Systematic Trading Strategies with Machine Learning Algorithms

Mock Exam

June 9, 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: primary model development using tree-based methods with trend scanning, and meta-model implementation using Variable Selection Networks for trade filtering.

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: Creating a Primary Model using Tree-Based Models and Trend Scanning (60 marks)**
  - Exercise 1: Labeling Methodology using Trend Scanning (3 questions)
  - Exercise 2: Tree-Based Models for the Primary Model (5 questions)
  - Exercise 3: Cluster-Based Feature Importance Analysis (7 questions)
- **Section 2: Meta-model using Variable Selection Networks (40 marks)**
  - Exercise 1: Meta-labeling and Threshold-Based Trade Filtering (3 questions)
  - Exercise 2: Variable Selection Networks (7 questions)

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

Section 1: Creating a Primary Model using tree based models and Trend Scanning

Introduction

Financial markets exhibit complex temporal patterns that can be systematically identified and exploited through machine learning techniques. Effective trading strategies require both accurate signal generation and intelligent trade filtering to maximize profitability while minimizing risk.

In this section, we explore a comprehensive approach to systematic trading that consists of three major components:

1. **Labeling Methodology**: We begin by implementing a trend detection framework that identifies directional movements in financial time series using trend scanning techniques.
2. **Tree-Based Models for the Primary Model**: We develop supervised learning models using Random Forest algorithms to map features to trend labels.
3. **Cluster-Based Feature Importance Analysis**: We implement advanced feature importance analysis that groups correlated features and provides robust importance estimates.

The combination of these three components creates a comprehensive framework for systematic trading strategy development.

Exercise 1 (Labeling Methodology)**.**

Trend scanning is a data-driven labeling method that identifies significant directional movements in financial time series by fitting linear regression models over multiple forward-looking horizons. Unlike fixed-horizon approaches, trend scanning adapts to market dynamics by selecting the optimal look-ahead period that exhibits the strongest statistical trend.

In this exercise, we implement the trend scanning methodology as covered in the course. The approach has two main hyperparameters that define the search space for optimal trend detection:

- $H_{\min}$: Minimum look-ahead horizon
- $H_{\max}$: Maximum look-ahead horizon

We want to label time steps from 1 to T. For a specific time step t, we evaluate all horizons $H \in \{H_{\min}, H_{\min}+1, \ldots, H_{\max}\}$ to find the one that maximizes the statistical significance of the observed trend, as illustrated in Figure 1.

The trend scanning approach works by fitting a linear regression model for each possible horizon H and selecting the horizon that produces the most statistically significant trend. Let $\{P_s\}_{s=1}^{T}$ be a sequence of asset prices.

For a specific time step t and horizon H, we have prices from P_t to P_{t+H}. We fit the following linear model to the forward-looking price series:

$$P_{t+h} = \beta_0 + \beta_1 h + \varepsilon_{t+h}, \quad h = 0, 1, \ldots, H$$

where β_1 represents the trend coefficient (slope) and ε_{t+h} is the error term. The statistical significance of the trend is measured by the t-statistic:

$$t_{\hat{\beta}_1} = \frac{\hat{\beta}_1}{\hat{\sigma}_{\hat{\beta}_1}}$$

where $\hat{\beta}_1$ is the estimated slope coefficient and $\hat{\sigma}_{\hat{\beta}_1}$ is its standard error.

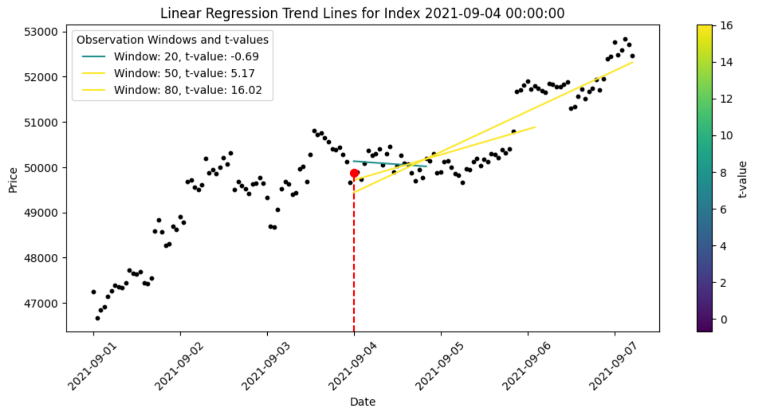

Figure 1: Trend scanning example: evaluating multiple horizons to find optimal trend

As illustrated in the figure above, for each time step, we compute t-statistics across different horizons and select the one that indicates the most significant trend.

Q1. Based on the trend scanning methodology:

- For each horizon H, we compute a t-statistic $t_{\hat{\beta}_1}(H)$. How do we select the optimal horizon H^* among all the candidates $H \in \{H_{\min}, \ldots, H_{\max}\}$?
- Once we have identified the optimal horizon H^*, how do we generate the final trend label $y_t \in \{-1, +1\}$ for this time step?

- **Optimal horizon selection:** The optimal horizon H^* is selected by choosing the horizon that maximizes the absolute value of the t-statistic:

$$H^* = \arg \max_{H \in \{H_{\min}, \ldots, H_{\max}\}} |t_{\hat{\beta}_1}(H)|$$

  This ensures we select the horizon that exhibits the most statistically significant trend, regardless of direction.
- **Label generation:** Once we have identified the optimal horizon H^*, we generate the trend label based on the sign of the slope coefficient at that horizon:

$$y_t = \begin{cases} +1 & \text{if } \hat{\beta}_1(H^*) > 0 \text{ (upward trend)} \\ -1 & \text{if } \hat{\beta}_1(H^*) \leq 0 \text{ (downward trend)} \end{cases}$$

  The label $+1$ indicates a significant upward trend, while -1 indicates a significant downward trend.

Q2. Complete the trend scanning algorithm for labeling a single time step t:

Algorithm 1 Trend Scanning for Single Time Step

```
Require: Price series $\{P_s\}_{s=1}^T$, time step $t$, $H_{\min}$, $H_{\max}$
Ensure: Trend label $y_t$ and optimal horizon $H_t^*$
 1: Initialize $t_{\max} = 0$, $H_t^* = H_{\min}$
 2: for $H = H_{\min}$ to $H_{\max}$ do
 3:     [BLANK 1]                                        ▷ Fill in the blank
 4:     [BLANK 2]                                        ▷ Fill in the blank
 5:     Calculate t-statistic: $t_{\hat{\beta}_1} = \frac{\hat{\beta}_1}{\hat{\sigma}_{\hat{\beta}_1}}$
 6:     if $|t_{\hat{\beta}_1}| > t_{\max}$ then
 7:         $t_{\max}$ = [BLANK 3]                       ▷ Fill in the blank
 8:         $H_t^* = H$
 9:     end if
10: end for
11: Generate label: $y_t$ = [BLANK 4]                    ▷ Fill in the blank
12: return $y_t$, $H_t^*$
```

- **BLANK 1:** Create regression data: $\mathbf{h} = [0, 1, 2, \ldots, H]^T$ and $\mathbf{P} = [P_t, P_{t+1}, P_{t+2}, \ldots, P_{t+H}]^T$

- **BLANK 2:** Fit linear regression: $\hat{\beta}_1, \hat{\sigma}_{\hat{\beta}_1} = \text{LinearRegression}(\mathbf{h}, \mathbf{P})$
- **BLANK 3:** $t_{\max} = |t_{\hat{\beta}_1}|$
- **BLANK 4:** $y_t = \text{sign}(\hat{\beta}_1(H_t^*))$ where $\text{sign}(x) = +1$ if $x > 0$, else -1

With this algorithm, we have generated y_t associated with time step t.

