SEN605 · WEEK 5

Machine Learning for Financial Classification

Logistic Regression · Random Forest · XGBoost · Look-ahead bias · Classification metrics

Instructor
Assoc. Prof. Dr. Onur POLAT

Course
SEN605 — Elective

Prerequisite
Weeks 1–4 completed

Language
English

Graduate School of Informatics· Systems Engineering Program Hacettepe University

Week 5 — Agenda

What we'll cover today

01 Classification Problem Framework

02 Look-Ahead Bias

03 Logistic Regression

04 Random Forest

05 XGBoost

06 Model Evaluation Metrics

07 Pipeline: Walk-Forward Validation

08 Application: Equity Direction Prediction

3

Week 4 to Week 5 — Bridge

From statistical modeling to machine learning

📊 GARCH Completed

Conditional volatility σ_t, VaR estimation and regime analysis completed — these become model features

⚙️ Ready Feature Set

ret_1d, vol_21d, RSI, MACD, vol_ratio + σ_t (GARCH) — model inputs are ready

🎯 New Problem Type

Regression (predict a number) → Classification (predict direction: up/down)

🔮 Today's Goal

Build an ML classifier that correctly predicts the direction (↑/↓) of tomorrow's close

⚠️ **Critical Difference:** In financial ML the test set is the future — leakage from training data has catastrophic consequences. Today's most important message: time-series validation.

4

Financial Classification Problem

Target variable · Feature set · Class imbalance

Target Variable (Y): $y_t = 1$ if $r_{t+1} > 0$ (tomorrow up) · $y_t = 0$ if $r_{t+1} \leq 0$ (tomorrow down)

Feature Set (X)

- Technical indicators: RSI, MACD, Bollinger position
- Return features: ret_1d, ret_5d, ret_21d, ret_63d
- Volatility: vol_10d, vol_21d, vol_ratio, σ_t (GARCH)
- Volume: volume_ratio (today vs 20-day average)
- Macro: VIX, TED spread, short/long rate differential

Class Imbalance

- Equity markets roughly 53% up / 47% down
- Class weighting: class_weight='balanced'
- SMOTE: synthetic oversampling of the minority class
- Threshold tuning: optimize instead of using predict_proba > 0.5
- Metric: accuracy is misleading — use F1 and AUC

Temporal Structure Problem

- Random train/test split is FORBIDDEN — temporal order breaks
- TimeSeriesSplit or Walk-Forward Validation is mandatory
- Leakage risk in feature construction: careful use of rolling
- Lookahead bias: do not normalize before splitting the test set
- Purging: leave a gap between training and test

SECTION 1

Look-Ahead Bias (How future information leaks into training)

The most dangerous trap in financial ML

SEN605 · Week 5

6

Look-Ahead Bias — Definition and Examples

Training a model with information you could never have known in real time

Definition: Look-ahead bias occurs when future information (directly or indirectly) is used in model training or feature construction. Backtest results then show unrealistic performance.

Normalization Leakage

❌
```
# WRONG: fit on all data
scaler = StandardScaler().fit(X)
# → test min/max info leaked into
training!
```

✅
```
# CORRECT: fit on train only
scaler = StandardScaler().fit(X_train)
X_train_s = scaler.transform(X_train)
X_test_s  = scaler.transform(X_test)
```

Rolling-Feature Leakage

❌
```
# WRONG: includes future values
df["ma20"] =
df["Close"].rolling(20).mean()
# the last value of a window=20 is
today's price
# but today's price is in the target!
```

✅
```
# CORRECT: lag with shift
df["ma20"] = df["Close"].rolling(20).mean()
# Feature at t uses only information up to
t-1:
df["ma20_lag"] = df["ma20"].shift(1)
```

Random Train/Test Split

❌
```
# WRONG: sklearn default split
X_tr, X_te, y_tr, y_te =
train_test_split(
    X, y, test_size=0.2, shuffle=True)
# Future data can leak into training!
```

