple rolling windows (or bootstrap blocks) and track how often each feature is selected.

Let S_k be the selected set on window k . Define selection frequency:

$$f_j = \frac{1}{K} \sum_{k=1}^K \mathbb{I}\{j \in S_k\}$$

Interpretation:

- high $f_j \Rightarrow$ robust signal (more trustworthy)
- low $f_j \Rightarrow$ unstable / regime-specific / likely noise

Understanding Pipelines in Scikit-Learn

- A pipeline is a sequence of data processing steps that are applied in a structured way.
- It helps automate machine learning workflows by chaining together multiple operations.
- Instead of manually applying transformations (e.g., scaling, feature selection, model training), a pipeline does it all in one step.
- Think of a pipeline as an assembly line:
 - Raw data enters the pipeline.
 - The data is preprocessed (e.g., missing values handled, scaling applied).
 - Feature selection picks the most relevant features.
 - A machine learning model is trained on the transformed data.
 - Predictions are generated efficiently.

Feature Selection Pipeline in Scikit-Learn

- Machine learning pipelines in Scikit-Learn allow seamless integration of preprocessing, feature selection, and model training.
- They enhance workflow efficiency, reproducibility, and ensure that transformations are applied consistently during training and inference.
- A typical pipeline consists of:
 - Preprocessing (e.g., standardization, encoding categorical variables)
 - Feature Selection (e.g., Recursive Feature Elimination, SelectKBest)
 - Model Training (e.g., Logistic Regression, Random Forest)
- Pipelines prevent data leakage by ensuring that feature selection and scaling are only learned from the training data.

Example: Feature Selection Pipeline in Scikit-Learn

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.linear_model import LogisticRegression

