Systematic Trading Strategies with Machine Learning Algorithms

Final Exam

06-18-2025

The purpose of this exam is to evaluate your understanding of systematic trading strategies using machine learning techniques and your ability to apply the concepts covered in this course. The exam consists of two independent sections: financial market regime switching strategies using Hidden Markov Models and metamodel trade filtering, and time series forecasting with clustering and neural network architectures for multiple series.

Instructions:

- Read all questions carefully before answering
- Show all mathematical derivations and algorithmic steps to receive full credit
- Answer questions in order as some concepts build upon previous ones
- Calculators are permitted for numerical computations

Exam Structure: (Total marks: 100)

- **Section 1: Regime Switching Trading Strategy (60 marks)**
  - Exercise 1: Hidden Markov Models for regime detection (5 questions)
  - Exercise 2: Metamodel framework for trade filtering (10 questions)
- **Section 2: Multiple Time Series Forecasting with Machine Learning (40 marks)**
  - Exercise 3: Time series clustering with K-means and GMMs (5 questions)
  - Exercise 4: Regression models and Temporal Fusion Transformer (5 questions)

Note: Each question is worth 4 marks.
Time allowed: 2 hours

Section 1: A Regime Switching Trading Strategy

Introduction

Financial markets exhibit different behaviors or "regimes" over time. Identifying these regimes and predicting transitions between them can yield significant advantages in developing robust trading strategies.

In this section, we explore a comprehensive approach to regime-based trading that consists of two major components:

1. **Primary Model Using Hidden Markov Models (HMMs)**: We begin by implementing a regime detection framework based on HMMs, which excel at modeling sequential data with latent states. This model will identify market regimes from time series data and predict regime transitions.
2. **Metamodel Framework for Trade Filtering**: To enhance our primary model's performance, we develop a second-layer metamodel that filters trading signals to maximize profitability. This metamodel framework encompasses:
   - Application of the triple barrier labeling method to generate meta-labels for training
   - Design of a comprehensive metamodel framework.
   - Implementation of ensemble learning techniques, incorporating both bagging and boosting classifiers
   - Evaluation of the metamodel's impact on overall trading performance

The combination of these two models creates a trading system that can both identify regime changes and selectively execute trades with higher probability of success.

Exercise 1 (Creating a primary model using Hidden Markov Models)**.**

In quantitative finance, financial time series often exhibit distinct regimes or states, characterized by different statistical properties. Hidden Markov Models provide a powerful framework for detecting and modeling these unobservable market regimes, which can then inform trading decisions.

In this exercise, we analyze financial returns data through windows to capture temporal dynamics. Let us formalize our approach:

Let $\{r_i^t\}_{i=1}^T$ be a sequence of financial returns within window w_t, where T is the size of the window. From these returns, we construct feature vectors $X^t \in \mathbb{R}^D$ that capture relevant information for each window w_t. Each vector X^t represents the time-frequency content extracted from the returns in window w_t.

We collect vectors X^t from time $t = 1$ to $t = T_f$. For model training, we employ a rolling window approach of size L, where data from time $t = L + 1$ to $t = T_f$ constitutes our test set.

We define sequences of vectors $(S_l)_{1 \le l \le n}$ of size L (the rolling window) as follows:

$$
\begin{aligned}
S_1 &: X_1, X_2, \ldots, X_L \\
S_2 &: X_2, X_3, \ldots, X_{L+1} \\
&\vdots \\
S_l &: X_l, X_{l+1}, \ldots, X_{L+l-1} \\
&\vdots \\
S_n &: X_n, X_{n+1}, \ldots, X_{L+n-1}
\end{aligned}
$$

Where $n = T_f - L + 1$. For clarity of notation, when working with a specific sequence S_l, we will rewrite it as $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_L$ with corresponding hidden states $H_1, H_2, \ldots, H_L$.

1. Parameterization of Hidden Markov Models

We aim to model the regime-switching behavior using an HMM with M distinct hidden states, each representing a different market regime. Each regime is characterized by Gaussian emission distributions that generate the observed feature vectors.

For each sequence S_l, our HMM is parameterized by $\theta^{(l)} = \left(\pi^{(l)}, Q^{(l)}, \mu^{(l)}, \Sigma^{(l)}\right)$:

- Initial state probability vector $\pi^{(l)}$
- State transition matrix $Q^{(l)}$
- Mean vectors $\mu^{(l)} = (\mu_1^{(l)}, \mu_2^{(l)}, \ldots, \mu_M^{(l)})$ where each $\mu_m^{(l)}$ corresponds to hidden state m
- Covariance matrices $\Sigma^{(l)} = (\Sigma_1^{(l)}, \Sigma_2^{(l)}, \ldots, \Sigma_M^{(l)})$ where each $\Sigma_m^{(l)}$ corresponds to hidden state m

Q1. Provide a comprehensive description of the parameters $\pi^{(l)}$, $Q^{(l)}$, $\mu^{(l)}$, and $\Sigma^{(l)}$ within the Hidden Markov Model framework. For each parameter, specify its exact dimensions and explain the mathematical constraints it must satisfy (such as non-negativity, normalization, or positive definiteness).

In the context of Hidden Markov Models, the parameters $\pi^{(l)}$, $Q^{(l)}$, $\mu^{(l)}$, and $\Sigma^{(l)}$ represent:

- **1. Initial state probability vector $\pi^{(l)}$:**
  - Represents the probability distribution over hidden states at time $t = 1$
  - Dimensions: $\pi^{(l)} \in \mathbb{R}^M$ (vector of length M, where M is the number of hidden states)
  - Constraints:
    - Non-negativity: $\pi_i^{(l)} \ge 0$ for all $i \in \{1, ..., M\}$
    - Normalization: $\sum_{i=1}^{M} \pi_i^{(l)} = 1$
- **2. State transition matrix $Q^{(l)}$:**
  - Represents the probability of transitioning from one hidden state to another
  - $Q_{ij}^{(l)}$ is the probability of transitioning from state i to state j
  - Dimensions: $Q^{(l)} \in \mathbb{R}^{M \times M}$ (square matrix of size $M \times M$)
  - Constraints:
    - Non-negativity: $Q_{ij}^{(l)} \ge 0$ for all $i, j \in \{1, ..., M\}$
    - Row-stochasticity: $\sum_{j=1}^{M} Q_{ij}^{(l)} = 1$ for all $i \in \{1, ..., M\}$
- **3. Mean vectors $\mu^{(l)}$:**
  - Represents the mean of the Gaussian emission distribution for each hidden state
  - $\mu_m^{(l)}$ is the mean vector for hidden state m
  - Dimensions: $\mu_m^{(l)} \in \mathbb{R}^D$ for each $m \in \{1, ..., M\}$
    - Collection: $\mu^{(l)} = \{\mu_1^{(l)}, \mu_2^{(l)}, ..., \mu_M^{(l)}\}$
  - Constraints: None (can take any real values)
- **4. Covariance matrices $\Sigma^{(l)}$:**
  - Represents the covariance matrix of the Gaussian emission distribution for each hidden state
  - $\Sigma_m^{(l)}$ is the covariance matrix for hidden state m
  - Dimensions: $\Sigma_m^{(l)} \in \mathbb{R}^{D \times D}$ for each $m \in \{1, ..., M\}$
    - Collection: $\Sigma^{(l)} = \{\Sigma_1^{(l)}, \Sigma_2^{(l)}, ..., \Sigma_M^{(l)}\}$
  - Constraints:
    - Symmetry: $\Sigma_m^{(l)} = (\Sigma_m^{(l)})^T$ for all $m \in \{1, ..., M\}$
    - Positive definiteness: $x^T \Sigma_m^{(l)} x > 0$ for all non-zero $x \in \mathbb{R}^D$ and all $m \in \{1, ..., M\}$

2. Training the HMM

For each sequence S_l, we need to estimate the parameter set $\theta^{(l)} = \{\pi^{(l)}, Q^{(l)}, \mu^{(l)}, \Sigma^{(l)}\}$ that maximizes the likelihood of observing the sequence given the model.