The trend scanning technique can also be used backward-looking to generate features at time step t.

Q3. What modifications should be made to the aforementioned algorithm to generate a new feature that captures the strength of historical trends ending at time t?

To generate a backward-looking feature that captures historical trend strength ending at time t, we make the following modifications: **Modified Algorithm Structure:**

- **Time direction:** Instead of looking forward from time t, we look backward. For horizon H, we use prices from P_{t-H} to P_t.
- **Regression setup:** For each horizon H, create:
  - $\mathbf{h} = [0, 1, 2, \ldots, H]^T$ (time indices)
  - $\mathbf{P} = [P_{t-H}, P_{t-H+1}, \ldots, P_{t-1}, P_t]^T$ (historical prices)
- **Feature generation:** Instead of generating a binary label $y_t \in \{-1, +1\}$, we generate a continuous feature:

$$\text{TrendStrength}_t = t_{\hat{\beta}_1}(H^*) \cdot \text{sign}(\hat{\beta}_1(H^*))$$

  This preserves both the magnitude (statistical significance) and direction of the trend.

This backward-looking feature can then be used as an input to the tree-based models for predicting future trends.

This process of labeling and feature generation (along with many other indicators) are used over the entire dataset to generate training data $\{(\mathbf{X}_t, y_t)\}$ for $t \in \{1, \ldots, T\}$. The next step is to train tree-based models to map the features to the targets.

Exercise 2 (Tree-Based Models for the Primary Model)**.**

We will fit a Random Forest algorithm on the training data $\{(\mathbf{X}_t, y_t)\}_{t=1}^T$, where $y_t \in \{-1, +1\}$ represents the trend labels generated from the trend scanning methodology. The Random Forest consists of multiple decision trees, each trained on a bootstrap sample of the data.

Q1. Cite and briefly describe three key hyperparameters of a decision tree algorithm that should be tuned to optimize performance.

Three key hyperparameters for decision tree optimization:

- **1. Maximum Depth (max_depth):** Controls the maximum number of levels in the tree. Deeper trees can model more complex patterns but may overfit the training data; limiting depth helps reduce overfitting.
- **2. Minimum Samples per Leaf (min_samples_leaf):** Specifies the minimum number of samples required to be at a leaf node. Increasing this value can reduce model complexity and overfitting by enforcing broader generalizations at the leaves.
- **3. Impurity Criterion (criterion):** Defines the function used to measure the quality of a split, such as Gini impurity or entropy.

Recall that for binary classification with a set S containing classes $\{-1, +1\}$, the entropy is defined as:

$$H(S) = -p_{+1} \log_2(p_{+1}) - p_{-1} \log_2(p_{-1})$$

where $p_{+1} = \frac{|\{s \in S : \text{class}(s) = +1\}|}{|S|}$ and $p_{-1} = \frac{|\{s \in S : \text{class}(s) = -1\}|}{|S|}$ are the proportions of samples in S belonging to classes $+1$ and -1 respectively.

Consider a parent node containing 100 samples with equal distribution (50 samples with label $+1$ and 50 samples with label -1). A categorical feature splits this node based on feature values 0 and 1, resulting in:

Node	Feature Value	Label $+1$	Label -1
Parent	-	50	50
Left Child	0	50	0
Right Child	1	0	50

Q2. Calculate the Information Gain (IG) for the split described in the example above using entropy as the impurity measure. Calculate the entropy before the split, the weighted entropy after the split, and the information gain.

- **Entropy before split (Parent node):** $p_{+1} = \frac{50}{100} = 0.5$ and $p_{-1} = \frac{50}{100} = 0.5$ $H(\text{Parent}) = 1$
- **Entropy after split:** Left child: $p_{+1} = \frac{50}{50} = 1$, $p_{-1} = \frac{0}{50} = 0$ $H(\text{Left}) = -1\log_2(1) - 0\log_2(0) = 0$ (using convention $0\log_2(0) = 0$) Right child: $p_{+1} = \frac{0}{50} = 0$, $p_{-1} = \frac{50}{50} = 1$ $H(\text{Right}) = -0\log_2(0) - 1\log_2(1) = 0$
- **Weighted entropy after split:** $H_{\text{weighted}} = \frac{50}{100} \times 0 + \frac{50}{100} \times 0 = 0$
- **Information Gain:** $IG = H(\text{Parent}) - H_{\text{weighted}} = 1 - 0 = 1$

Q3. Complete the decision tree learning algorithm for our trend classification problem:

Algorithm 2 Decision Tree Learning Algorithm

```
Require: Training data $\{(\mathbf{X}_t, y_t)\}_{t=1}^{T}$ where $y_t \in \{-1, +1\}$, stopping criteria
Ensure: Decision tree $T$
 1: Initialize tree with single root node containing all data
 2: while nodes can be split and stopping criteria not met do
 3:     for each leaf node with region $\mathcal{R}$ do
 4:         Find $(j^*, \tau^*)$ that maximizes: [BLANK 1]                 ▷ Fill in the blank
 5:         $IG(j, \tau) = I(\mathcal{R}) - \frac{|\mathcal{R}_L|}{|\mathcal{R}|} I(\mathcal{R}_L) - \frac{|\mathcal{R}_R|}{|\mathcal{R}|} I(\mathcal{R}_R)$
 6:         Where $\mathcal{R}_L = \{\mathbf{X}_t \in \mathcal{R} : \mathbf{X}_{t,j} \le \tau\}$ and $\mathcal{R}_R = \{\mathbf{X}_t \in \mathcal{R} : \mathbf{X}_{t,j} > \tau\}$
 7:         Create left child node with samples: [BLANK 2]  ▷ Fill in the blank
 8:         Create right child node with samples: [BLANK 3] ▷ Fill in the blank
 9:     end for
10: end while
11: Assign prediction to each leaf node: [BLANK 4]            ▷ Fill in the blank
12: return $T$
```

- **BLANK 1:** $IG(j, \tau)$ where j is feature index and τ is threshold value
- **BLANK 2:** $\{(\mathbf{X}_t, y_t) : \mathbf{X}_t \in \mathcal{R}_L\}$
- **BLANK 3:** $\{(\mathbf{X}_t, y_t) : \mathbf{X}_t \in \mathcal{R}_R\}$
- **BLANK 4:** $\hat{y} = \arg\max_{c \in \{-1,+1\}} \sum_{(\mathbf{X}_t, y_t) \in \text{leaf}} \mathbb{I}(y_t = c)$ (majority class)

We will use Random Forest to avoid overfitting, as illustrated in Figure 2.

