

Supervised Learning Fundamentals

Prepared by Dr. Sumaya Alkubaisi

FNT633
Machine Learning in Finance

Lecture 5



Learning Goals

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

- Identify main types of ML and their financial applications.
- Distinguish between regression and classification.
- Evaluate model performance using MSE, RMSE, and R^2 .
- Apply K-Fold Cross Validation.
- Implement KNN and Decision Tree models for finance datasets.

Types of Machine Learning

- **Supervised Learning:** Model learns from labeled data (input–output pairs).
Examples: credit scoring, stock price prediction.
- **Unsupervised Learning:** Model finds hidden patterns or clusters in unlabeled data.
Examples: customer segmentation, anomaly detection.
- **Reinforcement Learning:** Model learns by interacting with an environment and receiving feedback (rewards or penalties).
Examples: portfolio optimization, trading bots.

Supervised Learning

In supervised learning, we have:

- **Features (X):** Inputs (e.g., income, credit score, investment size)
- **Target (y):** Output or label (e.g., default risk = Yes/No, volatility = High/Low)
- **The model's goal:** map inputs to outputs by minimizing prediction error.

■ Supervised Finance Applications

- Credit Scoring: Predict loan approval likelihood.
- Fraud Detection: Identify suspicious transactions.
- Loan Approval: Classify clients as approved or rejected based on features.

Model Assumptions Reminder

- ❖ In this lecture, we assume the data roughly satisfies the key assumptions for each model — for example, linearity in regression or balanced class distribution in classification.

Detailed statistical checks (normality, significance testing, and visualization) were already covered in the Statistical Foundations for Data Science lecture.

Example:

- Before modeling portfolio returns, perform a normality test.
- If returns are not normally distributed, you may need to transform data or use a non-parametric method for more reliable predictions.

Chapter
5

What is Regression?

- **Regression** is a supervised learning method used to predict continuous numerical values — numbers that can take many possible values.
- It helps us understand how changes in one or more factors (independent variables) affect an outcome (dependent variable).

In finance, regression can be used to:

- Estimate expected returns of a stock or portfolio.
- Predict portfolio volatility (risk).
- Analyze risk-adjusted performance, such as the Sharpe ratio.
- Forecast financial metrics like revenue or interest rates.

Regression helps answer questions like: “How much will a company’s profit increase if sales go up?”
“How will a stock’s price change if market conditions improve?”

Chapter
5

Linear Regression

- Linear Regression assumes a straight-line relationship between variables. It's one of the simplest and most widely used regression techniques.

a. Simple Linear Regression

Equation: $y = \beta_0 + \beta_1 x + \epsilon$

y: dependent variable (what we want to predict)

x: independent variable (predictor)

β_0 : intercept (value of y when x = 0)

β_1 : slope (how much y changes when x increases by 1)

ϵ : random error (difference between actual and predicted values)

Example: -Predicting a student's exam score (y) based on hours studied (x).

As study hours increase, exam scores usually rise — showing a positive linear relationship.

-Predicting a stock's return (y) based on overall market performance (x).

If the market performs well, the stock's return is likely to increase in a similar trend.

b. Multiple Linear Regression

Equation: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \varepsilon$

Used when more than one factor influences the outcome.

It helps us understand how several variables together affect a result.

- Example:
Predicting a house price (y) based on its size (x_1), number of bedrooms (x_2), and location score (x_3).
Each feature adds different information to improve prediction accuracy.
- Predicting a company's stock return (y) using market performance (x_1), company growth rate (x_2), and sector performance (x_3).
Here, each variable contributes to explaining changes in the stock's return.

Chapter
5

Measuring Model Performance

To know how well our regression model works, we measure its error and fit.

Mean Squared Error (MSE)

$$\text{MSE} = \sum (y_{\text{true}} - y_{\text{pred}})^2 / n$$

- Measures average squared difference between actual and predicted values.
- Lower MSE = better performance.

Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{\text{MSE}}$$

- Easier to interpret because it's in the same units as the target variable.
- Lower RMSE means the model's predictions are closer to the real values.

Example: -When predicting daily portfolio returns, a low MSE means the model's predicted returns are close to the real market results.

-When forecasting stock prices, a smaller RMSE means the model's predictions are only a few dollars away from actual prices.

R^2 (Coefficient of Determination)

Measures how much of the variability in y is explained by the model.

R^2 ranges from 0 to 1:

1 $\rightarrow$ perfect prediction

0 $\rightarrow$ no relationship

Example:

-If $R^2 = 0.9$ when predicting car prices, it means 90% of the variation in prices can be explained by features like brand, mileage, and age.

-If $R^2 = 0.85$ when predicting stock returns, it means 85% of the variation in returns can be explained by the selected financial indicators (e.g., company performance or market movement).

Cross Validation

Cross Validation is a technique to test how well a model performs on unseen data — in other words, how well it generalizes beyond the dataset it was trained on. Instead of using a single train-test split, cross validation gives a more reliable estimate of performance.

K-Fold Cross Validation

- The dataset is split into K equal parts (called folds).
- The model is trained on K-1 folds and tested on the remaining fold.
- This process repeats K times, each time using a different fold for testing.
- The average of all K test results gives the final model performance.

Cross Validation

Why it's important:

- Reduces the risk of overfitting (model memorizing data).
- Provides a more accurate and fair evaluation.
- Helps in model comparison (deciding which model performs best).

Example:

Testing a model that predicts loan default risk. Cross validation ensures the model performs reliably across different customer groups, not just the ones seen during training.