Q2. The Expectation-Maximization (EM) algorithm, specifically the Baum-Welch algorithm, is used for training HMMs. Provide a comprehensive description of this algorithm.

The Baum-Welch algorithm is an application of the EM algorithm for HMMs. It iteratively estimates parameters $\theta = \{\pi, Q, \mu, \Sigma\}$ by maximizing the likelihood of observed data. **Algorithm Overview:**

i. **Initialization:** Choose initial parameters $\theta^{(0)}$

ii. **E-step:** Compute expected sufficient statistics using the Forward-Backward algorithm:

- Calculate forward variables α_t and backward variables β_t
- Compute filtering probabilities: $\xi_t = \frac{\alpha_t}{\mathbb{1}_M^T \alpha_t}$
- Compute smoothing probabilities: $\psi_t = \frac{\alpha_t \circ \beta_t}{\mathbb{1}_M^T \alpha_T}$
- Compute joint smoothing probabilities: $\phi_t = \frac{\mathrm{diag}(\alpha_t) Q \Gamma(t+1) \mathrm{diag}(\beta_{t+1})}{\mathbb{1}_M^T \alpha_T}$

iii. **M-step:** Update parameters using the computed probabilities:

$$\pi_h^{(i+1)} = \psi(1,h)$$
$$Q_{h,h'}^{(i+1)} = \frac{\sum_{t=1}^{L-1} \phi(t,h,h')}{\sum_{t=1}^{L-1} \psi(t,h)}$$
$$\mu_h^{(i+1)} = \frac{\sum_{t=1}^{L} \psi(t,h) \tilde{X}_t}{\sum_{t=1}^{L} \psi(t,h)}$$
$$\Sigma_h^{(i+1)} = \frac{\sum_{t=1}^{L} \psi(t,h)(\tilde{X}_t - \mu_h^{(i+1)})(\tilde{X}_t - \mu_h^{(i+1)})^T}{\sum_{t=1}^{L} \psi(t,h)}$$

iv. **Convergence:** Repeat E-step and M-step until convergence (log-likelihood change $<$ threshold)

The algorithm guarantees monotonic increase in the likelihood and converges to a local maximum.

3. Using a Rolling Window to Predict the Next Hidden State

After training the HMM on a sequence S_l, our objective is to predict the probability distribution over all possible hidden states at the next time step, that is, to compute $p(H_{L+1} = h | \tilde{X}_{1:L})$ for all $h \in \{1, 2, \ldots, M\}$, as shown in Figure 1. This probability distribution over possible regimes will inform our trading decisions.

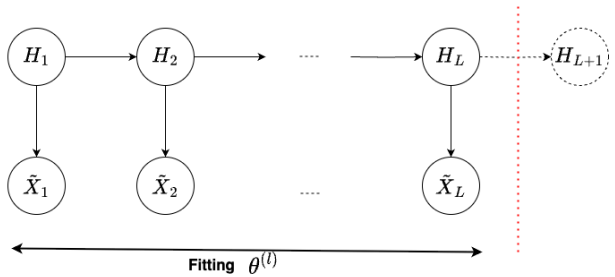

Figure 1: The HMM prediction problem

We introduce the following notations for our approach:

- **Filtering probabilities**: $\xi \in \mathbb{R}^{L \times M}$

$$\forall (t,h) \in [\![1,L]\!] \times [\![1,M]\!], \quad \xi(t,h) := p(H_t = h \mid \tilde{X}_1, \ldots, \tilde{X}_t)$$

- We define the **alpha variables**, which represent the joint probability of the observed sequence up to time t and the hidden state at time t:

$$\forall (t,h) \in [\![1,L]\!] \times [\![1,M]\!], \quad \alpha(t,h) := p(\tilde{X}_1, \ldots, \tilde{X}_t, H_t = h)$$

- We also define the **emission tensor** $\Gamma(t)$, a diagonal matrix encoding the likelihood of observing $\tilde{X}_t$ given each possible hidden state:

$$\forall t \in [\![1,L]\!] : \Gamma(t) := \begin{pmatrix} p(\tilde{X}_t \mid H_t = 1) & & \\ & \ddots & \\ & & p(\tilde{X}_t \mid H_t = M) \end{pmatrix} \in \mathbb{R}^{M \times M}$$

- We use vector notation for compact representation:

$$\forall t \in [\![1,L]\!] \quad \alpha_t = (\alpha(t,1), \ldots, \alpha(t,M))^T \in \mathbb{R}^M$$
$$\forall t \in [\![1,L]\!] \quad \xi_t = (\xi(t,1), \ldots, \xi(t,M))^T \in \mathbb{R}^M$$

For Forward Propagation (recursive calculation of α variables), we have:

$$\alpha_1 = \Gamma(1)\pi, \quad \forall t \geq 2 : \alpha_t = \Gamma(t) Q^T \alpha_{t-1}$$

Q3. Rigorously prove that filtering probabilities can be derived from alpha variables using:

$$\forall t \in [\![1,L]\!] \quad \xi_t = \frac{\alpha_t}{\mathbb{1}_M^T \alpha_t}$$

$$
\begin{aligned}
\xi(t,h) &= p(H_t = h \mid \tilde{X}_{1:t}) \\
&= \frac{p(H_t = h, \tilde{X}_{1:t})}{p(\tilde{X}_{1:t})} \quad \text{(by Bayes' rule)} \\
&= \frac{\alpha(t,h)}{p(\tilde{X}_{1:t})} \quad \text{(by definition of } \alpha) \\
&= \frac{\alpha(t,h)}{\sum_{h'=1}^{M} p(H_t = h', \tilde{X}_{1:t})} \quad \text{(by law of total probability)} \\
&= \frac{\alpha(t,h)}{\sum_{h'=1}^{M} \alpha(t,h')} \\
&= \frac{\alpha(t,h)}{\mathbb{1}_M^T \alpha_t}
\end{aligned}
$$

where $\mathbb{1}_M^T \alpha_t = \sum_{h'=1}^{M} \alpha(t,h')$ represents the sum of all components of α_t. Therefore, in vector form: $\xi_t = \frac{\alpha_t}{\mathbb{1}_M^T \alpha_t}$

Q4. Develop a detailed algorithm for the Forward algorithm to calculate alpha variables and filtering probabilities:

Algorithm 1 Forward Algorithm for HMM

Require: π, Q, $\Gamma(1), \ldots, \Gamma(L)$
Ensure: α variables and filtering probabilities ξ
1: **...** ▷ Fill in the blank
2: **return** α, ξ

Algorithm 2 Forward Algorithm for HMM

Require: π, Q, $\Gamma(1), \ldots, \Gamma(L)$
Ensure: α variables and filtering probabilities ξ
1: **Initialize:** $\alpha_1 = \Gamma(1)\pi$
2: $\xi_1 = \frac{\alpha_1}{\mathbb{1}_M^T \alpha_1}$
3: **for** $t = 2$ to L **do**
4: $\quad \alpha_t = \Gamma(t) Q^T \alpha_{t-1}$
5: $\quad \xi_t = \frac{\alpha_t}{\mathbb{1}_M^T \alpha_t}$
6: **end for**
7: **return** α, ξ