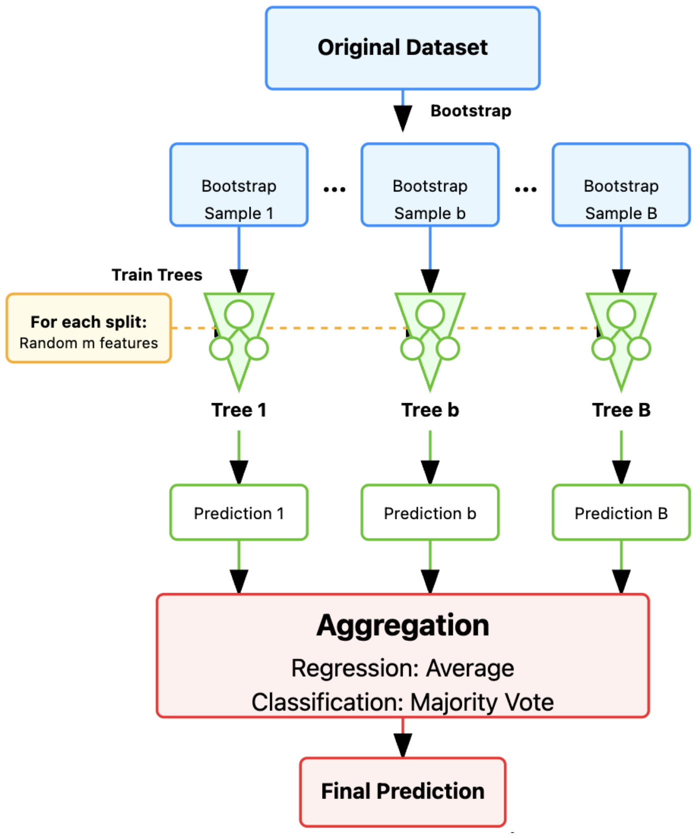

Figure 2: Random Forest algorithm overview

Q4. What are the two main sources of randomness introduced by Random Forest to avoid overfitting compared to a single decision tree?

The two main sources of randomness in Random Forest are:

- **Bootstrap Sampling (Bagging):** Each tree is trained on a different bootstrap sample of the training data, created by sampling with replacement. This ensures that each tree sees a slightly different version of the dataset, reducing overfitting to specific data points.
- **Random Feature Selection:** At each split in each tree, only a random subset of features (typically $\sqrt{D}$ features out of D total features) is considered for the best split. This prevents trees from always using the same strong features and increases diversity among trees.

Q5. We would like to evaluate the Random Forest algorithm. Cite and briefly describe two evaluation metrics that don't depend on a specific threshold to turn the probability of positive label into hard predictions (-1/+1).

Two threshold-independent evaluation metrics:

- **1. Area Under the ROC Curve (AUC-ROC):** Measures the area under the Receiver Operating Characteristic curve, which plots True Positive Rate vs. False Positive Rate across all possible thresholds. AUC ranges from 0 to 1, where 0.5 indicates random performance and 1.0 indicates perfect classification.
- **2. Log Loss (Cross-Entropy Loss):** Evaluates the quality of probabilistic predictions by penalizing false classifications with a logarithmic loss function. It captures how far predicted probabilities are from the actual labels, with lower values indicating better calibrated and more confident models. The formula is:

$$\text{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

where $y_i \in \{0,1\}$ is the true label and p_i is the predicted probability of the positive class.

Exercise 3 (Cluster-Based Feature Importance Analysis)**.**

Traditional feature importance methods like Mean Decrease Impurity (MDI) or Permutation Feature Importance (PFI) can be misleading when features are correlated or noisy. To address this limitation, we develop a cluster-based approach that groups similar features and provides more robust importance estimates.

Let $\mathbf{X} \in \mathbb{R}^{T\times D}$ be our feature matrix where T is the number of time steps and D is the number of features. Each column $\mathbf{X}_{:,j}$ represents feature j across all time steps.

1. **Step 1: Initial Feature Importance with MDI**

We start by computing the Mean Decrease Impurity (MDI) for each feature using our trained Random Forest. Let MDI_j denote the importance score for feature j, where:

$$\sum_{j=1}^{D} \text{MDI}_j = 1 \quad \text{and} \quad \text{MDI}_j \geq 0 \quad \forall j$$

However, MDI has known limitations that we must address.

Q1. Explain why a noisy feature can have $\text{MDI}_j > 0$ even if it doesn't bring any information to the prediction task.

A noisy feature can have $\text{MDI}_j > 0$ due to the following reasons:

- **In-sample evaluation:** MDI is computed on the same data used for training, so it captures reductions in training impurity rather than true predictive power, making it susceptible to overfitting artifacts.
- **Overfitting bias:** Decision trees can overfit to noise, especially in deep trees. The algorithm will find splits that reduce impurity on the training set even if these splits are based on random fluctuations rather than true signal.
- **High cardinality bias:** Features with many unique values (like continuous noisy features) have more opportunities to create splits that appear to reduce impurity, giving them artificially high MDI scores.

2. **Step 2: Out-of-Sample Feature Importance**

To mitigate MDI's bias toward noisy features, we employ Permutation Feature Importance (PFI), which measures the decrease in model performance when a feature's values are randomly shuffled.

Let m be our trained model, $\mathcal{D} = \{(\mathbf{X}_i, y_i)\}_{i=1}^{T}$ be our validation dataset, and $s(\mathcal{D})$ be a performance metric (e.g., accuracy). For feature j, we create multiple corrupted datasets $\tilde{\mathcal{D}}_{k,j}$ for $k \in \{1,\ldots,K\}$ by randomly permuting the j-th feature column across different random seeds.