✅
```
# CORRECT: temporal order preserved
cut = int(len(X) * 0.8)
X_tr, X_te = X[:cut], X[cut:]
y_tr, y_te = y[:cut], y[cut:]
```

SECTION 2

Logistic Regression

Baseline linear model for binary classification

SEN605 · Week 5

8

Logistic Regression

Linear decision boundary · Interpretability · Baseline reference model

$$P(y_t=1 \mid X_t) = \sigma(\beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n) = 1 / (1 + e^{-z})$$

σ(z) = sigmoid function · Output: probability in [0,1] · Decision: P > threshold → Class 1

Advantages

- Probabilistic output: P(up) = 72% is interpretable
- Speed: trains in seconds even on large datasets
- Baseline model: other models must beat this
- Coefficients are interpretable: which feature matters how much
- Regularization: L1 (Lasso) for feature selection

Disadvantages

- Linear decision boundary — cannot capture non-linear relationships
- Sensitive to feature engineering — interaction terms must be

 # Logistic Regression pipeline

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report

# Prepare the feature matrix and target
X = df[['ret_1d','ret_5d','ret_21d',
        'vol_21d','rsi','macd',
        'bb_pos','ma10_oran']].dropna()
y = (df['log_ret'].shift(-1) > 0).astype(int)
X, y = X.align(y, join='inner', axis=0)

# Time-based split
n = len(X)
X_tr = X.iloc[:int(n*0.8)]
X_te = X.iloc[int(n*0.8):]
```

SECTION 3

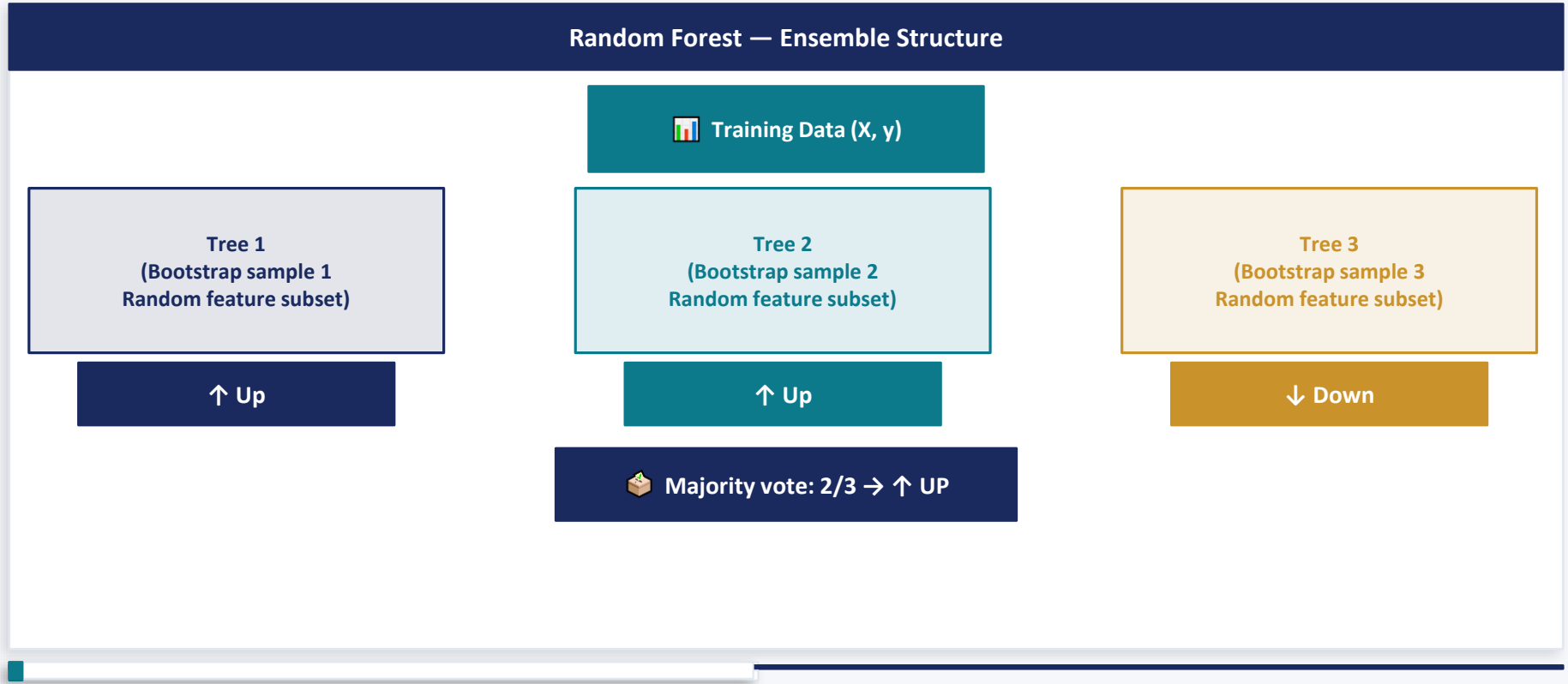
Random Forest

Decision-tree ensemble · Feature importance ranking

SEN605 · Week 5

Random Forest — Decision-Tree Ensemble

Bagging + random feature selection → strong ensemble model



Random Forest — Python and Feature Importance

Core Hyperparameters

- n_estimators: number of trees (100-500 typical)
- max_depth: depth limit (5-15 in finance)
- max_features: features considered at each split
- min_samples_leaf: minimum observations per leaf (5-20)
- class_weight='balanced': for class imbalance
- random_state: fixed value for reproducibility

Feature Importance

- feature_importances_: mean Gini decrease
- Permutation importance: more reliable but slower
- SHAP values: per-prediction explanations
- Most important feature ≠ most predictive power