Python Demo: Linear Regression

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
model = LinearRegression()  
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
```

- `train_test_split(X, y, test_size=0.2, random_state=0)`: Splits your dataset into training and testing sets.

`X_train` and `y_train`: Data used to train the model.

`X_test` and `y_test`: Data used to evaluate the model's performance.

`test_size=0.2`: 20% of data is for testing, 80% for training.

`model = LinearRegression()`: Creates a linear regression model object, which will try to learn a linear relationship between features (X) and target (y).

`model.fit(X_train, y_train)`: Trains the model using the training data so it can learn the relationship between features and target.

`y_pred = model.predict(X_test)`: What it does: Uses the trained model to predict target values for the testing data (X_test).

Classification

Classification predicts categories or labels, not continuous numbers. It tells us which class an observation belongs to based on its features.

- Example:
Predicting if an email is Spam / Not Spam, or if a tumor is Benign / Malignant.
- Credit Risk: High / Medium / Low
- Fraud Detection: Fraud / Not Fraud
- Investment Category: Aggressive / Balanced / Conservative

In finance, classification helps automate decision-making by assigning outcomes to clear, predefined classes.

Logistic Regression

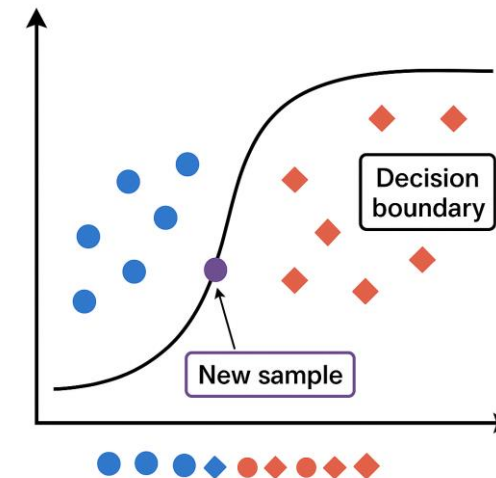
Logistic Regression is a statistical model used for binary or multi-class classification. It predicts the probability that an observation belongs to a certain class.

Instead of producing a direct number (like linear regression), it uses a sigmoid function to map predictions to values between 0 and 1.

$$P(y=1) = 1 / 1 + e^{-(\beta_0 + \beta_1 x)}$$

Output close to 1 → likely belongs to the positive class.

Output close to 0 → likely belongs to the negative class.



Logistic Regression

Example

- Logistic regression might output 0.85 for a loan application — meaning there's an 85% chance this person will default (belong to the “High Risk” class).
- If the output is 0.10, that means only a 10% chance they're “High Risk.”
- Then we decide a cutoff (usually 0.5):
 - $\geq 0.5 \rightarrow$ classify as 1 (positive class), e.g., “Yes, default.”
 - $< 0.5 \rightarrow$ classify as 0 (negative class), e.g., “No, safe.”

So, logistic regression basically converts a prediction into a probability — and then we label it as “yes” or “no” depending on that probability.

K Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple but powerful machine learning algorithm that makes predictions based on similarity.

It doesn't try to learn an internal mathematical model — instead, it looks at nearby data points to decide an outcome.

For classification → it assigns a label based on the majority class among its K nearest neighbors.

For regression → it predicts a numerical value by taking the average of the K nearest points.

Example:

Classifying whether a loan applicant is “High Risk” or “Low Risk” based on the most similar borrowers in the dataset.

Distance Measurement

KNN depends on measuring how “close” two points are — that’s the distance metric. The smaller the distance, the more similar the data points.

Euclidean Distance (most common): $d = \sqrt{\sum (x_1 - x_2)^2}$