Q2. Complete the Permutation Feature Importance algorithm:

Algorithm 3 Permutation Feature Importance (PFI)

```
Require: Fitted model $m$, validation data $\mathcal{D}$, repetitions $K$
Ensure: Feature importance scores $\{PFI_j\}_{j=1}^{D}$ with standard deviations $\{\sigma_j\}_{j=1}^{D}$
 1: Compute reference score $s_0 = s(m, \mathcal{D})$
 2: for each feature $j \in \{1,\ldots,D\}$ do
 3:     Initialize array $scores_j$ of length $K$
 4:     for each repetition $k \in \{1,\ldots,K\}$ do
 5:         [BLANK 1]                                        ▷ Fill in the blank
 6:         Compute score $s_{k,j} = s(m, \tilde{\mathcal{D}}_{k,j})$
 7:         Store in array: $scores_j[k]$ = [BLANK 2]          ▷ Fill in the blank
 8:     end for
 9:     Compute $PFI_j = \text{mean}(scores_j)$ and $\sigma_j = \text{std}(scores_j)$
10: end for
11: return $\{PFI_j\}_{j=1}^{D}$ and $\{\sigma_j\}_{j=1}^{D}$
```

- **BLANK 1:** Create corrupted dataset $\tilde{\mathcal{D}}_{k,j}$ by randomly permuting feature j in $\mathcal{D}$
- **BLANK 2:** $s_0 - s_{k,j}$ (performance drop due to permutation)

3. **Step 3: Addressing Feature Correlation**

In our trend scanning framework, many generated features are highly correlated (e.g., different moving averages, overlapping technical indicators). This correlation creates challenges for feature importance analysis.

Let $\mathbf{C} \in \mathbb{R}^{D\times D}$ be the Spearman correlation matrix of our features, where $C_{i,j}$ represents the correlation between features i and j.

Q3. Explain how feature correlation affects permutation feature importance analysis through the substitution effect. When one important feature is permuted, how can correlated features mask the performance drop? Why does this motivate cluster-based feature importance analysis?

Feature correlation creates the **substitution effect** in permutation feature importance:

i. **Why the substitution effect causes underestimated importance:**
   - **Masking effect:** When an important feature is permuted (corrupted), highly correlated features can substitute for it and maintain model performance. This happens because correlated features contain similar information and can compensate for the loss of the permuted feature.
   - **Underestimated importance:** The substitution effect leads to underestimated importance scores for features that are part of correlated groups, as the performance drop is smaller than it would be if all correlated features were permuted simultaneously.

ii. **Motivation for cluster-based analysis:**
   - Features within the same cluster share similar information and should be analyzed together.
   - Permuting entire clusters provides more accurate importance estimates by eliminating substitution effects.
   - It identifies redundant features that can be removed without information loss.
   - It provides more interpretable results by grouping related features.

4. **Step 4: Distance Matrix Construction**

To cluster correlated features, we transform the correlation matrix into a distance matrix. We define the distance between features i and j as:

$$d_{i,j} = 1 - |C_{i,j}|$$

This ensures that highly correlated features (positive or negative correlation) are close in distance space, while uncorrelated features are far apart.

5. **Step 5: Dimensionality Reduction and K-means Clustering**

We apply Principal Component Analysis (PCA) to the distance matrix to reduce dimensionality. Let $\mathbf{Z} \in \mathbb{R}^{D\times d}$ be the PCA-transformed data where $d \ll D$ is chosen to retain at least 95% of the variance.

Next, we determine the optimal number of clusters using silhouette analysis.

For each data point i in cluster C_k, the silhouette coefficient is:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where $a(i)$ is the average intra-cluster distance, and $b(i)$ is the average distance to the nearest different cluster.

Q4. Complete the algorithm to find the optimal number of clusters:

Algorithm 4 Optimal Cluster Selection using Silhouette Score

```
Require: PCA-reduced data $\mathbf{Z} \in \mathbb{R}^{D\times d}$, maximum clusters $K_{\max}$
Ensure: Optimal number of clusters $K^*$
1: Initialize $silhouette\_scores = []$
2: for $K = 2$ to $K_{\max}$ do
3:     [BLANK 1]                                  ▷ Fill in the blank
4:     [BLANK 2]                                  ▷ Fill in the blank
5:     Append $score$ to $silhouette\_scores$
6: end for
7: $K^* =$ [BLANK 3]                              ▷ Fill in the blank
8: return $K^*$
```

- **BLANK 1:** Apply K-means clustering on the PCA-reduced data $\mathbf{Z}$ using the current value of K: `clusters = KMeans(n_clusters=K).fit(Z)`.
- **BLANK 2:** Compute the silhouette score for the clustering result using: `score = silhouette_score(Z, clusters.labels_)`.

- **BLANK 3:** Set K^* to the number of clusters corresponding to the highest silhouette score using: $K^* = \arg\max_i \texttt{silhouette\_scores}[i] + 2$, since the list index starts at $K = 2$.

6. **Step 6: Gaussian Mixture Model for Soft Clustering**

While K-means provides hard cluster assignments, we use Gaussian Mixture Models (GMM) to obtain probabilistic cluster memberships. This allows features to belong to multiple clusters with different probabilities.

Let $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ be the GMM parameters where:

- $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$ are mixture weights with $\sum_{k=1}^{K} \pi_k = 1$
- $\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K)$ are component means where $\boldsymbol{\mu}_k \in \mathbb{R}^d$
- $\boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K)$ are covariance matrices where $\boldsymbol{\Sigma}_k \in \mathbb{R}^{d \times d}$

Q5. For a GMM with K components modeling d-dimensional data, calculate the total number of parameters. Consider:

- Mixture weights: $\pi_1, \ldots, \pi_K$.
- Component means: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$.
- Full covariance matrices: $\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K$. (symmetric positive definite)

- **Total number of parameters without applying any constraints:**
  - Mixture weights: K parameters
  - Component means: $K \times d$ parameters
  - Covariance matrices: $K \times d^2$ parameters
  - **Total:**
$$K + Kd + Kd^2 = K(1 + d + d^2)$$
- **You may also answer by reducing the number of parameters by taking into account the following constraints:**
  - The mixture weights must sum to 1: this removes 1 degree of freedom, giving $K - 1$ free parameters.
  - Each covariance matrix is symmetric, so it has only $\frac{d(d+1)}{2}$ unique entries rather than d^2.
  - **Total number of parameters with constraints:**
    - Mixture weights: $K - 1$
    - Component means: $K \times d$
    - Covariance matrices: $K \times \frac{d(d+1)}{2}$
    - **Total:**
$$(K-1) + Kd + K \cdot \frac{d(d+1)}{2}$$

The GMM is trained using the Expectation-Maximization (EM) algorithm, which alternates between computing posterior probabilities (E-step) and updating parameters (M-step).

Q6. Complete the EM algorithm for GMM parameter estimation. The posterior probabilities (responsibilities) are given by:

$$\tau_{i,k} = \frac{\pi_k \mathcal{N}(\mathbf{z}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{z}_i | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

where $\mathbf{z}_i$ is the PCA representation of feature i.