```
# Random Forest + Feature Importance

from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

# Model definition and training
rf = RandomForestClassifier(
    n_estimators=300,
    max_depth=8,
    max_features='sqrt',      # try √p features
    min_samples_leaf=10,      # prevents overfitting
    class_weight='balanced',
    n_jobs=-1,                # all CPU cores
    random_state=42
)
pipe_rf = Pipeline([
    ('clf', rf)
])
pipe_rf.fit(X_tr, y_tr)
y_pred_rf = pipe_rf.predict(X_te)
print("RF Classification Report:")
print(classification_report(y_te, y_pred_rf,
    target_names=['Down','Up']))

# Feature importance plot
```

SECTION 4

XGBoost

Gradient Boosting · Sequential tree learning · Financial benchmarking

SEN605 · Week 5

XGBoost — Classification with Gradient Boosting

Chen & Guestrin (2016) — king of tabular data

Random Forest (Parallel)

Trees are trained independently → majority vote. Reduces variance (bagging).

XGBoost (Sequential)

Each tree corrects the errors of the previous → residual learning. Reduces bias (boosting).

XGBoost Advantages

- Generally outperforms Random Forest
- Built-in L1+L2 regularization — overfitting control
- Handles missing values natively
- GPU support: fast on large datasets
- Built-in SHAP integration: model interpretability
- King of tabular data in Kaggle competitions

Core Hyperparameters

- n_estimators, max_depth, learning_rate (eta)
- subsample, colsample_bytree, scale_pos_weight