Q5. Using the filtering probabilities $\xi(L, \cdot)$ and the transition matrix Q, derive the formula for predicting the next hidden state $p(H_{L+1} = h \mid \tilde{X}_{1:L})$ for all $h \in [\![1, M]\!]$.

$$
\begin{aligned}
p(H_{L+1} = h \mid \tilde{X}_{1:L}) &= \sum_{h'=1}^{M} p(H_{L+1} = h, H_L = h' \mid \tilde{X}_{1:L}) \\
&= \sum_{h'=1}^{M} p(H_{L+1} = h \mid H_L = h', \tilde{X}_{1:L}) \cdot p(H_L = h' \mid \tilde{X}_{1:L}) \\
&= \sum_{h'=1}^{M} p(H_{L+1} = h \mid H_L = h') \cdot p(H_L = h' \mid \tilde{X}_{1:L}) \\
&\quad \text{(by the Markov property)} \\
&= \sum_{h'=1}^{M} Q_{h',h} \cdot \xi(L, h')
\end{aligned}
$$

In vector form: $p(H_{L+1} = \cdot \mid \tilde{X}_{1:L}) = Q^T \xi_L$ Therefore, the prediction for the next hidden state is given by:

$$
p(H_{L+1} = h \mid \tilde{X}_{1:L}) = \sum_{h'=1}^{M} Q_{h',h} \cdot \xi(L, h')
$$

The hidden states in our model correspond to different market regimes characterized by distinct risk-return profiles, as measured by their Sharpe ratios. The transition matrix Q typically suggests high stability in the hidden states, meaning regimes tend to persist over time.

We translate these regime predictions into actionable trading signals as follows:

- A prediction of -1 corresponds to a short position
- A prediction of $+1$ corresponds to a long position

The specific regime-to-signal mapping depends on the characteristics of each regime (e.g., mean return, volatility, Sharpe ratio).

The next section will address the critical objective of filtering out non-profitable trades from this primary strategy using tree-based models. This metamodel layer will help identify which signals from the HMM are most likely to result in profitable trades, thereby improving the overall performance of the trading system.

Exercise 2 (Introducing a Metamodel to Filter Out Non-Profitable Trades)**.**

The primary HMM model generates trading signals based on detected hidden regimes, but not all signals result in profitable trades. To improve the strategy's

performance, we introduce a metamodel that predicts which trades are likely to be profitable. The metamodel acts as a filter, learning to identify market conditions where the primary model's predictions are reliable.

Key distinction:

- The **Primary Model** is trained to detect hidden regimes and generate trading signals.
- The **Metamodel** is trained to detect the conditions under which the primary model works well.

The metamodel, represented in Figure 2 uses different features than the primary model to avoid learning redundant information. It learns the environments where the primary model has alpha.

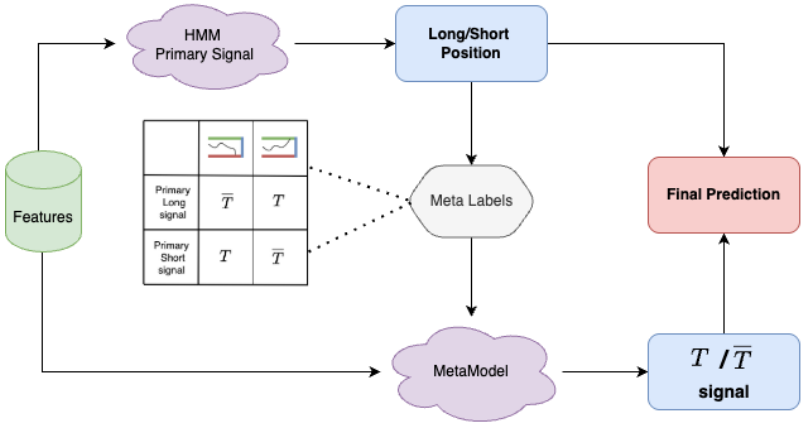

Figure 2: The Metamodel framework

1. **The Triple Barrier Labeling Method**

To train our metamodel, we need to create labels that indicate whether a trade would have been profitable. We use the triple barrier method, which sets three exit conditions for each trade:

- Upper horizontal barrier: profit-taking level
- Lower horizontal barrier: stop-loss level
- Vertical barrier: time-based exit (labeled based on the sign of the return)

The width of the horizontal barriers (i.e., their distance from the entry price) is scaled proportionally to the volatility, ensuring that profit targets and stop-loss levels are wider in high-volatility periods and narrower in low-volatility periods. We define a meta-label based on the outcome of each trade:

- T (label 1): If the trade would have been profitable
- $\overline{T}$ (label 0): If the trade would have been unprofitable

Q1. Complete the following table by determining the meta-labels for each situation. Consider both Long and Short positions initiated by the primary model, and all possible barrier hits. For each scenario, specify the meta-label name (what the label represents) and fill in the corresponding value $l_i \in \{0, 1\}$:

Primary Signal	Barrier Hit	Meta-label
Long Position	Upper horizontal barrier	l_1
Long Position	Lower horizontal barrier	l_2
Long Position	Vertical barrier (positive return)	l_3
Long Position	Vertical barrier (negative return)	l_4
Short Position	Upper horizontal barrier	l_5
Short Position	Lower horizontal barrier	l_6
Short Position	Vertical barrier (positive return)	l_7
Short Position	Vertical barrier (negative return)	l_8

Primary Signal	Barrier Hit	Meta-label
Long Position	Upper horizontal barrier	$l_1 = 1$ (Profitable)
Long Position	Lower horizontal barrier	$l_2 = 0$ (Unprofitable)
Long Position	Vertical barrier (positive return)	$l_3 = 1$ (Profitable)
Long Position	Vertical barrier (negative return)	$l_4 = 0$ (Unprofitable)
Short Position	Upper horizontal barrier	$l_5 = 0$ (Unprofitable)
Short Position	Lower horizontal barrier	$l_6 = 1$ (Profitable)
Short Position	Vertical barrier (positive return)	$l_7 = 0$ (Unprofitable)
Short Position	Vertical barrier (negative return)	$l_8 = 1$ (Profitable)

2. **Using a Random Forest Classifier to Predict the Meta-labels**

We employ a Random Forest (bagging) classifier to map market condition features to the meta-labels. Assume we have 55% profitable trades and 45% unprofitable trades in our historical data.

Q2. List and describe four key hyperparameters of a Random Forest Classifier that should be tuned to optimize performance.

Four key hyperparameters of Random Forest:

- **n_estimators** (Number of trees):
  - Controls the number of decision trees in the forest
  - More trees generally improve performance but increase computational cost
- **max_depth** (Maximum tree depth):
  - Limits how deep each tree can grow.
  - Prevents overfitting by controlling model complexity.
  - Deeper trees can capture more complex patterns but may overfit.
- **min_samples_split** (Minimum samples to split):
  - Minimum number of samples required to split an internal node.
  - Higher values prevent the tree from creating too specific rules.
  - Helps control overfitting by ensuring sufficient data for splits.
- **max_features** (Features per split):
  - Number of features to consider when looking for the best split.
  - Common values: $\sqrt{p}$ for classification.
  - Introduces randomness and decorrelates trees in the forest

Q3. A decision tree within the Random Forest is considering a split on a particular feature. The parent node contains 100 samples with 50% belonging to each class (balanced distribution). After the split, 60% of samples go to the left child (which contains only samples from class 0), and 40% go to the right child (which contains only samples from class 1). Calculate the information gain using entropy as the impurity measure. Show all calculations for entropy before split, entropy after split, and information gain.