Algorithm 5 EM Algorithm for GMM

```
Require: PCA data $\{\mathbf{z}_i\}_{i=1}^{D}$, number of components $K$
Ensure: Parameters $\boldsymbol{\theta} = (\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$
 1: Initialize parameters $\boldsymbol{\theta}^{(0)}$
 2: $t \leftarrow 0$
 3: while not converged do
 4:     E-step: Compute responsibilities $\tau_{i,k}^{(t)}$ for all $i \in \{1, \ldots, D\}$, $k \in \{1, \ldots, K\}$
 5:     M-step: Update parameters:
 6:         $\pi_k^{(t+1)}$ = [BLANK 1]                    ▷ Fill in the blank
 7:         $\boldsymbol{\mu}_k^{(t+1)}$ = [BLANK 2]                    ▷ Fill in the blank
 8:         $\boldsymbol{\Sigma}_k^{(t+1)}$ = [BLANK 3]                    ▷ Fill in the blank
 9:     $t \leftarrow t + 1$
10: end while
11: return $\boldsymbol{\theta}^{(t)}$
```

- **BLANK 1:** $\pi_k^{(t+1)} = \frac{1}{D} \sum_{i=1}^{D} \tau_{i,k}^{(t)}$
- **BLANK 2:** $\boldsymbol{\mu}_k^{(t+1)} = \frac{\sum_{i=1}^{D} \tau_{i,k}^{(t)} \mathbf{z}_i}{\sum_{i=1}^{D} \tau_{i,k}^{(t)}}$
- **BLANK 3:** $\boldsymbol{\Sigma}_k^{(t+1)} = \frac{\sum_{i=1}^{D} \tau_{i,k}^{(t)} (\mathbf{z}_i - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{z}_i - \boldsymbol{\mu}_k^{(t+1)})^T}{\sum_{i=1}^{D} \tau_{i,k}^{(t)}}$

7. **Step 7: Hard Cluster Assignment**

After training the GMM, we obtain a probability matrix $\mathbf{P} \in \mathbb{R}^{D \times K}$ where $P_{i,k}$ represents the probability that feature i belongs to cluster k.

Q7. Given the soft probability matrix $\mathbf{P}$, derive the hard cluster assignment for each feature.

The hard cluster assignment for each feature is obtained by selecting the cluster with the highest posterior probability:

$$c_i = \arg \max_{k \in \{1,\dots,K\}} P_{i,k}, \quad \text{for all } i \in \{1,\dots,D\}$$

Section 2: Meta-model using Variable Selection Networks

Introduction

While the primary Random Forest model from Section 1 successfully generates trading signals based on trend scanning labels, not every signal translates into profitable trades. Market conditions, volatility regimes, and structural breaks can significantly impact the reliability of these predictions. To enhance the overall trading strategy performance, we implement a meta-modeling framework that learns to identify when the primary model's signals are most likely to succeed.

This section introduces a comprehensive approach to trade filtering through meta-learning, consisting of three critical components:

1. **Meta-labeling Framework**: We develop a systematic approach to generate binary labels that indicate trade profitability using barrier-based exit strategies adapted to market volatility.

2. **Feature Engineering for Meta-learning**: We construct a distinct feature set that captures market microstructure, regime characteristics, and environmental conditions rather than price patterns used by the primary model.

3. **Variable Selection Networks**: We implement advanced neural network architectures specifically designed for feature selection and binary classification in high-dimensional financial data.

The meta-model operates as an intelligent filter that evaluates market conditions and predicts the probability that the primary model's signal will generate alpha. This probabilistic framework allows for flexible threshold-based trade filtering, enabling systematic control over the precision-recall trade-off.

Exercise 4 (Meta-labeling and Threshold-Based Trade Filtering)**.**

The primary Random Forest model generates trend signals $s_t \in \{-1, +1\}$ based on the trend scanning methodology. However, these signals exhibit varying success rates across different market regimes. Our meta-model M_{meta} learns to predict the probability $p_t = P(\text{Profitable Trade}|\text{Market Conditions}_t)$ that a trade initiated at time t will be profitable.

Architecture Overview:

- The **Primary Model** processes price-based features to generate directional signals $s_t \in \{-1, +1\}$.
- The **Meta-model** analyzes market environment features to predict signal reliability p_t.
- The **Final Decision** trades only if $p_t > \tau$ for threshold τ.

The meta-model framework, illustrated in Figure 3, processes a distinct feature space to avoid information leakage and redundancy with the primary model.

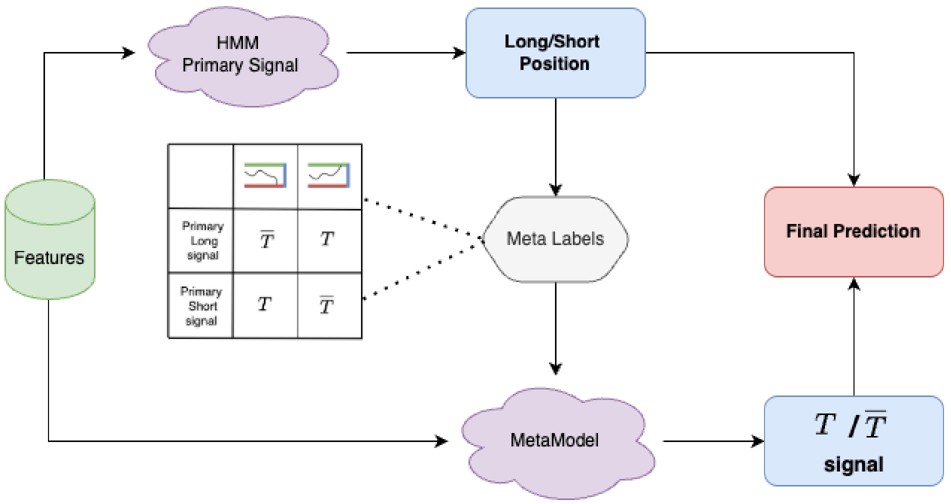

Figure 3: Metamodel Framework

1. **Meta-label Generation using Adaptive Barriers**

To train our meta-model, we require binary labels $y_t^{\text{meta}} \in \{0, 1\}$ indicating trade profitability. We employ an adaptive triple-barrier method that dynamically adjusts barrier widths based on realized volatility.

For a trade initiated at time t with signal s_t, we define:

- Entry price: P_t
- Volatility estimate: $\hat{\sigma}_t$ (computed from historical price returns)
- Upper barrier: $P_t \times (1 + \alpha \times \hat{\sigma}_t)$
- Lower barrier: $P_t \times (1 - \alpha \times \hat{\sigma}_t)$

- Time barrier: $t + H_{\max}$ (maximum holding period)

The volatility scaling factor α ensures that profit targets and stop-losses adapt to market conditions.

Q1. Consider the adaptive barrier method for meta-label generation at time t.