```
# XGBoost classification

import xgboost as xgb
from sklearn.metrics import roc_auc_score

# Weight for class imbalance
neg = (y_tr == 0).sum()
pos = (y_tr == 1).sum()

xgb_clf = xgb.XGBClassifier(
    n_estimators=500,
    max_depth=4,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    scale_pos_weight=neg/pos,   # imbalance
    use_label_encoder=False,
```

SECTION 5

Model Evaluation Metrics

Accuracy misleads — Confusion matrix, F1, AUC-ROC

SEN605 · Week 5

Classification Metrics

Confusion Matrix · Precision · Recall · F1 · AUC-ROC

Confusion Matrix

Actual \ Predicted	Down	Up
Down	TN 210 — True Negative	FP 45 — False Positive
Up	FN 55 — False Negative	TP 190 — True Positive

Accuracy 400/500 = 80.0%

(TP+TN)/(TP+TN+FP+FN) · Can mislead — do not use under class imbalance!

Precision 190/235 = 80.9%

TP / (TP+FP) · When we predict up, how often are we right?

Recall (Sensitivity) 190/245 = 77.6%

TP / (TP+FN) · What fraction of true up-days did we catch?

F1 Score 2×(0.809×0.776)/(0.809+0.776) = 79.2%

2×(P×R)/(P+R) · Balance of Precision and Recall

AUC-ROC 0.85 → iyi

Area under the curve [0,1] · Threshold-independent model quality

ROC Curve and Threshold Optimization

Choosing the right threshold for financial objectives

Reading the ROC Curve

- X axis: FPR = FP/(FP+TN) — false alarm rate
- Y axis: TPR = TP/(TP+FN) = Recall
- AUC=0.5: random model (diagonal)
- AUC=1.0: perfect model
- AUC > 0.65 is considered good in financial classification
- Threshold choice: optimize F1, Youden J or profit given costs

Financial Threshold Selection

- With trading costs: prioritize precision > recall
- Long-only strategy: maximize recall on up-days
- Balanced strategy: find the threshold that maximizes F1
- Optimal Youden: argmax(TPR – FPR)

 # ROC and metric calculation

```
from sklearn.metrics import (
    roc_curve, roc_auc_score,
    classification_report,
    confusion_matrix, ConfusionMatrixDisplay)
import matplotlib.pyplot as plt

# Probability predictions
y_prob_lr  = pipe_lr.predict_proba(X_te)[:,1]
y_prob_rf  = pipe_rf.predict_proba(X_te)[:,1]
y_prob_xgb = xgb_clf.predict_proba(X_te)[:,1]

# ROC curves
fig, ax = plt.subplots(figsize=(8, 6))
for name, prob in [('LR', y_prob_lr),
                   ('RF', y_prob_rf),
                   ('XGB', y_prob_xgb)]:
    fpr, tpr, _ = roc_curve(y_te, prob)
    auc = roc_auc_score(y_te, prob)
    ax.plot(fpr, tpr, lw=2,
            label=f'{name} (AUC={auc:.3f})')
ax.plot([0,1],[0,1],'--', color='gray', lw=1)
ax.set_xlabel('FPR (False Alarm)');
ax.set_ylabel('TPR (Recall)')
```

SECTION 6

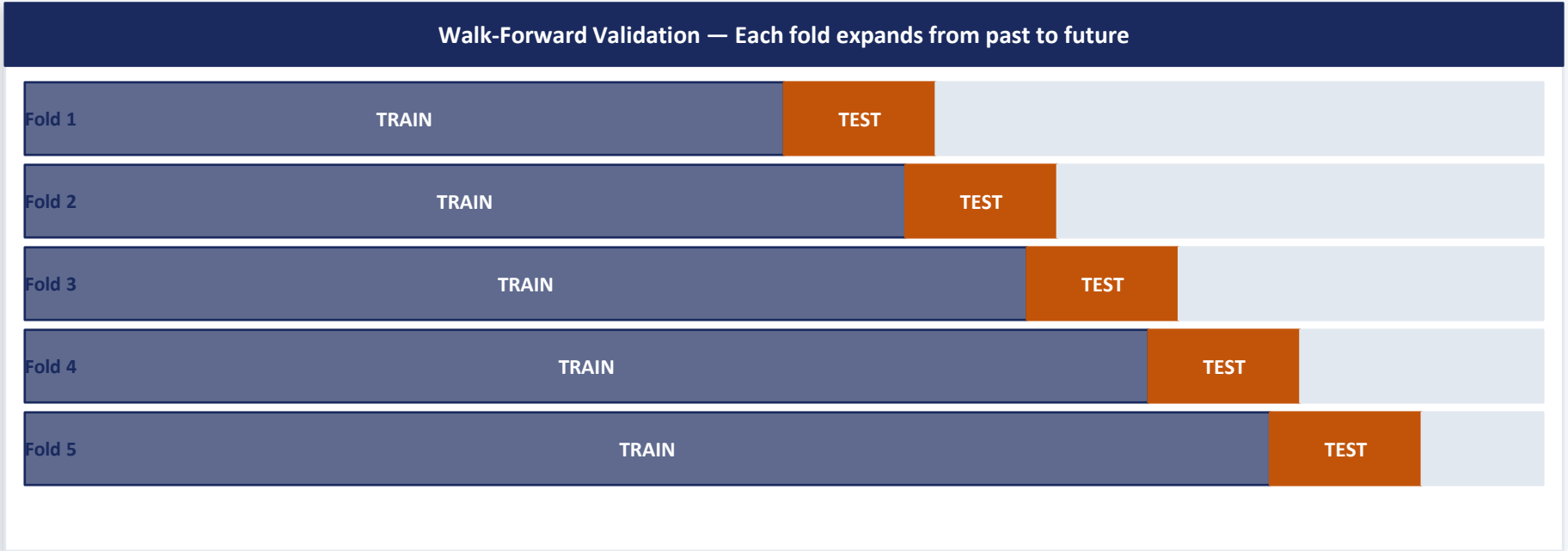
Walk-Forward Validation

The only correct validation method for time series

SEN605 · Week 5

Walk-Forward Validation

Cross-validation method that preserves temporal order