Works well for continuous numeric data (e.g., income, age, balance).

Manhattan Distance: $d = \sum |x_1 - x_2|$

Useful when dealing with high-dimensional or grid-like data.

Minkowski Distance:

A generalized form that includes both Euclidean and Manhattan.

Example: Comparing two customers based on attributes like income, loan amount, and credit score — smaller distance = more similar financial behavior.

- Choose the number of neighbors (K) — often an odd number to avoid ties.
- Compute distances between the new data point and all others in the dataset.
- Select the K closest points (the nearest neighbors).
- Make a prediction:
 - For classification → use majority voting.
 - For regression → take the average of neighbor values.

Example:

Predicting house prices — the price of a new house can be estimated by averaging the prices of the 5 nearest houses in the area.

Predicting whether a customer will default based on the payment history of the most similar customers.

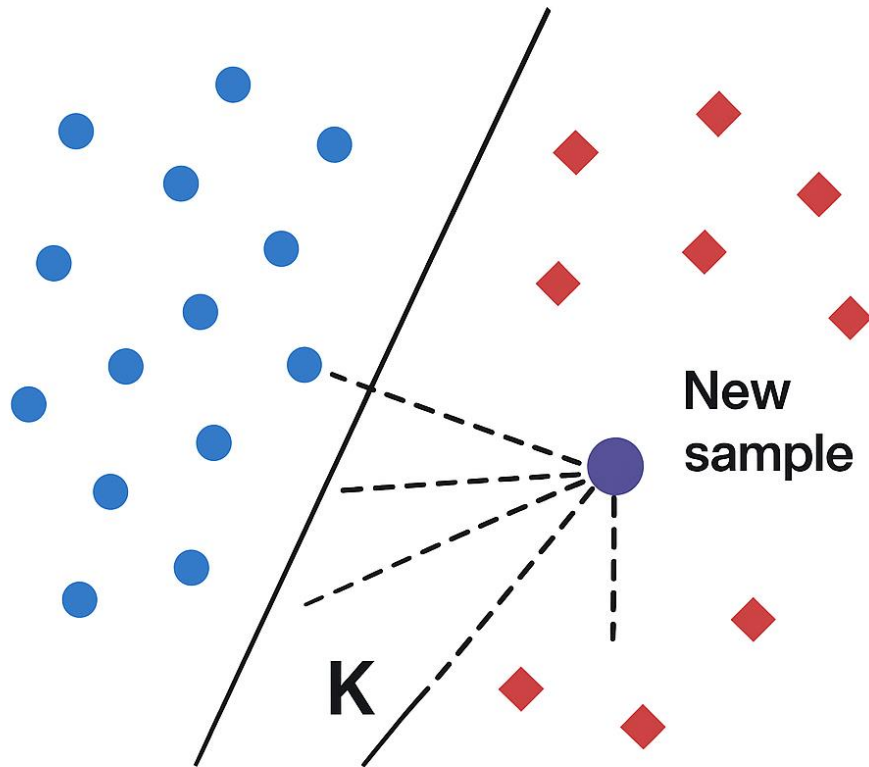
How to choose K

There's no fixed rule, but here are best practices:

- Start small (e.g., $K = 3, 5, 7$) and test performance.
- Use Cross-Validation to find which K gives the best accuracy.
- Generally:
 - Too small K (like 1 or 2): very sensitive to noise → overfitting (high variance, low bias).
 - Too large K: too smooth → underfitting (low variance, high bias) — too general / model ignores important patterns.
- Often, choose odd K values for classification to avoid ties.

Example: When predicting loan default risk, you might test $K = 3, 5, 7$ and find that $K = 5$ gives the best accuracy on your validation data.

That means looking at the 5 most similar past borrowers gives a balanced and reliable prediction.



K-Nearest Neighbors

- KNN classifies the new sample by checking which group (blue or red) its nearest neighbors belong to.
- The class with the majority of close neighbors becomes the prediction.

Since KNN depends on distance, all features must be on the same scale — otherwise, larger numerical ranges dominate the distance calculation.

For example, “income” (in thousands) might overshadow “age” (in years) if not scaled properly.

Common scaling methods:

MinMaxScaler

StandardScaler

Advantages:

- Simple, intuitive, and easy to implement.
- No explicit training phase — predictions are made directly from stored data.
- Adapts well to non-linear relationships.

Limitations:

- Slow for large datasets — must calculate distances for every point.
- Sensitive to irrelevant features — noisy data can reduce accuracy.
- Performance depends on K — too small $\rightarrow$ overfits; too large $\rightarrow$ oversmooths results.
- Requires feature scaling for fair distance comparisons.

KNN can be very effective in finance when relationships between features are non-linear and pattern-based.

Customer Segmentation:

Group customers with similar spending or saving behaviors for personalized marketing.

Credit Risk Categorization:

Classify borrowers into low, medium, or high risk groups based on historical repayment patterns.

Portfolio Behavior Classification:

Group investment portfolios by performance characteristics or volatility profiles to aid decision-making.

Fraud Detection:

Identify transactions that differ significantly from their nearest neighbors — potential anomalies.

Python Demo: Linear Regression