Algorithm 6 Adaptive Meta-label Generation for Single Time Step

```
Require: Price series $\{P_s\}_{s=t}^{t+H_{\max}}$, signal $s_t$, volatility estimate $\hat{\sigma}_t$, scaling factor
    $\alpha$, max holding period $H_{\max}$
Ensure: Meta-label $y_t^{\text{meta}}$
 1: Set entry price: $P_{\text{entry}} = P_t$
 2: Set barriers: $P_{\text{upper}} =$ [BLANK 1], $P_{\text{lower}} =$ [BLANK 2]
 3: Initialize $exit\_time = t + H_{\max}$, $exit\_price = P_{t+H_{\max}}$
 4: for $h = 1$ to $H_{\max}$ do
 5:     if $P_{t+h} \geq P_{\text{upper}}$ then
 6:         $exit\_time = t + h$, $exit\_price = P_{t+h}$
 7:         Break
 8:     else if $P_{t+h} \leq P_{\text{lower}}$ then
 9:         $exit\_time = t + h$, $exit\_price = P_{t+h}$
10:         Break
11:     end if
12: end for
13: Compute trade return: $R_t = s_t \times \frac{exit\_price - P_{\text{entry}}}{P_{\text{entry}}}$
14: Generate meta-label: $y_t^{\text{meta}} =$ [BLANK 3]                ▷ Fill in the blank
15: return $y_t^{\text{meta}}$
```

- **BLANK 1:** $P_{\text{upper}} = P_t \times (1 + \alpha \times \hat{\sigma}_t)$
- **BLANK 2:** $P_{\text{lower}} = P_t \times (1 - \alpha \times \hat{\sigma}_t)$
- **BLANK 3:** $y_t^{\text{meta}} = \begin{cases} 1 & \text{if } R_t > 0 \\ 0 & \text{if } R_t \leq 0 \end{cases}$ (binary label based on trade profitability)

2. **Meta-model Feature Engineering**

The meta-model requires features that capture market environment and trading conditions, distinct from the price-based technical indicators used by the primary model. We construct features across several categories: volatility regime indicators, latent states from HMMs, and primary model confidence measures (Random Forest prediction probabilities, feature importance stability). Let $\mathbf{X}_t \in \mathbb{R}^{D_{\text{meta}}}$ denote the meta-feature vector at time t.

3. **Variable Selection Networks for Binary Classification**

We implement a Variable Selection Network (VSN) that simultaneously performs feature selection and binary classification. The VSN architecture consists of a feature selection layer that learns attention weights for each feature, and classification layers that process selected features to output probability $p_t = P(y_t^{\text{meta}} = 1|\mathbf{X}_t)$.

The VSN produces probabilistic outputs that must be converted to binary decisions using threshold τ:

$$\hat{y}_t^{\text{meta}}(\tau) = \begin{cases} 1 & \text{if } p_t > \tau \\ 0 & \text{if } p_t \leq \tau \end{cases}$$

The final trading decision is:

$$\text{Final Signal}_t(\tau) = \begin{cases} s_t & \text{if } p_t > \tau \\ 0 & \text{if } p_t \leq \tau \end{cases}$$

Different threshold values τ control the trade-off between precision and recall, effectively filtering different proportions of the primary model's signals.

We evaluate three trading strategies on a test set of 2000 trading opportunities where the primary model generated signals. The ground truth shows 1200 profitable trades and 800 unprofitable trades. Consider the following confusion matrices:

- **Strategy A (Primary Model Only):** Always trade when primary model signals.

	Predicted Trade	Predicted No Trade
Actual Profitable	1200	0
Actual Unprofitable	800	0

- **Strategy B (Primary + Meta-model, τ_1):**

	Predicted Trade	Predicted No Trade
Actual Profitable	1080	120
Actual Unprofitable	600	200

- **Strategy C (Primary + Meta-model, τ_2):**

	Predicted Trade	Predicted No Trade
Actual Profitable	950	250
Actual Unprofitable	350	450

Q2. Calculate the precision and recall for each strategy.

- **Strategy A (Primary Model Only):**
  - Precision $= \frac{TP}{TP+FP} = \frac{1200}{1200+800} = \frac{1200}{2000} = 0.60$
  - Recall $= \frac{TP}{TP+FN} = \frac{1200}{1200+0} = \frac{1200}{1200} = 1.00$

- **Strategy B (Primary + Meta-model, τ_1):**
  - Precision $= \frac{TP}{TP+FP} = \frac{1080}{1080+600} = \frac{1080}{1680} = 0.643$
  - Recall $= \frac{TP}{TP+FN} = \frac{1080}{1080+120} = \frac{1080}{1200} = 0.90$

- **Strategy C (Primary + Meta-model, τ_2):**
  - Precision $= \frac{TP}{TP+FP} = \frac{950}{950+350} = \frac{950}{1300} = 0.731$
  - Recall $= \frac{TP}{TP+FN} = \frac{950}{950+250} = \frac{950}{1200} = 0.792$

Q3. Using the precision and recall values calculated in Q2:

- Compute the F1 score for each strategy
- Determine which meta-model threshold (τ_1 or τ_2) provides the optimal balance between precision and recall.

- **F1 Score Calculations:** The F1 score is calculated as: $F1 = 2 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$
  - **Strategy A:** $F1_A = 2 \times \frac{0.60\times1.00}{0.60+1.00} = 2 \times \frac{0.60}{1.60} = 0.75$
  - **Strategy B:** $F1_B = 2 \times \frac{0.643\times0.90}{0.643+0.90} = 2 \times \frac{0.579}{1.543} = 0.750$
  - **Strategy C:** $F1_C = 2 \times \frac{0.731\times0.792}{0.731+0.792} = 2 \times \frac{0.579}{1.523} = 0.760$
- **Optimal Threshold:** Strategy C with threshold τ_2 provides the optimal balance between precision and recall, achieving the highest F1 score of 0.760. This threshold offers better precision (0.731) while maintaining reasonable recall (0.792), making it more effective at filtering unprofitable trades while retaining most profitable opportunities.

Exercise 5 (Variable Selection Networks)**.**

We now implement a Variable Selection Network (VSN) to predict the meta-labels $y_t^{\text{meta}} \in \{0,1\}$ using the meta-features $\mathbf{X}_t \in \mathbb{R}^D$ constructed in Exercise 4. Our training dataset consists of N_T samples $\{(\mathbf{X}_i, y_i^{\text{meta}})\}_{i=1}^{N_T}$ where each feature vector $\mathbf{X}_i$ contains both numerical and categorical variables.

The feature space is partitioned as $D = D_n + D_c$, where:

- D_n numerical features: x_i^d for $d \in \{1,\ldots,D_n\}$
- D_c categorical features: $x_i^{d'}$ for $d' \in \{D_n+1,\ldots,D_n+D_c\}$
- Each categorical variable $x_i^{d'}$ has $n_{d'}$ possible categories

The VSN architecture transforms each feature (numerical or categorical) into a D_e-dimensional embedding vector, then applies gating mechanisms for feature selection and final prediction.

1. **Input Transformation Layer**

The first layer processes numerical and categorical features differently:

- **Dense Layers** for numerical features: $x_i^d \to \boldsymbol{\xi}_i^d \in \mathbb{R}^{D_e}$
- **Embedding Layers** for categorical features: $x_i^{d'} \to \boldsymbol{\xi}_i^{d'} \in \mathbb{R}^{D_e}$

Q1. Consider the embedding layer for categorical features:

- For a categorical variable d' with $n_{d'}$ categories embedded into D_e dimensions, calculate the total number of parameters in the embedding layer.
- Explain two key advantages of embedding layers over one-hot encoding for categorical variables with high cardinality.