```
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5, gap=1)  # gap: purging — prevents leakage
for fold, (tr_idx, te_idx) in enumerate(tscv.split(X)):
```

Hyperparameter Optimization

Grid Search · Random Search · Optuna Bayesian Optimization

Grid Search

Tries every combination. Exact but very slow — grows exponentially with the number of parameters. Suitable for small grids.

✅ **Exact result** ⚠️ Exponential time

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'clf__n_estimators':[100,300],
    'clf__max_depth':[4,6,8],
    'clf__min_samples_leaf':[5,10]
}
gscv = GridSearchCV(pipe_rf, param_grid,
    cv=TimeSeriesSplit(5), scoring='roc_auc')
gscv.fit(X_tr, y_tr)
print(gscv.best_params_)
```

Optuna (Bayesian)

Performs intelligent search guided by previous results. 10-100× faster than Grid Search for the same quality — ideal in large parameter spaces.

✅ **Fast and smart** ⚠️ Extra dependency

```
import optuna
def objective(trial):
    n_est = trial.suggest_int('n_estimators',100,500)
    depth = trial.suggest_int('max_depth',3,10)
    clf = XGBClassifier(n_estimators=n_est,
        max_depth=depth,random_state=42)
    clf.fit(X_tr, y_tr)
    return roc_auc_score(y_te,
        clf.predict_proba(X_te)[:,1])
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)
print(study.best_params)
```

3-Model Comparison — Summary Table

Performance · Interpretability · Speed · Use case

Criterion	Logistic Reg.	Random Forest	XGBoost
Non-linear relationships	❌ No	✅ Yes	✅ Yes
Interpretability	✅ High (coefficients)	⚠️ Medium (importances)	⚠️ Medium (SHAP)
Training speed	⚡ Very fast	🔷 Parallel-fast	🔷 Sequential-fast
Overfitting risk	Low	Medium (via max_depth)	Medium (via eta, lambda)
Hyperparameter count	Few (C, solver)	Medium (5-6 params)	Many (10+ params)
Class imbalance	class_weight	class_weight	scale_pos_weight
Performance on financial data	Baseline model	Usually better than LR	Usually the best
Recommended use case	First baseline	Shortlist, noisy data	Main model, benchmark