Node	P(Label = 1)	P(Label = 0)
Parent	0.5	0.5
Left Child	0.0	1.0
Right Child	1.0	0.0

Step 1: Calculate entropy before split (parent node)

$$H(parent) = 1.0$$

Step 2: Calculate entropy for each child node Left child (60 samples, all class 0):

$$H(left) = 0$$

Right child (40 samples, all class 1):

$$H(right) = 0$$

Step 3: Calculate weighted entropy after split

$$\begin{aligned} H(after) &= \frac{60}{100} \times H(left) + \frac{40}{100} \times H(right) \\ &= 0.6 \times 0 + 0.4 \times 0 \\ &= 0 \end{aligned}$$

Step 4: Calculate information gain

$$\begin{aligned} IG &= H(parent) - H(after) \\ &= 1.0 - 0 \\ &= 1.0 \end{aligned}$$

Conclusion: The information gain is 1.0, which is the maximum possible. This split perfectly separates the two classes.

Q4. To tune the hyperparameters of our Random Forest using cross-validation, we need appropriate evaluation metrics. List and briefly describe four possible evaluation metrics suitable for this binary classification task.

Four evaluation metrics for binary classification:

- **Accuracy**:
  - Formula: $\frac{TP+TN}{TP+TN+FP+FN}$
  - Measures overall correctness of predictions
  - Can be misleading for imbalanced datasets
- **Precision**:
  - Formula: $\frac{TP}{TP+FP}$
  - Proportion of predicted positive cases that are actually positive
  - Important when false positives are costly (e.g., avoiding unprofitable trades)
- **Recall (Sensitivity)**:
  - Formula: $\frac{TP}{TP+FN}$
  - Proportion of actual positive cases correctly identified
  - Important when false negatives are costly (e.g., missing profitable trades)
- **F1 Score**:
  - Formula: $2 \times \frac{Precision \times Recall}{Precision + Recall}$
  - Harmonic mean of precision and recall
  - Provides balanced measure when both false positives and false negatives matter

Q5. Provide a detailed description of the Random Forest training algorithm, including the bootstrap sampling process, random feature selection at each split, tree construction, and final prediction aggregation.

Random Forest Training Algorithm:

- **Bootstrap Sampling:**
  - For each tree $t = 1, ..., T$:
  - Sample N instances with replacement from training set of size N.
- **Tree Construction with Random Feature Selection:**
  - For each bootstrap sample, grow an unpruned decision tree:
  - At each node:
    - Randomly select m features from total p features (typically $m = \sqrt{p}$)
    - Find best split among these m features using impurity measure (e.g., Gini, entropy)
    - Split node using the selected feature and threshold
  - Continue splitting until stopping criteria met (min samples, max depth, etc.)
- **Prediction Aggregation:**
  - For classification: Majority voting

$$\hat{y} = \text{mode}\{h_1(x), h_2(x), ..., h_T(x)\}$$

3. Introducing a Boosting Algorithm as an Alternative Approach

As an alternative to the Random Forest approach, we explore using a boosting algorithm for our metamodel. In this approach, we use the same feature set x_i (technical indicators, market data, etc.) paired with the meta-labels y_i generated by the triple barrier method, where $y_i \in \{0, 1\}$ indicates whether a trade would be unprofitable (0) or profitable (1) for each sample $i = 1, 2, ..., N$. Boosting sequentially builds an ensemble where each model focuses on correcting the errors of previous models, as shown in figure 3.

The boosting algorithm will learn to classify whether the primary model's signals are likely to result in profitable trades.

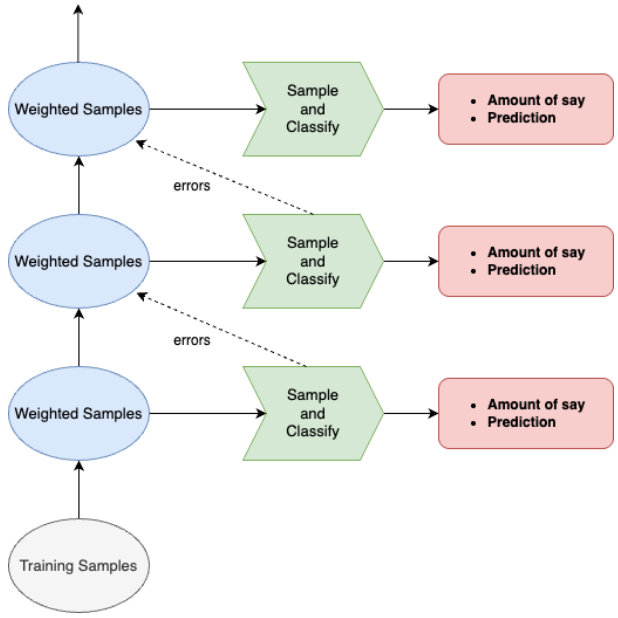

Figure 3: The AdaBoost approach

Q6. In AdaBoost, each weak classifier f_t is assigned a weight α_t that determines its contribution to the final ensemble. Given the weighted error rate ϵ_t, the formula $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ is used. Explain the intuition behind this formula and why classifiers with lower error rates receive higher weights.

- **Mathematical Properties:**
  - When $\epsilon_t < 0.5$ (better than random): $\alpha_t > 0$
  - When $\epsilon_t = 0.5$ (random guessing): $\alpha_t = 0$
  - When $\epsilon_t > 0.5$ (worse than random): $\alpha_t < 0$
  - As $\epsilon_t \to 0$ (perfect classifier): $\alpha_t \to \infty$
  - As $\epsilon_t \to 1$ (always wrong): $\alpha_t \to -\infty$
- **Why this formula?**
  - Derived from minimizing the exponential loss function
  - Ensures that the combined weighted error of the ensemble decreases exponentially

Q7. AdaBoost updates the sample weights $w_t(i)$ after each iteration t to focus on misclassified examples. Describe the weight update rule for transitioning from $w_t(i)$ to $w_{t+1}(i)$ based on whether $f_t(x_i)$ correctly

classifies (x_i, y_i). Explain why this adaptive reweighting mechanism is crucial for boosting's effectiveness.

AdaBoost Weight Update Rule:

- **The update formula:**

$$w_{t+1}(i) = w_t(i) \cdot \exp(\alpha_t(1 - 2\mathbb{I}\{y_i = f_t(x_i)\}))$$

where:
  - $y_i \in \{0, 1\}$ is the true label
  - $f_t(x_i) \in \{0, 1\}$ is the prediction
  - α_t is the classifier weight
  - $\mathbb{I}\{\cdot\}$ is the indicator function (1 if true, 0 if false)
- **Behavior based on classification outcome:**
  - **Correct classification:** $y_i = f_t(x_i)$

$$\mathbb{I}\{y_i = f_t(x_i)\} = 1$$

$$w_{t+1}(i) = w_t(i) \cdot \exp(\alpha_t(1 - 2 \cdot 1)) = w_t(i) \cdot \exp(-\alpha_t)$$

Weight decreases (since $\alpha_t > 0$ for good classifiers)
  - **Misclassification:** $y_i \neq f_t(x_i)$

$$\mathbb{I}\{y_i = f_t(x_i)\} = 0$$

$$w_{t+1}(i) = w_t(i) \cdot \exp(\alpha_t(1 - 2 \cdot 0)) = w_t(i) \cdot \exp(\alpha_t)$$

Weight increases exponentially
- **Normalization:** After updating, weights are normalized: $w_{t+1}(i) = \frac{w_{t+1}(i)}{\sum_j w_{t+1}(j)}$
- **Why this is crucial for boosting:**
  - **Focus on hard examples:** Misclassified samples get exponentially higher weights
  - **Sequential improvement:** Each classifier focuses on errors of previous ones
  - **Diverse weak learners:** Forces each classifier to solve different problems
  - **Ensemble strength:** Combines experts on different parts of the feature space

Q8. When making predictions with an AdaBoost ensemble, the final classification combines all T weak classifiers. Write the complete formula for the final classification rule $H(x)$ in terms of the individual classifiers $f_t(x)$ and their weights α_t. Explain how this weighted combination differs from simple majority voting.