# Define the pipeline
pipeline = Pipeline([
# Standardize features
('scaler', StandardScaler()),
# Select top 5 features
('feature_selection', SelectKBest(score_func=f_classif, k=5)),
# Train logistic regression model
('classifier', LogisticRegression())
])
# Fit the pipeline on training data
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test) # Make predictions
```

Advantages of Using Pipelines

- **Automation:** Reduces manual steps and minimizes human errors.
- **Prevents Data Leakage:** Ensures feature selection and scaling are applied correctly.
- **Code Reusability:** Simplifies experimentation with different models and preprocessing techniques.
- **Hyperparameter Tuning:** Easily integrates with GridSearchCV and RandomizedSearchCV for optimizing parameters.

Conclusion and Best Practices

- Choose method based on dataset size and complexity.
- Filter methods for quick insights.
- Wrapper methods for optimal selection.
- Embedded methods for model-specific optimization.
- Always validate selected features.

Dimensionality Reduction

Feature Extraction and Feature Selection

- Even though feature extraction and feature selection processes share some overlap, often these terms are erroneously equated.
- **Feature extraction** is the process of using domain knowledge to extract new variables from raw data that make machine learning algorithms work.
- The **feature selection** process is based on selecting the most consistent, relevant, and non-redundant features.
- The feature selection process is based on selecting the most consistent, relevant, and non-redundant feature subset from feature vectors.
- It not only reduces training time and model complexity, but it eventually helps to **prevent overfitting**.

Principal Component Analysis (PCA)

- PCA is a widely used technique that transforms the original features into a new set of orthogonal components.
- The first few principal components retain the most variance in the data.
- Helps in reducing dimensionality while preserving important patterns.
- PCA works by:
 - Standardizing the dataset.
 - Computing the covariance matrix.
 - Finding eigenvectors and eigenvalues.
 - Selecting the top principal components.
 - Transforming the data into the new feature space.
- PCA is effective for noise reduction and visualization.

- Given a dataset with n features, PCA computes the covariance matrix:

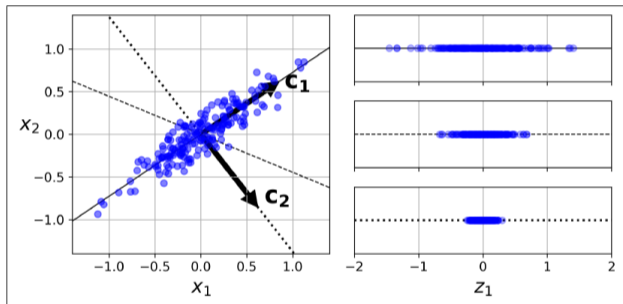
$$C = \frac{1}{n} X^T X \quad (4)$$

- The eigenvectors and eigenvalues of C represent the principal components and their importance.
- The data is projected onto the top k eigenvectors with the highest eigenvalues:

$$Z = XW \quad (5)$$

where W is the matrix of selected eigenvectors.

Standardization and Gradient Descent



Example: PCA in Scikit-Learn

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

Advantages and Limitations of PCA

- **Advantages:**

- Reduces redundancy by removing correlated features.
- Improves model efficiency by reducing computation time.
- Helps in visualizing high-dimensional data.

- **Limitations:**

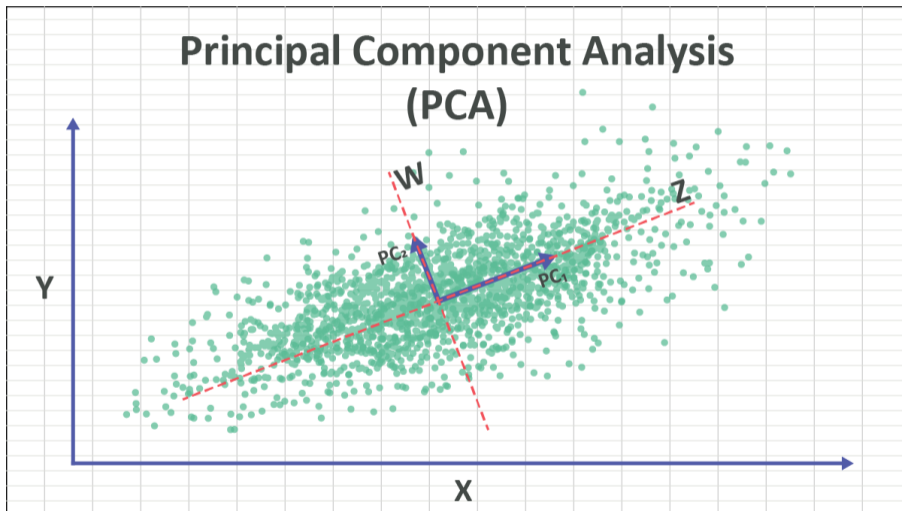
- Assumes linear relationships in data.
- Loses interpretability of original features.
- Sensitive to data scaling; standardization is required.

When to Use PCA?

- When dealing with high-dimensional data to reduce computational costs.
- When visualization of data is required in lower dimensions.
- When removing noise and redundancy is essential for better model performance.
- When feature interpretability is less important compared to efficiency.

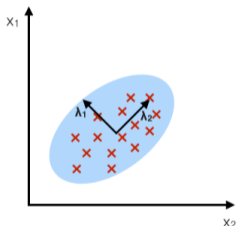
Linear Discriminant Analysis (LDA)

- LDA is a supervised dimensionality reduction technique that maximizes class separability.
- Unlike PCA, which focuses on variance, LDA optimizes for class separation.
- Works well for classification problems by finding a lower-dimensional space that retains discriminative information.
- LDA assumes that different classes generate data based on Gaussian distributions with the same covariance matrix.



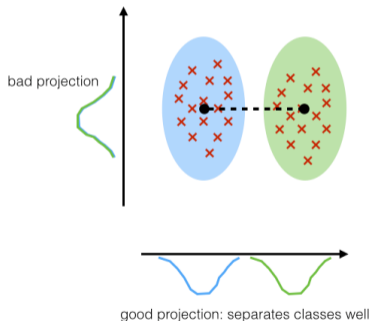
PCA:

component axes that maximize the variance



LDA:

maximizing the component axes for class-separation



- LDA computes two key matrices:
 - Within-class scatter matrix (S_W): Measures variance within each class.
 - Between-class scatter matrix (S_B): Measures variance between different classes.
- LDA solves the generalized eigenvalue problem:

$$S_W^{-1} S_B v = \lambda v \quad (6)$$

where eigenvectors corresponding to the largest eigenvalues define the optimal projection space.

Example: LDA in Scikit-Learn