- **Parameter Count:** For a categorical variable d' with $n_{d'}$ categories embedded into D_e dimensions, the embedding layer contains exactly $n_{d'} \times D_e$ parameters. Each category is mapped to a unique D_e-dimensional vector, requiring a lookup table of size $n_{d'} \times D_e$.
- **Advantages over One-Hot Encoding:**
  - **1. Dimensionality Efficiency:** One-hot encoding creates sparse vectors of size $n_{d'}$ with only one non-zero element, while embeddings create dense vectors of size D_e where $D_e \ll n_{d'}$ for high-cardinality variables. This significantly reduces memory usage and computational overhead.
  - **2. Learned Representations:** Embedding layers learn meaningful representations that capture semantic relationships between categories through training, while one-hot encoding treats all categories as equally distant. This enables the model to understand that similar categories should have similar embeddings, improving generalization and feature interactions.

The Input Transformation layer, illustrated in Figure 4, converts the input matrix to embedded feature vectors.

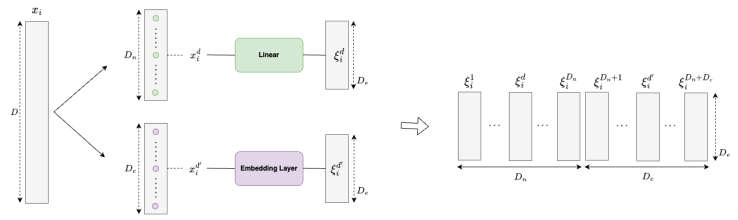

Figure 4: Input Transformation Layer processing numerical and categorical features

After applying the Input Transformation layer to a batch of N samples with D_n numerical and D_c categorical features, each embedded into D_e dimensions.

Q2. What is the shape of the output tensor containing all embedded features?

- The output tensor has shape $\mathbb{R}^{N\times(D_n+D_c)\times D_e}$, where:
  - N is the batch size (number of samples)
  - $(D_n + D_c)$ is the total number of features (numerical + categorical)
  - D_e is the embedding dimension
- Each of the N samples has $(D_n + D_c)$ features, and each feature is represented as a D_e-dimensional embedding vector, regardless of whether it originated from a numerical or categorical variable.

2. **Gated Residual Networks (GRN)**

Each embedded feature $\boldsymbol{\xi}_i^d$ is processed through a Gated Residual Network, as shown in Figure 5, which applies gating mechanisms and residual connections to control information flow.

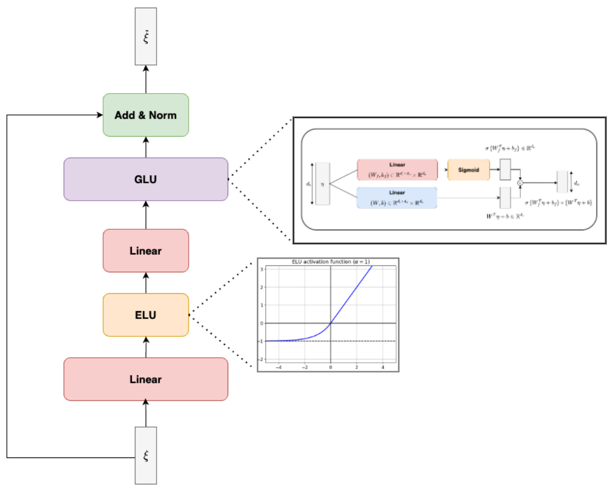

Figure 5: Gated Residual Network architecture with gating mechanism

The GRN transformation produces $\tilde{\boldsymbol{\xi}}_i^d \in \mathbb{R}^{D_o}$ for each feature d. These transformed features are then processed through the feature selection mechanism illustrated in Figure 6.

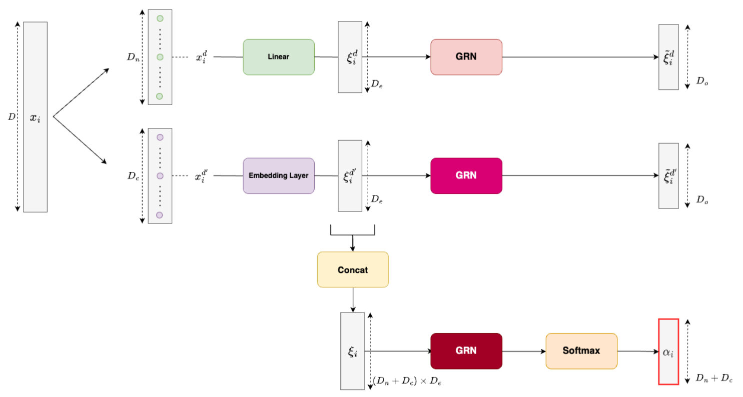

Figure 6: GRN transformation and feature selection mechanism

Q3. The concatenated feature vector $\boldsymbol{\xi}_i$ is processed through a GRN followed by a Softmax activation to produce attention weights $\boldsymbol{\alpha}_i$:

- What is the purpose of applying the Softmax function to obtain $\boldsymbol{\alpha}_i$?
- What is the shape of $\boldsymbol{\alpha}_i$ and what constraint does the Softmax impose?

- What does each component α_i^d represent in the context of feature importance?

- **Purpose of Softmax:** The Softmax function normalizes the raw attention scores to create a probability distribution over features. This ensures that attention weights are non-negative and sum to 1, creating a valid weighting scheme for feature selection.
- **Shape and Constraint:** $\boldsymbol{\alpha}_i \in \mathbb{R}^{D_n+D_c}$ where each component $\alpha_i^d \geq 0$ and $\sum_{d=1}^{D_n+D_c} \alpha_i^d = 1$. The Softmax imposes the constraint that all attention weights are non-negative and sum to unity.
- **Feature Importance Interpretation:** Each component α_i^d represents the relative importance of feature d for sample i. Higher values indicate that feature d is more relevant for making predictions about sample i, while values close to zero suggest the feature should be suppressed. This provides interpretable feature selection at the instance level.

3. **Feature Selection and Final Representation**

The final feature representation is computed as a weighted sum:

$$\tilde{\boldsymbol{\xi}}_i = \sum_{d=1}^{D_n+D_c} \alpha_i^d \tilde{\boldsymbol{\xi}}_i^d$$

The complete VSN architecture is illustrated in Figure 7.