```
knn = KNeighborsClassifier(n_neighbors=5)  
knn.fit(X_train_scaled, y_train_cls)
```

```
# Predict  
y_pred_cls = knn.predict(X_test_scaled)
```

```
# Accuracy  
acc = accuracy_score(y_test_cls, y_pred_cls)  
print(f"KNN Accuracy (k=5): {acc:.3f}")
```

knn = KNeighborsClassifier(n_neighbors=5): Creates a K-Nearest Neighbors model that looks at 5 nearest neighbors to make predictions.

knn.fit(X_train_scaled, y_train_cls): Trains the KNN model using the scaled training data and corresponding class labels.

y_pred_cls = knn.predict(X_test_scaled): Uses the trained model to predict classes for the test data.

acc = accuracy_score(y_test_cls, y_pred_cls): Calculates how accurate the predictions are compared to the true test labels.

Decision Trees

- A Decision Tree is a tree-shaped model that makes predictions by splitting data step-by-step based on feature values.
Each internal node represents a decision rule (e.g., “Is income > \$50,000?”), and each leaf node represents a final prediction or class (e.g., “Low Risk”).
- It works like a flowchart — starting from a question at the top and following branches until reaching a final decision.

Example:

Classifying a customer as High Risk, Medium Risk, or Low Risk based on their loan history, income, and spending behavior.

Splitting Criteria

Decision Trees divide the dataset at each node to make the groups as pure as possible (i.e., each branch contains mostly one class).

Common splitting criteria include:

- **Entropy / Information Gain:** Measures how much information a split adds.
Higher information gain = better split.
- **Gini Index:** Measures impurity (how mixed the classes are).
Lower Gini = purer node.

Example:

If we split customers by “income > 60K,” and one group mostly contains non-defaulters, that’s a good split with high information gain.

Advantages

- Easy to understand and visualize — can be displayed like a decision flowchart.
- Interpretable — managers can easily follow the logic (“if income > X and credit score < Y, then high risk”).
- Handles both numerical and categorical features.
- No need for feature scaling or normalization, unlike KNN or regression.

Limitations

- Can overfit the data:
Trees might become too detailed and memorize noise rather than learning real patterns.
(We’ll cover pruning and Random Forests as solutions in Lecture 6.)
- Unstable with small data changes:
A slight change in data can lead to a very different tree structure.
- Bias toward features with more levels:
If one variable has many unique values, it may dominate splits unfairly.

Decision Trees are widely used in financial decision-making and risk modeling because of their clarity and traceability.

- **Loan Default Prediction:**
Predict whether a customer will default based on salary, credit score, and debt ratio.
- **Fraud Detection:**
Identify suspicious transactions using features like transaction amount, frequency, and merchant category.
- **Client Creditworthiness Classification:**
Classify clients into High, Medium, or Low creditworthiness levels for lending decisions.
- **Portfolio Risk Categorization:**
Categorize investment portfolios by volatility or performance indicators.

Real-world finance use case: Portfolio Risk Prediction.

Dataset: The dataset used here (Portfolio Risk Prediction) was created and explored in Lecture 4.

Goal: build predictive models that help banks or investors assess how risky or volatile a portfolio is — either as a continuous score (volatility) or as a categorical label (risk level). **Tasks:** **Regression Task** → Predict the numerical value of portfolio_volatility (how much the portfolio's value fluctuates).

Model used: Linear Regression

Classification Task → Classify the portfolio as Low, Medium, or High risk based on volatility and other financial features.

Model used: K-Nearest Neighbors (KNN)

Lab Activity: Supervised Learning with Financial Data

```
# -----
```

```
# Application: Portfolio Risk Prediction
```

```
# -----
```

```
# Importing essential libraries
```

```
import pandas as pd
```

```
import numpy as np
```

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split, KFold, cross_val_score
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.neighbors import KNeighborsClassifier
```

Lab Activity: Supervised Learning with Financial Data

```
# The dataset was generated in Lecture 4
# Assume it is saved as 'portfolio_risk.csv'
data = pd.read_csv('portfolio_risk.csv')
print("Dataset Loaded Successfully!")
data.head()

# Check basic info
data.info()

# Summary statistics
data.describe().round(3)

# Check for duplicates and missing values
print("Duplicated rows:", data.duplicated().sum())
print("\nMissing values:\n", data.isnull().sum())
```

Lab Activity: Supervised Learning with Financial Data

```
# Predict portfolio_volatility (continuous variable)
```

```
X = data[['avg_return', 'volatility', 'investment_amount', 'credit_score']]
```

```
y = data['portfolio_volatility']
```

```
# Split Dataset for Regression Task— Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize and train model
```

```
reg_model = LinearRegression()
```

```
reg_model.fit(X_train, y_train)
```

```
# Predictions
```

```
y_pred = reg_model.predict(X_test)
```

Lab Activity: Supervised Learning with Financial Data

```
# Compute evaluation metrics
```

```
mse = mean_squared_error(y_test, y_pred)
```

```
rmse = np.sqrt(mse)
```

```
r2 = r2_score(y_test, y_pred)
```

```
print("Regression Model Evaluation:")
```

```
print(f"MSE = {mse:.3f}")
```

```
print(f"RMSE = {rmse:.3f}")
```

```
print(f"R2 = {r2:.3f}")
```

```
# Apply 5-Fold Cross Validation
```

```
cv = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
cv_scores = cross_val_score(reg_model, X, y, cv=cv, scoring='r2')
```

```
print("Cross-Validation R2 Scores:", cv_scores)
```

```
print("Average R2:", np.mean(cv_scores).round(3))
```