AdaBoost Final Classification Rule:

- **The formula:**

$$f_{boost}(x) = \mathbb{I}\left\{\sum_{t=1}^{T} \alpha_t f_t(x) \geq \frac{1}{2}\sum_{t=1}^{T} \alpha_t\right\}$$

where:
  - $f_t(x) \in \{0, 1\}$ is the prediction of classifier t
  - α_t is the weight of classifier t
  - $\mathbb{I}\{\cdot\}$ is the indicator function returning 0 or 1
- **Difference from simple majority voting:**
  - Simple majority voting: $f_{majority}(x) = \mathbb{I}\{\sum_{t=1}^{T} f_t(x) \geq T/2\}$
  - AdaBoost: Uses weighted sum with performance-based weights α_t
  - Better classifiers have stronger influence
  - Can override many weak classifiers with few strong ones

Q9. Complete the AdaBoost algorithm implementation by filling in the missing steps:

Algorithm 3 AdaBoost Algorithm

Require: Training data $\{(x_i, y_i)\}_{i=1}^N$, $x \in \mathcal{X}$, $y \in \{0, 1\}$, number of iterations T
Ensure: AdaBoost model

1: Initialize weights $w_1(i) = \frac{1}{N}$ for $i = 1 : N$
2: **for** $t = 1$ to T **do**
3: Train classifier f_t on weighted training data with weights w_t
4: Calculate weighted error: **[BLANK 1]** ▷ Fill in the blank
5: Calculate model weight: $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
6: Scale weights: **[BLANK 2]** ▷ Fill in the blank
7: Normalize: **[BLANK 3]** ▷ Fill in the blank
8: **end for**
9: **return** Classification rule: **[BLANK 4]** ▷ Fill in the blank

Completed AdaBoost Algorithm:

- **BLANK 1:** $\epsilon_t = \sum_{i=1}^{N} w_t(i)\mathbb{I}\{y_i \neq f_t(x_i)\}$
- **BLANK 2:** $\hat{w}_{t+1}(i) = w_t(i) \cdot e^{\alpha_t(1-2\mathbb{I}\{y_i=f_t(x_i)\})}$
- **BLANK 3:** $w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_{j=1}^{N} \hat{w}_{t+1}(j)}$
- **BLANK 4:** $f_{boost}(x_0) = \mathbb{I}\{\sum_{t=1}^{T} \alpha_t f_t(x_0) \geq \frac{1}{2}\sum_{t=1}^{T} \alpha_t\}$

4. **Performance Comparison**

After implementing both ensemble methods, we compare two trading strategies. To evaluate their effectiveness, we examine the confusion matrices for each approach:

Model A (Primary Model Alone): This strategy trades whenever the primary HMM model generates a signal (-1 for short, +1 for long). Since we always follow the primary model's signals, we always predict T (trade).

Model B (Primary + Metamodel): This strategy only trades when both the primary model generates a signal AND the metamodel predicts the trade will be profitable. The final decision is the product of the primary signal and the metamodel prediction.

The confusion matrices below show the performance of both models on a test set of 1000 trading opportunities:

Model A (Primary Model Alone):

	Predicted T	Predicted $\overline{T}$
Actual T	550	0
Actual $\overline{T}$	450	0

Model B (Primary + Metamodel):

	Predicted T	Predicted $\overline{T}$
Actual T	500	50
Actual $\overline{T}$	80	370

Q10. Using the confusion matrices provided above, calculate the precision, recall, and F1 score for each model. Show all calculations and provide a conclusion about the effectiveness of the metamodel approach. Discuss the trade-off between missing profitable trades versus avoiding unprofitable ones, and explain why the metamodel significantly improves the overall trading strategy.

Model A Calculations:

- TP = 550, FP = 450, FN = 0, TN = 0
- Precision = $\frac{TP}{TP+FP} = \frac{550}{550+450} = \frac{550}{1000} = 0.55$
- Recall = $\frac{TP}{TP+FN} = \frac{550}{550+0} = \frac{550}{550} = 1.0$
- F1 Score = $2 \times \frac{0.55 \times 1.0}{0.55+1.0} = 2 \times \frac{0.55}{1.55} = 0.71$

Model B Calculations:

- TP = 500, FP = 80, FN = 50, TN = 370
- Precision = $\frac{TP}{TP+FP} = \frac{500}{500+80} = \frac{500}{580} = 0.862$
- Recall = $\frac{TP}{TP+FN} = \frac{500}{500+50} = \frac{500}{550} = 0.909$
- F1 Score = $2 \times \frac{0.862 \times 0.909}{0.862+0.909} = 2 \times \frac{0.784}{1.771} = 0.885$

Comparison Summary:

Model	Precision	Recall	F1 Score
Model A	0.55	1.00	0.71
Model B	0.86	0.91	0.89

Conclusions:

1. **Metamodel Effectiveness:**
   - Model B dramatically improves precision from 55% to 86%
   - F1 score increases from 0.71 to 0.89, showing overall improvement
   - The metamodel successfully filters out 82% of unprofitable trades (370/450)
2. **Trade-off Analysis:**
   - Model A: Takes every trade, capturing all profits but also all losses
   - Model B: Misses 9% of profitable trades (50/550) but avoids 82% of losses
   - The trade-off favors avoiding losses over capturing every profit

Section 2: Forecasting Energy Consumption with Sequential Neural Networks

Introduction

The primary objective of this section is to develop a framework for forecasting multiple time series by first grouping them into meaningful clusters before applying regression models. This clustering-based approach leverages the insight that time series with similar patterns may benefit from specialized forecasting models, ultimately improving prediction accuracy.

We consider a collection of time series $(y_i)_{i \in \mathcal{I}}$, where $\mathcal{I}$ denotes the index set of all available time series. Our methodology consists of:

1. Extracting feature vectors that capture the structural characteristics of each time series
2. Applying clustering algorithms (K-means and Gaussian Mixture Models) to group similar time series
3. Developing forecasting strategies that utilize cluster information through either:
   - Training separate ensemble models (Random Forest) for each cluster
   - Incorporating cluster assignments as features in Sequential Neural Networks (Temporal Fusion Transformer)

Exercise 3 (Feature-driven Time Series Clustering)**.**

Feature-driven time series clustering provides a systematic approach to identify groups of similar time series. We extract features that capture structural characteristics such as trend, seasonality, periodicity, skewness, kurtosis, etc.

For each time series $(y_i)_{i\in\mathcal{I}}$, we compute D statistical features, then apply Principal Component Analysis (PCA) to reduce dimensionality while retaining 90% of the variance. This results in feature vectors $F_i \in \mathbb{R}^d$ where $d << D$, with each time series y_i now represented by its corresponding feature vector F_i.

1. **K-means Clustering of Feature Vectors**

   We apply K-means clustering to the feature vectors $\{F_i\}_{i\in\mathcal{I}}$ to identify K distinct groups of time series.