```
from sklearn.discriminant_analysis
    import LinearDiscriminantAnalysis

from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)
```

Advantages and Limitations of LDA

- **Advantages:**

- Improves class separability in classification problems.
- Reduces dimensionality while preserving discriminative information.
- Computationally efficient compared to other non-linear methods.

- **Limitations:**

- Assumes normally distributed classes with equal covariance matrices.
- Less effective when class distributions overlap significantly.
- Requires labeled data for training.

When to Use LDA?

- When working with classification tasks that require feature reduction.
- When the goal is to maximize class separation in lower-dimensional space.
- When the dataset follows Gaussian distribution assumptions.

Comparison: PCA vs LDA

Method	Objective	Type
PCA	Maximize variance	Unsupervised
LDA	Maximize class separability	Supervised

- LDA is a powerful tool for supervised dimensionality reduction, particularly in classification tasks.
- It optimizes feature space for class separability, making it useful for many real-world applications.
- Choosing between PCA and LDA depends on the problem: use PCA for general structure preservation and LDA for class separability.

Conclusions

Lesson 2 — What Was the Real Problem?

Throughout this lecture, we focused on a single underlying issue:

In financial applications, data are high-dimensional, noisy, and unstable.

This makes naive machine learning:

- highly prone to overfitting,
- extremely sensitive to validation choices,
- often misleading out-of-sample.

The curse of dimensionality is not a technical inconvenience — it is a structural feature of financial data.

Key Takeaway #1 — Managing Complexity

As the number of features grows:

- variance increases faster than signal,
- in-sample performance becomes unreliable,
- out-of-sample performance typically degrades.

Important:

- More features do not mean more information.
- In finance, many predictors can be redundant or unstable.

Managing complexity is the core task of ML in finance.

Key Takeaway #2 — Regularization, Selection, Reduction

We discussed three complementary strategies:

- **Regularization:** constrain model parameters to control variance.
- **Feature selection:** reduce dimensionality by discarding predictors.
- **Dimensionality reduction:** compress information into fewer components.

None of these is universally superior.

They are tools to control the same problem: how flexible your model is allowed to be.

Key Takeaway #2 — Regularization, Selection, Reduction

Goal: maximize out-of-sample robustness under non-stationarity.

- **Need interpretability:** Ridge / Elastic Net; conservative feature selection; stability checks.
- **Strong collinearity:** Ridge or Elastic Net; optionally PCA as a validated preprocessing step.
- **Compute limited:** filter methods + embedded methods (regularized linear models, tree-based).
- **Very high p/n :** dimension reduction + strong regularization; simplify the target and features.

Key Takeaway #3 — Validation Is Part of the Model

In financial data:

- observations are not i.i.d.,
- labels may overlap in time,
- regimes change.

As a consequence:

- random cross-validation is often invalid,
- leakage can easily invalidate results,
- evaluation design strongly affects conclusions.

A model without a correct validation scheme is meaningless.

Key Takeaway #4 — Stability Can Matter More Than Accuracy

In finance, small predictive edges are:

- fragile,
- regime-dependent,
- easily destroyed by noise and costs.

Therefore:

- feature selection should be stable over time,
- coefficients should not fluctuate wildly,
- performance should be consistent across windows.

Robustness beats peak performance.

Machine Learning does not fail in finance because models are weak.

It fails because complexity, noise, and validation are underestimated.

What Changes in Lesson 3

In the next lecture, we will start introducing **supervised learning models**.

This raises a natural question:

If models are powerful, why do we still worry so much about dimensionality and validation?

Lesson 3 — What Supervised Models Actually Do

Supervised learning models:

- do not remove noise,
- do not guarantee predictability,
- do not fix non-stationarity.

What they do is:

- define a class of functions,
- impose an inductive bias,
- determine how complexity grows with data.

Choosing a model means choosing a way to trade bias for variance.

How Lesson 2 Prepares Lesson 3

Everything we introduced in Lesson 2 will remain active:

- regularization will control model flexibility,
- feature selection will interact with model structure,
- dimensionality reduction will shape the input space,
- validation will determine what we believe.

Lesson 3 is not a fresh start. It builds directly on the constraints we have just learned.

Preview of Lesson 3

In Lesson 3 we will:

- introduce supervised learning models step by step,
- start from linear and regularized models,
- analyze their geometry and assumptions,
- study their behavior under high dimensionality,
- evaluate them using time-aware validation.

Focus: understanding model behavior, not memorizing algorithms.

There is no powerful model without controlled complexity.

There is no good performance without honest validation.

Lesson 3 will show how supervised models fit into this framework.