Lab Activity: Supervised Learning with Financial Data

```
# Preparing data to predict 'risk_level' (categorical variable)
```

```
X_cls = data[['avg_return', 'volatility', 'investment_amount', 'credit_score']]
```

```
y_cls = data['risk_level']
```

```
# Encode categorical target if not numeric
```

```
y_cls = y_cls.astype('category').cat.codes
```

```
# Split into training and test sets
```

```
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(  
    X_cls, y_cls, test_size=0.2, random_state=42)
```

```
# Feature Scaling
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train_cls)
```

```
X_test_scaled = scaler.transform(X_test_cls)
```

Lab Activity: Supervised Learning with Financial Data

```
# Initialize KNN classifier (start with k=5)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train_cls)
```

```
# Predict
y_pred_cls = knn.predict(X_test_scaled)
```

```
# Accuracy
acc = accuracy_score(y_test_cls, y_pred_cls)
print(f"KNN Accuracy (k=5): {acc:.3f}")
```

```
# Apply cross-validation on KNN
cv_scores_knn = cross_val_score(knn, X_cls, y_cls, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy Scores:", cv_scores_knn)
print("Average Accuracy:", np.mean(cv_scores_knn).round(3))
```

Lab Activity: Supervised Learning with Financial Data

KNN Model Comparison (Different K Values)

```
k_values = [3, 5, 7, 9, 11]
```

```
acc_scores = []
```

```
for k in k_values:
```

```
    model = KNeighborsClassifier(n_neighbors=k)
```

```
    model.fit(X_train_scaled, y_train_cls)
```

```
    pred = model.predict(X_test_scaled)
```

```
    acc = accuracy_score(y_test_cls, pred)
```

```
    acc_scores.append(acc)
```

```
plt.plot(k_values, acc_scores, marker='o')
```

```
plt.title("KNN Accuracy for Different K Values")
```

```
plt.xlabel("Number of Neighbors (K)")
```

```
plt.ylabel("Accuracy")
```

```
plt.grid(True)
```

```
plt.show()
```

```
best_k = k_values[np.argmax(acc_scores)]
```

```
print(f"Best K Value: {best_k}, Accuracy: {max(acc_scores):.3f}")
```

Lab Activity: Supervised Learning with Financial Data

Summary of Results

```
print("==== Regression Results ====")
print(f"MSE: {mse:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"R2: {r2:.3f}")

print("\n==== KNN Classification Results ====")
print(f"Best K: {best_k}")
print(f"Accuracy: {max(acc_scores):.3f}")
```

Recap

- Introduced supervised learning for prediction and classification using labeled financial data.
- Applied Linear Regression to predict portfolio volatility.
- Evaluated performance using MSE, RMSE, and R^2 metrics.
- Used 5-Fold Cross-Validation to ensure model reliability.
- Implemented K-Nearest Neighbors (KNN) for classifying portfolio risk levels.
- Highlighted the importance of feature scaling and model comparison.
- Introduced decision tree concept

References

- Klaas, J. (2021). Machine Learning for Finance: Principles and Practice for Financial Insiders. Wiley.
- López de Prado, M. (2018). Advances in Financial Machine Learning. Wiley.
- Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (3rd ed.). O'Reilly Media.
- Raschka, S., & Mirjalili, V. (2019). Python Machine Learning (3rd ed.). Packt Publishing.
- Scikit-learn Documentation. (2025). Supervised Learning Overview. https://scikit-learn.org/stable/supervised_learning.html
- IBM Skills Network. (2024). Machine Learning Foundations for Finance — Course Materials.