   Q1. Complete the following K-means algorithm:

Algorithm 4 K-means Algorithm for Time Series Clustering

Require: Feature vectors $\{F_i\}_{i\in\mathcal{I}}$ where $F_i \in \mathbb{R}^d$, number of clusters K
Ensure: Assignment function $\Psi : \mathcal{I} \to \{1, \dots, K\}$ and centroids $c_1, \dots, c_K$

```
Initialize: Choose c_1, ..., c_K from {F_i}_{i∈I} at random
repeat
    for i ∈ I do
        Assign to nearest centroid: [BLANK 1]        ▷ Fill in the blank
    end for
    for k = 1 to K do
        Update centroid: [BLANK 2]                   ▷ Fill in the blank
    end for
until convergence
return Ψ, c_1, ..., c_K
```

- **BLANK 1:** $\Psi(i) = \arg\min_{k\in\{1,\dots,K\}} \|F_i - c_k\|^2$
- **BLANK 2:** $c_k = \frac{1}{\sum_{i\in\mathcal{I}} \mathbb{I}(\Psi(i)=k)} \sum_{i\in\mathcal{I}} \mathbb{I}(\Psi(i) = k) F_i$

Q2. The distortion measure for K-means is defined as:

$$J(\Psi, c) = \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \left\|F_i - c_{\Psi(i)}\right\|^2$$

Explain why this distortion measure decreases monotonically after each complete iteration (assignment step followed by update step) and why this guarantees convergence to a local minimum.

For each iteration t of the algorithm, we define the distortion at time t as:

$$J(\Psi^{(t)}, c^{(t)}) = \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \|F_i - c^{(t)}_{\Psi^{(t)}(i)}\|^2$$

We can show that for all t:

$$J(\Psi^{(t)}, c^{(t)}) \geq J(\Psi^{(t+1)}, c^{(t+1)})$$

We have:

$$\Psi^{(t+1)}(i) = \arg \min_{k\in\{1,\dots,K\}} \|F_i - c^{(t)}_k\|^2$$

So,

$$J(\Psi^{(t)}, c^{(t)}) = \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \|F_i - c^{(t)}_{\Psi^{(t)}(i)}\|^2 \tag{1}$$

$$\geq \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \|F_i - c^{(t)}_{\Psi^{(t+1)}(i)}\|^2 \tag{2}$$

$$= \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \sum_{k=1}^{K} z_i^k \|F_i - c^{(t)}_k\|^2 \tag{3}$$

where $z = (z_i^k)_{(i,k)\in\mathcal{I}\times\{1,\dots,K\}}$ is such that:

$$\forall (i,k) \in \mathcal{I} \times \{1, \dots, K\} \quad z_i^k = \mathbb{I}(\Psi^{(t+1)}(i) = k)$$

We define:

$$\forall c = (c_1, \dots, c_K) \in \mathbb{R}^{K\times d} \quad \mathcal{L}(c) := \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \sum_{k=1}^{K} z_i^k \|F_i - c_k\|^2$$

It's straightforward that:

$$\mathcal{L}(c^{(t)}) = \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \|F_i - c^{(t)}_{\Psi^{(t+1)}(i)}\|^2$$

and

$$\mathcal{L}(c^{(t+1)}) = \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \|F_i - c^{(t+1)}_{\Psi^{(t+1)}(i)}\|^2$$

We wish to minimize $\mathcal{L}$ w.r.t c. We have:

$$\forall k \in \{1, \dots, K\} \quad \nabla_{c_k} \mathcal{L}(c) = \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} z_i^k \nabla_{c_k} (\|F_i - c\|^2)$$

where we can compute:

$$\nabla_{c_k} (\|F_i - c\|^2) = -2(F_i - c)$$

Thus,

$$\forall k \in \{1, \ldots, K\} \quad \nabla_{c_k} \mathcal{L}(c) = \frac{-2}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} z_i^k (F_i - c)$$

Therefore,

$$\begin{aligned} \nabla_c \mathcal{L}(c) = 0 &\iff \forall k \in \{1, \ldots, K\} \quad \nabla_{c_k} \mathcal{L}(c) = 0 \\ &\iff \forall k \in \{1, \ldots, K\} \quad c_k = \frac{\sum_{i \in \mathcal{I}} z_i^k F_i}{\sum_{i \in \mathcal{I}} z_i^k} \\ &\iff \forall k \in \{1, \ldots, K\} \quad c_k = c_k^{(t+1)} \\ &\iff c = c^{(t+1)} \end{aligned}$$

We conclude that

$$\forall c = (c_1, \ldots, c_K) \in \mathbb{R}^{K \times d} \quad \mathcal{L}(c) \geq \mathcal{L}(c^{(t+1)})$$

And therefore:

$$\mathcal{L}(c^{(t)}) \geq \mathcal{L}(c^{(t+1)})$$

From the above equations, we conclude that:

$$J(\Psi^{(t)}, c^{(t)}) \geq J(\Psi^{(t+1)}, c^{(t+1)})$$

Convergence: The number of possible assignments is finite. Thus, there exists t such that

$$J(\Psi^{(t)}, c^{(t)}) = J(\Psi^{(t+1)}, c^{(t+1)})$$

The K-means algorithm stops after a finite number of steps.

2. Gaussian Mixture Models for Soft Clustering

After determining the optimal number of clusters using silhouette scores, we implement Gaussian Mixture Models to obtain probabilistic cluster assignments.

Q3. For a GMM with K components modeling d-dimensional feature vectors:

- Define the complete parameter set $\theta = (\pi, \mu, \Sigma)$ where:
  - $\pi = (\pi_1, \ldots, \pi_K)$ are the mixture weights
  - $\mu = (\mu_1, \ldots, \mu_K)$ are the component means
  - $\Sigma = (\Sigma_1, \ldots, \Sigma_K)$ are the covariance matrices
- Specify the shape of each parameter
- List all constraints that must be satisfied by these parameters

Parameter shapes:

- $\pi \in \mathbb{R}^K$ (vector of length K)
- $\mu_j \in \mathbb{R}^d$ for each $j \in \{1, \ldots, K\}$, so $\mu \in \mathbb{R}^{K \times d}$
- $\Sigma_j \in \mathbb{R}^{d \times d}$ for each $j \in \{1, \ldots, K\}$, so $\Sigma \in \mathbb{R}^{K \times d \times d}$

Constraints:

- **Mixture weights π:**
  - Non-negativity: $\pi_j \geq 0$ for all $j \in \{1, \ldots, K\}$
  - Normalization: $\sum_{j=1}^K \pi_j = 1$
- **Means μ:** No constraints (can take any real values)
- **Covariance matrices Σ:**
  - Symmetry: $\Sigma_j = \Sigma_j^T$ for all j
  - Positive definiteness: $x^T \Sigma_j x > 0$ for all non-zero $x \in \mathbb{R}^d$ and all j

3. EM Algorithm for GMM

The Expectation-Maximization algorithm iteratively estimates GMM parameters.