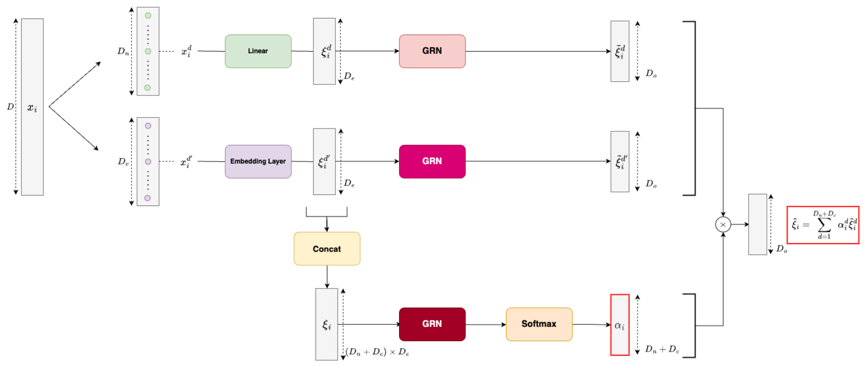

Figure 7: Complete Variable Selection Network architecture

Q4. Describe the complete forward propagation from an input batch matrix $\mathbf{X} \in \mathbb{R}^{N_b \times D}$ to the final representation $\tilde{\boldsymbol{\xi}} \in \mathbb{R}^{N_b \times D_o}$. For each transformation step, specify:

- The operation performed
- The shape of the data after the transformation
- The key parameters involved

Assume a batch size of N_b samples.

i. **Step 1 - Input Transformation:**
   - Operation: Input $\mathbf{X} \in \mathbb{R}^{N_b \times D}$ is processed through embedding/dense layers. Numerical features use a linear transformation $\mathbf{W}_{\text{num}} \in \mathbb{R}^{D_e \times 1}$; categorical features use embedding lookup tables.
   - Output shape: $\mathbb{R}^{N_b \times (D_n + D_c) \times D_e}$
   - Parameters: $\mathbf{W}_{\text{num}}$ for numerical features, embedding matrices for categorical features

ii. **Step 2 - Individual GRN Processing:**
   - Operation: Each embedded feature $\boldsymbol{\xi}_i^d$ is processed through its own GRN: $\tilde{\boldsymbol{\xi}}_i^d = \text{GRN}(\boldsymbol{\xi}_i^d)$
   - Output shape: $\mathbb{R}^{N_b \times (D_n + D_c) \times D_o}$
   - Parameters: $\boldsymbol{\theta}_{\text{GRN}}$ (one set per feature)

iii. **Step 3 - Attention Weight Calculation:**
   - Operation: Concatenated features $[\boldsymbol{\xi}_i^1; \ldots; \boldsymbol{\xi}_i^{D_n+D_c}]$ are processed through a shared GRN followed by Softmax to compute attention weights
   - Output shape: $\boldsymbol{\alpha} \in \mathbb{R}^{N_b \times (D_n + D_c)}$
   - Parameters: shared $\boldsymbol{\theta}_{\text{GRN}}$ for the attention mechanism

iv. **Step 4 - Weighted Aggregation:**
   - Operation: Final representation computed as $\tilde{\boldsymbol{\xi}}_i = \sum_{d=1}^{D_n+D_c} \alpha_i^d \tilde{\boldsymbol{\xi}}_i^d$
   - Output shape: $\mathbb{R}^{N_b \times D_o}$
   - Parameters: none (pure aggregation)

4. **Binary Classification Layer**

Since we have a binary classification problem for predicting meta-labels, we add a final Dense layer to the VSN architecture.

Q5. Describe the final Dense layer for binary classification:

- What transformation does this layer perform on $\tilde{\boldsymbol{\xi}}_i \in \mathbb{R}^{D_o}$?
- How many parameters does this layer contain?
- What is the shape of the final output and what does it represent?
- What activation function should be used and why?

- **Transformation:** The Dense layer performs a linear transformation $p_i = \sigma(\mathbf{w}^T \tilde{\boldsymbol{\xi}}_i + b)$ where $\mathbf{w} \in \mathbb{R}^{D_o}$ is the weight vector and $b \in \mathbb{R}$ is the bias term.
- **Parameter Count:** The layer contains $D_o + 1$ parameters: D_o weights in vector $\mathbf{w}$ plus 1 bias term b.
- **Output Shape:** The final output has shape $\mathbb{R}^{N_b \times 1}$, where each element represents the probability that the corresponding sample belongs to the positive class (profitable trade).
- **Activation Function:** Sigmoid activation $\sigma(z) = \frac{1}{1+e^{-z}}$ should be used because it maps real-valued outputs to the interval $(0, 1)$, making them interpretable as probabilities. This is essential for binary classification as it ensures outputs represent valid probability estimates for the positive class.

5. **Loss Function and Training**

Q6. What loss function should be used to train this VSN model for binary classification? Provide the mathematical formulation.

Binary Cross-Entropy Loss should be used for training the VSN model. The mathematical formulation is:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} \left[y_i^{\text{meta}} \log(p_i) + (1 - y_i^{\text{meta}}) \log(1 - p_i) \right]$$

where $\boldsymbol{\theta}$ represents all model parameters, $y_i^{\text{meta}} \in \{0, 1\}$ is the true binary label, $p_i \in (0, 1)$ is the predicted probability from the sigmoid output, and N is the number of training samples.

Q7. Complete the Stochastic Gradient Descent training algorithm for the VSN model:

Algorithm 7 SGD Training for Variable Selection Network

```
Require: Training data, validation data, learning rate $\eta$, batch size $B$, epochs $E$
Ensure: Optimal parameters $\boldsymbol{\theta}^*$
1:  Initialize parameters $\boldsymbol{\theta}^{(0)}$ = [BLANK 1]                              ▷ Fill in the blank
2:  Initialize train_losses = [], val_losses = []
3:  $t \leftarrow 0$
4:  for epoch $e = 1$ to $E$ do
5:      for each batch $\{(\mathbf{Z}_i, y_i^{\text{meta}})\}_{i \in \mathcal{B}}$ in training data do
6:          Perform forward propagation: $\{p_i\}_{i \in \mathcal{B}} = \text{VSN}(\{\mathbf{Z}_i\}_{i \in \mathcal{B}}; \boldsymbol{\theta}^{(t)})$
7:          Calculate loss function: $L_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \ell(p_i, y_i^{\text{meta}})$
8:          Update weights: $\boldsymbol{\theta}^{(t+1)}$ = [BLANK 2]                       ▷ Fill in the blank
9:          $t \leftarrow t + 1$
10:     end for
11:     Compute training loss and append to train_losses
12:     Compute validation loss and append to val_losses
13: end for
14: Select $\boldsymbol{\theta}^*$ with minimum validation loss
15: return $\boldsymbol{\theta}^*$
```

- **BLANK 1:** Initialize weights randomly (optionally using Xavier or He initialization, e.g., $\mathcal{N}(0, \sqrt{2/n_{\text{in}}})$)
- **BLANK 2:** $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} L_{\mathcal{B}}$ (gradient descent update rule)