SECTION 7

Application: Equity Direction Prediction

Full ML pipeline — from feature to metric

SEN605 · Week 5

ML Pipeline — Step 1–2: Features and Split

Lab application

1 **Build the Feature Matrix**

```
import yfinance as yf, numpy as np, pandas as pd
from arch import arch_model

idx = yf.download("^GSPC",
    start="2018-01-01", end="2026-10-01",
    auto_adjust=True)['Close'].dropna()
r = np.log(idx / idx.shift(1))

df = pd.DataFrame(index=idx.index)
df['ret_1d']  = r
df['ret_5d']  = np.log(idx / idx.shift(5))
```

2 **Time-Based Train/Test Split and Scaling**

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

features = ['ret_1d','ret_5d','ret_21d',
            'vol_21d','ma10_ratio','rsi','garch_vol']
X = df[features]
y = df['target']

df['garch_vol'] = gjr_conditional_volatility / 100
# Time-based split (80% train, 20% test)
# Target: tomorrow's direction
n = len(X)
```

ML Pipeline — Step 3–4: Model Training and Evaluation

Lab application

3 Train 3 Models

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb

neg, pos = (y_tr==0).sum(), (y_tr==1).sum()

# Logistic Regression
lr = LogisticRegression(C=0.1, class_weight='balanced',
    max_iter=1000, random_state=42)
lr.fit(X_tr_s, y_tr)
```

4 Performance Evaluation

```
from sklearn.metrics import (roc_auc_score,
    classification_report, f1_score)

for name, model in [('LR',lr),('RF',rf),('XGB',xgb_m)]:
    prob  = model.predict_proba(X_te_s)[:,1]
    pred  = (prob > 0.5).astype(int)
    auc   = roc_auc_score(y_te, prob)
    f1    = f1_score(y_te, pred)
    scale_pos_weight=neg/pos, eval_metric='logloss',
    print(f"\n{'='*35}")
    random_state=42, n_jobs=-1)
    print(f"  {name}: AUC={auc:.4f}  F1={f1:.4f}")
```

ML Pipeline — Step 5–6: Walk-Forward and SHAP

Lab application

5 **Walk-Forward Validation**

```
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import roc_auc_score
import warnings; warnings.filterwarnings('ignore')

tscv = TimeSeriesSplit(n_splits=5, gap=5)
wf_results = {m:[] for m in ['LR','RF','XGB']}

for fold, (tr_i, te_i) in enumerate(tscv.split(X)):
    Xtr = scaler.fit_transform(X.iloc[tr_i])
    Xte = scaler.transform(X.iloc[te_i])
    ytr, yte = y.iloc[tr_i], y.iloc[te_i]
```

6 **Feature Importance and SHAP Plot**

```
import shap

# SHAP — for XGBoost
explainer = shap.TreeExplainer(xgb_m)
shap_vals = explainer.shap_values(X_te_s)

# Summary plot
shap.summary_plot(shap_vals, X_te_s,
    feature_names=features,
for m, aucs in wf_results.items():
    plot_type='bar', show=False)
```

From ML Signal to Strategy Backtest

The model predicts correctly — but does it make money?

Backtest Fundamentals

- Trading cost: ~0.1% round-trip for large caps
- Slippage: price impact on large orders
- Short selling: restricted in many markets — be careful
- Look-ahead bias: each decision uses only information at time t
- Overfitting: do not tune on the test set
- Benchmark: compare against buy & hold

Performance Metrics

- Total return: strategy vs. benchmark
- Sharpe ratio: (return − risk-free) / std
- Max Drawdown: largest peak-to-trough decline
- Calmar ratio: annual return / max drawdown

 # Simple backtest

```
# ML signal → long/cash strategy
y_prob = xgb_m.predict_proba(X_te_s)[:,1]
signal = pd.Series((y_prob > 0.5).astype(float),
    index=X_te.index)

# Realized returns (test period)
actual_ret = df.loc[X_te.index, 'ret_1d']

# Strategy: signal=1 → long, signal=0 → cash
strat_ret = signal * actual_ret
# Trading cost (on signal change)
cost = 0.001  # 0.1%
change = signal.diff().abs().fillna(0)
strat_ret -= change * cost

# Benchmark: buy & hold
benchmark_ret = actual_ret

# Cumulative return
cum_strat = (1 + strat_ret).cumprod()
cum_bench = (1 + benchmark_ret).cumprod()

fig, ax = plt.subplots(figsize=(12, 5))
```

Class Imbalance — SMOTE and Weighting

Fix a model that ignores the minority class

Why is Class Imbalance a Problem?