Q4. Complete the EM algorithm for GMMs:

Algorithm 5 EM Algorithm for GMMs

Require: Feature vectors $\{F_i\}_{i \in \mathcal{I}}$ where $F_i \in \mathbb{R}^d$, number of components K
Ensure: Optimal parameters $\theta^* = (\pi^*, \mu^*, \Sigma^*)$

```
Initialize parameters θ^(0) = (π^(0), μ^(0), Σ^(0))
Set t = 0
repeat
    E-step: Calculate posterior probabilities (responsibilities)
        For all i ∈ I and j ∈ {1, ..., K}:
        τ_i^j = [BLANK 1]                                   ▷ Fill in the blank
    M-step: Update parameters using responsibilities
        π_j^(t+1) = [BLANK 2]                               ▷ Fill in the blank
        μ_j^(t+1) = [BLANK 3]                               ▷ Fill in the blank
        Σ_j^(t+1) = (Σ_{i∈I} τ_i^j (F_i − μ_j^(t+1))(F_i − μ_j^(t+1))^T) / (Σ_{i∈I} τ_i^j)
    t ← t + 1
until convergence
return θ* = θ^(t)
```

- **BLANK 1:** $\tau_i^j = \frac{\pi_j^{(t)} \mathcal{N}(F_i | \mu_j^{(t)}, \Sigma_j^{(t)})}{\sum_{j'=1}^K \pi_{j'}^{(t)} \mathcal{N}(F_i | \mu_{j'}^{(t)}, \Sigma_{j'}^{(t)})}$

- **BLANK 2:** $\pi_j^{(t+1)} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \tau_i^j$
- **BLANK 3:** $\mu_j^{(t+1)} = \frac{\sum_{i \in \mathcal{I}} \tau_i^j F_i}{\sum_{i \in \mathcal{I}} \tau_i^j}$

4. **Cluster Assignment and Prediction**

Q5. Given a trained GMM with parameters $\theta = (\pi, \mu, \Sigma)$:

- Derive the formula for calculating the probability that a feature vector F_{new} belongs to cluster k
- Explain how to make a hard cluster assignment for each time series based on these probabilities

Probability calculation: Using Bayes' theorem, the probability that F_{new} belongs to cluster k is:

$$p(Z = k|F_{new}) = \frac{p(F_{new}|Z = k)p(Z = k)}{\sum_{j=1}^{K} p(F_{new}|Z = j)p(Z = j)}$$

Substituting the GMM parameters:

$$p(Z = k|F_{new}) = \frac{\pi_k \mathcal{N}(F_{new}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(F_{new}|\mu_j, \Sigma_j)}$$

Hard cluster assignment: To make a hard assignment, use the maximum a posteriori (MAP) rule:

$$\text{Cluster}(F_{new}) = \arg \max_{k \in \{1,\ldots,K\}} p(Z = k|F_{new})$$

This assigns each time series to the cluster with the highest posterior probability.

After performing clustering analysis, we have identified distinct groups of time series. The next step is to leverage this clustering information for improved forecasting. Two primary approaches will be considered:

Method 1: Training one Random Forest regressor per cluster of time series, allowing each cluster to have its specialized forecasting model.

Method 2: Using the cluster assignment as an additional feature in the Temporal Fusion Transformer, enabling the sequential neural network to incorporate cluster information in its predictions.

Exercise 4 (Regression Models for Time Series Forecasting)**.**

Having identified clusters of similar time series, our objective is now to predict future values for each time series.

- We use a backward window of size T_b for training, from time $t - T_b + 1$ to t
- We predict forward for a horizon of T_f time steps, from $t + 1$ to $t + T_f$

1. **Method 1: Cluster-Specific Regression Models**

We train separate regression models $\mathcal{M}_k$ for $k \in \{1, \ldots, K\}$, where each model specializes in forecasting time series belonging to cluster k.

Q1. For a specific node in a regression tree within one of the models $\mathcal{M}_k$:

- What objective function do we optimize to decide on which feature and threshold to split?
- What prediction value is assigned to samples in each child node after the split?

Objective function for splitting: We minimize the weighted sum of squared errors (SSE)

$$\text{Objective} = \min_{j,s} [\text{SSE}_{\text{left}}(j, s) + \text{SSE}_{\text{right}}(j, s)]$$

where:

- j is the feature to split on
- s is the threshold value
- $\text{SSE}_{\text{left}}(j, s) = \sum_{i:x_{ij} \leq s} (y_i - \bar{y}_{\text{left}})^2$
- $\text{SSE}_{\text{right}}(j, s) = \sum_{i:x_{ij} > s} (y_i - \bar{y}_{\text{right}})^2$

Prediction value in child nodes: After splitting, each child node is assigned the mean of all target values in that node:

- Left child: $\hat{y}_{\text{left}} = \frac{1}{n_{\text{left}}} \sum_{i:x_{ij} \leq s} y_i$
- Right child: $\hat{y}_{\text{right}} = \frac{1}{n_{\text{right}}} \sum_{i:x_{ij} > s} y_i$

Q2. Cite and explain at least two evaluation metrics suitable for assessing the performance of regression models in time series forecasting.

1. Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{|\mathcal{I}| \cdot T_f} \sum_{i \in \mathcal{I}} \sum_{t_f=1}^{T_f} |y_i^{t+t_f} - \hat{y}_i^{t+t_f}|$$

2. Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{I}| \cdot T_f} \sum_{i \in \mathcal{I}} \sum_{t_f=1}^{T_f} (y_i^{t+t_f} - \hat{y}_i^{t+t_f})^2}$$

3. Mean Absolute Percentage Error (MAPE):

$$\text{MAPE} = \frac{100}{|\mathcal{I}| \cdot T_f} \sum_{i \in \mathcal{I}} \sum_{t_f=1}^{T_f} \left| \frac{y_i^{t+t_f} - \hat{y}_i^{t+t_f}}{y_i^{t+t_f}} \right|$$

2. Method 2: Temporal Fusion Transformer with Cluster Information

We now incorporate cluster assignments into the Temporal Fusion Transformer (TFT) architecture. Figure 4 shows the complete TFT model structure.

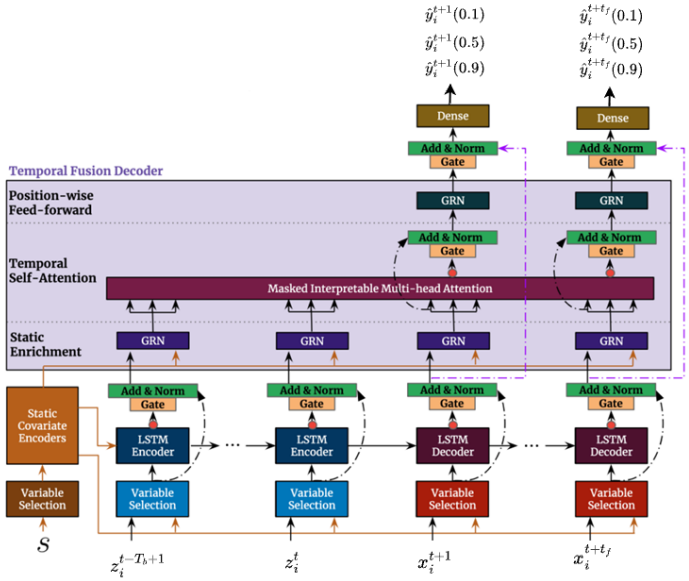

Figure 4: Temporal Fusion Transformer architecture

For a time series entity i at time t, the TFT accepts three types of inputs with the following notations:

- **Static attributes** $s_i \in \mathbb{R}^{d_s}$: Non-sequential features that remain constant over time, including entity-specific features like the cluster assignment
- **Time varying unknown** $(z_i^{t-T_b+1}, \ldots, z_i^t) \in \mathbb{R}^{T_b \times d_z}$: Sequential features available only up to the present time t
- **Time varying known** $(x_i^{t+1}, \ldots, x_i^{t+T_f}) \in \mathbb{R}^{T_f \times d_x}$: Sequential features known in advance for the future prediction period

Additionally, the target series consists of:

- **Past values**: $(y_i^{t-T_b+1}, \ldots, y_i^t)$ - historical data used for training
- **Future values to predict**: $(y_i^{t+1}, \ldots, y_i^{t+T_f})$ - the prediction horizon

As illustrated in Figure 5, the TFT outputs quantile predictions for $\mathcal{Q} = \{0.1, 0.5, 0.9\}$. For each time step $t + t_f$ where $t_f \in \{1, \ldots, T_f\}$, the model outputs predictions $\hat{y}_i^{t+t_f}(q)$ for each quantile $q \in \mathcal{Q}$.

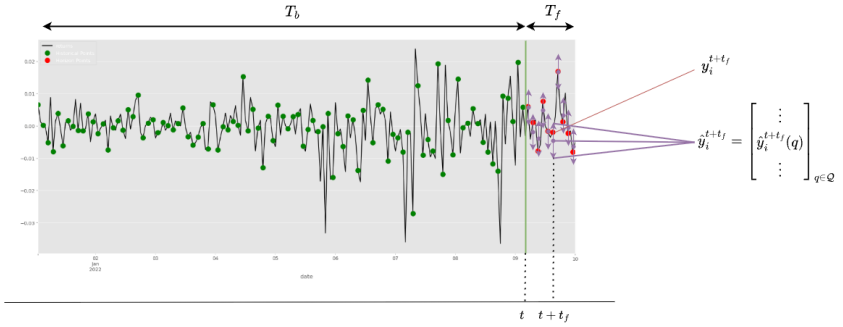

Figure 5: TFT quantile predictions

The quantile loss function is defined as:

$$\mathcal{L}(\mathcal{B}, \theta) = \sum_{i \in \mathcal{B}} \sum_{q \in \mathcal{Q}} \sum_{t_f=1}^{T_f} \frac{QL_q(y_i^{t+t_f}, \hat{y}_i^{t+t_f}(q))}{|\mathcal{B}| T_f}$$

where $\mathcal{B}$ is a batch of training data and:

$$QL_q(y, \hat{y}) = q(y - \hat{y})_+ + (1 - q)(\hat{y} - y)_+ = \max((q-1)(y - \hat{y}), q(y - \hat{y}))$$

The TFT provides two layers of interpretability: across time and across features. As shown in Figure 6, the Variable Selection Network (VSN) is responsible for interpretation across features.

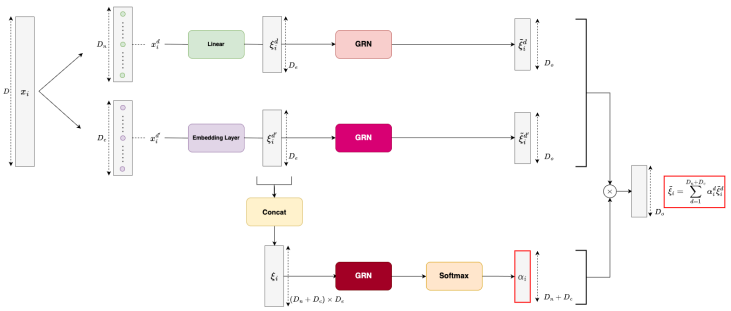

Figure 6: The VSN architecture

Q3. Based on the TFT architecture in Figure 4:

- Where in the TFT architecture is the cluster assignment (created in the previous exercise) incorporated?
- The VSN in Figure 4 outputs importance weights $\alpha_i \in \mathbb{R}^{d_s}$ for **Static attributes**. What is the meaning of these α parameters?
- What is the equivalent concept to these α weights in tree-based models?

- **1. Cluster assignment incorporation:** The cluster assignment is incorporated as part of the **Static attributes** $s_i \in \mathbb{R}^{d_s}$. These are non-sequential features that remain constant over time and are fed into the static variable selection component of the TFT.
- **2. Meaning of α parameters:** The $\alpha_i \in \mathbb{R}^{d_s}$ weights represent the relative importance or relevance of each static feature for the prediction task. They:
  - Act as learnable attention weights for feature selection
  - Determine how much each static attribute contributes to the final prediction
  - Enable the model to focus on the most relevant static features
  - Provide interpretability by showing which static features are most important
- **3. Equivalent concept in tree-based models:** In tree-based models, the equivalent concept is **Mean Decrease Impurity scores**:
  - Measured by how much each feature decreases node impurity (Gini importance)
  - Similar to α weights, they indicate which features contribute most to predictions.
  - Both provide interpretability by ranking feature relevance.

Q4. The TFT uses LSTM layers for sequential processing. As shown in figure 7 In an LSTM cell, we maintain:

- Cell state C^t: Preserves long-term memory
- Hidden state h^t: Represents short-term output

Given the previous states (h^{t-1}, C^{t-1}) and current input x^t, provide the complete LSTM update equations for:

- Forget gate f^t
- Input gate i^t
- Memory candidate $\tilde{C}^t$
- Output gate o^t
- Cell state update C^t
- Hidden state update h^t

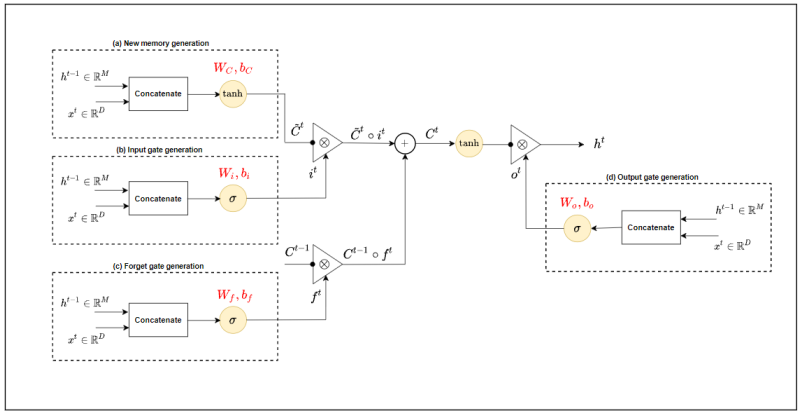

Figure 7: The LSTM layer

LSTM Update Equations:

- **Forget gate:**
$$f^t = \sigma(W_f \cdot [h^{t-1}, x^t] + b_f)$$
- **Input gate:**
$$i^t = \sigma(W_i \cdot [h^{t-1}, x^t] + b_i)$$
- **Memory candidate:**
$$\tilde{C}^t = \tanh(W_C \cdot [h^{t-1}, x^t] + b_C)$$
- **Output gate:**
$$o^t = \sigma(W_o \cdot [h^{t-1}, x^t] + b_o)$$

- **Cell state update:**

$$C^t = f^t \odot C^{t-1} + i^t \odot \tilde{C}^t$$

- **Hidden state update:**

$$h^t = o^t \odot \tanh(C^t)$$

Where:

- σ is the sigmoid activation function
- $\odot$ denotes element-wise multiplication
- $[h^{t-1}, x^t]$ is the concatenation of previous hidden state and current input
- W_f, W_i, W_C, W_o are weight matrices
- b_f, b_i, b_C, b_o are bias vectors

Q5. Consider the optimization of the quantile loss function. Describe an appropriate optimization algorithm to train the TFT model

- **Optimization Algorithm: Adam (Adaptive Moment Estimation)** Adam is particularly well-suited for training the TFT model because:
- **Key features:**
  - Combines advantages of AdaGrad and RMSProp
  - Maintains adaptive learning rates for each parameter
  - Uses first and second moment estimates of gradients
- **Update equations:**

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad \text{(first moment estimate)}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad \text{(second moment estimate)}$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \text{(bias-corrected first moment)}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad \text{(bias-corrected second moment)}$$
$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad \text{(parameter update)}$$

- **Typical hyperparameters:**
  - Learning rate: $\alpha = 0.001$
  - First moment decay: $\beta_1 = 0.9$
  - Second moment decay: $\beta_2 = 0.999$
  - Numerical stability: $\epsilon = 10^{-8}$
- **Additional training considerations:**
  - Gradient clipping to prevent exploding gradients in LSTM layers
  - Learning rate scheduling
  - Early stopping based on validation loss
  - Batch normalization or layer normalization for stable training