- Equity markets: ~53% up / 47% down — relatively balanced
- Crisis prediction: 2% crisis / 98% normal — severe imbalance
- A naive 'always normal' model scores 98% accuracy
- It never catches real crisis days — F1 near zero
- Accuracy is completely misleading here

Solution Methods

- class_weight='balanced': inverse-frequency weighting
- SMOTE: synthetic oversampling of the minority class
- Undersampling: shrink the majority class
- Threshold change: lower to e.g. predict_proba > 0.3
- Cost-sensitive learning: heavy penalty on false negatives

 # Balance via SMOTE

```
# pip install imbalanced-learn
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline

print("Original class distribution:")
print(pd.Series(y_tr).value_counts())

# SMOTE: add synthetic minority-class samples
smote = SMOTE(sampling_strategy=0.8,
    random_state=42, k_neighbors=5)

# imbalanced-learn Pipeline (SMOTE on training only)
pipe_smote = ImbPipeline([
    ('scaler',  StandardScaler()),
    ('smote',   smote),
    ('clf',     xgb.XGBClassifier(
        n_estimators=300, max_depth=4,
        learning_rate=0.05, random_state=42))
])
pipe_smote.fit(X_tr, y_tr)
y_pred_sm = pipe_smote.predict(X_te)

print("\nClassification report after SMOTE:")
```

SHAP Values — Model Interpretability

Explaining individual predictions with Shapley values

What is SHAP?

- Shapley values: derived from cooperative game theory
- Computes each feature's contribution to each prediction
- Global: feature importance across the full test set
- Local: explain a single observation's prediction
- Model-agnostic but fast on tree models (TreeExplainer)

SHAP Plot Types

- summary_plot: SHAP distribution across features
- bar_plot: mean |SHAP| — global importance
- waterfall_plot: single-prediction explanation
- force_plot: interactive individual prediction
- dependence_plot: feature vs. SHAP relationship

```
# SHAP analysis

import shap

# TreeExplainer for XGBoost (fast)
explainer = shap.TreeExplainer(xgb_m)
shap_vals = explainer.shap_values(X_te_s)

# 1) Summary plot (beeswarm)
shap.summary_plot(shap_vals, X_te_s,
    feature_names=features, show=False)
plt.title('SHAP Beeswarm — XGBoost')
plt.tight_layout(); plt.show()

# 2) Bar plot (global importance)
shap.summary_plot(shap_vals, X_te_s,
    feature_names=features,
    plot_type='bar', show=False)
plt.title('SHAP Feature Importance (Mean |SHAP|)')
plt.tight_layout(); plt.show()

# 3) Explain a single prediction (waterfall)
i_sample = 42  # 42nd test observation
shap.waterfall_plot(
    shap.Explanation(
```

Multi-Asset ML Strategy — Portfolio Ranking

Predict for each stock → pick the strongest signals

Portfolio Ranking Approach

- Predict with XGBoost for each stock in a selected universe
- Rank P(up) probabilities
- Go long on the top-N stocks (equal weight)
- Rebalance daily or weekly
- Strategy return: average return of selected stocks
- Long-short: top-N long + bottom-N short

 # Multi-stock signal

```
# XGBoost prediction per stock
symbols = {'AAPL':'AAPL','MSFT':'MSFT',
    'NVDA':'NVDA','JPM':'JPM','XOM':'XOM'}

predictions = {}
for name, ticker in symbols.items():
    raw = yf.download(ticker, start="2020-01-01",
        end="2024-12-31",
auto_adjust=True)['Close'].dropna()
    # ... feature construction + XGBoost prediction
```

Practical Constraints and Caveats

- Trading costs and slippage: severe for small-cap stocks — prefer large, liquid names
- Data quality: historical equity data has gaps and errors — careful cleaning is critical
- Regime shift: market structure changed significantly after 2020 — periodic retraining is essential
- Regulation: short-sale restrictions and leverage limits in regulated markets bound the strategy's design space

Common Pitfalls in Financial ML

Models that shine in backtest but fail in production

1. Data Leakage

Occurs when normalization, feature selection or hyperparameter tuning uses the test set. The model 'learns' information that is not really available — fails in production.

✅ Prevention: Pipeline: fit() on train only · Optuna/GridSearch: nested CV · Feature selection: on training set

2. Overfitting

Trees too deep, too many features, too few observations. Train AUC=0.95 but test AUC=0.52 → essentially random.

✅ Prevention: keep max_depth small · increase min_samples_leaf · use walk-forward validation · Occam's razor: prefer simpler models

3. Multiple Testing

Testing 100 different feature combinations guarantees a good result by chance. Each test has a 0.05 false-discovery rate — one out of 100 will look 'significant'.

✅ Prevention: pre-register hypotheses · Bonferroni correction · never use out-of-sample test set for tuning

4. Survivorship Bias

Working only with surviving stocks (ignoring delistings). The historical S&P 500 composition contains only successful stocks.

✅ Prevention: include delisted stocks · use point-in-time data from backtest providers

Preparing for Week 6 — LSTM for Time Series Prediction

From classical ML to deep learning

Criterion	Classical ML (RF/XGB)	LSTM (Deep Learning)
Temporal dependency	Manual lag features required	Learns sequences automatically
Sequence length	Fixed window size	Variable-length sequences
Long-term memory	Short windows — limited	LSTM cell — long memory
Data requirement	Hundreds of observations suffice	Thousands of observations ideal
Interpretability	Explainable via SHAP	Hard — attention layers help
Training time	Seconds–minutes	Minutes–hours (GPU recommended)

Preparing for Week 6: building an LSTM in PyTorch, preparing sequence data (sliding window), and the training loop. Update your environment now: pip install torch

Week 5 Summary

What we learned this week — foundation for Week 6

Problem Framework

- Direction prediction: target $y_t = 1\{r_{t+1}>0\}$
- Feature set: technical + volatility + GARCH
- Class imbalance: class_weight and SMOTE

Look-Ahead Bias

- Normalization leakage: fit on train only
- Time split: shuffle=False mandatory
- Purging: gap=5 between train and test

Models

- LR: baseline and interpretable
- RF: parallel ensemble, feature importances
- XGBoost: sequential boosting, usually best

Evaluation

- Accuracy misleads → F1 and AUC-ROC
- Walk-forward: 5-fold temporal CV
- SHAP: for model interpretability

Week 6: Deep Learning for Financial Prediction — LSTM, GRU and Time Series Modeling with PyTorch

Week 5 Assignment

1	For the S&P 500 (or an index of your choice) build a feature matrix containing ret_1d, ret_5d, vol_21d, RSI and MACD. Target: direction of tomorrow's log-return (1/0). Use a time-based 80/20 split.	20 points
2	Train Logistic Regression and Random Forest models. For each, report the confusion matrix, Precision, Recall, F1 and AUC-ROC on the test set. Which model is better?	30 points
3	Apply 5-fold walk-forward validation with XGBoost (TimeSeriesSplit). Report AUC for each fold and the mean ± std. Show the top-5 features with a SHAP bar plot.	30 points
4	Using the best model's signal, run a simple long/cash strategy backtest. Plot cumulative returns against the buy & hold benchmark. Compute the Sharpe ratio.	20 points