Algorithms and Data Structures with applications in Machine Learning

Final Exam

January 3, 2025

The purpose of this exam is to provide you with an opportunity to demonstrate your understanding of the topics covered in class and to apply the concepts we have explored. Below are the instructions to help you succeed:

Instructions:

- Ensure your handwriting is clear and leave adequate space for readability.
- Be concise and precise in your answers.
- Show all calculations to allow for partial credit where applicable.
- Calculators are permitted.

The exam is structured as follows:

- An **MCQ** section with 10 multiple-choice questions, worth a total of 20 marks.
- A **problem-solving section** consisting of 4 exercises, each worth 20 marks.
- Although the exercises are interconnected, they can be tackled and solved independently.

Section 1: Multiple Choice Questions (20 marks)

Q. 1 Recall the desired approximation for the GloVe model:

$$\log X_{ij} \approx W_i^T \tilde{W}_j + b_i + \tilde{b}_j$$

The term $W_i^T \tilde{W}_j$ represents the relationship between the word indexed by i and the word indexed by j through their embeddings.

Assume that we have trained embeddings $W_{\text{equity}}, W_{\text{market}}$ using this approximation and compare the dot product of these embeddings to the dot product of one-hot vectors for the same words.

Which of the following best describes the comparison? a

a. The dot product $W_{\text{equity}}^T W_{\text{market}}$ from embeddings is positive and large, while the dot product of their one-hot vectors is 0. a

b. The dot product $W_{\text{equity}}^T W_{\text{market}}$ from embeddings is 0, and the dot product of their one-hot vectors is also 0.

c. The dot product $W_{\text{equity}}^T W_{\text{market}}$ from embeddings is smaller than the dot product of their one-hot vectors, which is positive.

d. The dot product $W_{\text{equity}}^T W_{\text{market}}$ from embeddings is negative, while the dot product of their one-hot vectors is 0.

Q. 2 Recall the cost function J for the GloVe algorithm:

$$J(\theta) = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(\log X_{ij} - W_i^T \tilde{W}_j - b_i - \tilde{b}_j)^2 \tag{1}$$

The parameters to optimize are: a

- $W \in \mathcal{M}_{V,D}(\mathbb{R})$, the first embedding matrix,
- $\tilde{W} \in \mathcal{M}_{V,D}(\mathbb{R})$, the second embedding matrix,
- $b \in \mathbb{R}^V$, the bias vector for W,
- $\tilde{b} \in \mathbb{R}^V$, the bias vector for $\tilde{W}$.

What is the total number of parameters to train in the model, assuming the vocabulary size is V and the embedding dimension is D? a

a. $2VD + 2V$

b. $VD + V$

c. $V^2 + D^2$

d. $2V + D$

Q. 3 Which of the following equations correctly represents $\nabla_{\tilde{W}_j} J(\tilde{W}_j)$, the gradient of the loss (eq 1), based on its shape? b

a. $\nabla_{\tilde{W}_j} J(\tilde{W}_j) = -2 \sum_{i'=1}^{V} f(X_{i'j}) \left(\log X_{i'j} - W_{i'}^T \tilde{W}_j - b_{i'} - \tilde{b}_j \right)$

b. $\nabla_{\tilde{W}_j} J(\tilde{W}_j) = -2 \sum_{i'=1}^{V} f(X_{i'j}) \left(\log X_{i'j} - W_{i'}^T \tilde{W}_j - b_{i'} - \tilde{b}_j \right) W_{i'}$

c. $\nabla_{\tilde{W}_j} J(\tilde{W}_j) = -2 \sum_{i'=1}^{V} f(X_{i'j}) \left(\log X_{i'j} - W_{i'}^T \tilde{W}_j - b_{i'} - \tilde{b}_j \right) W_{i'} W_{i'}^T$

d. $\nabla_{\tilde{W}_j} J(\tilde{W}_j) = 0$

Q. 4 In the context of stable matchings, a valid partner for a man m is defined as: a

a. A woman w such that m and w are matched in at least one stable matching.

b. The woman w at the top of m's preference list.

c. Any woman w who accepts a proposal from m during the Gale-Shapley algorithm.

d. A woman w such that w and m are unmatched in all stable matchings.

Q. 5 Consider the following preferences of 5 men over 5 women, given as indices: c

$$0 : [2, \mathbf{0}, 1, \mathbf{3}, 4]$$
$$1 : [1, 0, \mathbf{2}, 3, 4]$$
$$2 : [3, 0, \mathbf{1}, \mathbf{4}, 2]$$
$$3 : [\mathbf{3}, 1, \mathbf{2}, 0, 4]$$
$$4 : [\mathbf{4}, 3, \mathbf{2}, 1, 0]$$

The preferences shown in **bold** indicate the valid partners for each man. Using the Gale-Shapley algorithm (where men propose), the resulting stable matches are:

a. $\{(0,2),(1,0),(2,1),(3,3),(4,4)\}$

b. $\{(0,3),(1,1),(2,0),(3,2),(4,4)\}$

c. $\{(0,0),(1,2),(2,1),(3,3),(4,4)\}$

d. $\{(0,0),(1,2),(2,1),(3,0),(4,3)\}$

Q. 6 Let Y be a random variable that can take values from the finite set $\mathcal{Y} = \{y_1, y_2, \ldots, y_n\}$. The probability distribution of Y is denoted as $p(y) = \mathbb{P}(Y = y)$, where $y \in \mathcal{Y}$. The entropy of Y is defined as:

$$H(Y) = -\sum_{y \in \mathcal{Y}} p(y) \log_2(p(y)).$$

Which of the following statements best explains the meaning of entropy? d

a. Entropy measures the likelihood of the most probable value of Y.

b. Entropy represents the total uncertainty across all possible outcomes of Y, without averaging.

c. Entropy is the measure of how many outcomes Y can possibly take.

d. Entropy quantifies the average uncertainty reduction (in bits) when the value of Y is revealed.

Q. 7 Decision Trees are known to have a tendency to overfit the training data, especially when the tree is deep. Random Forest addresses this issue by: a

a. Combining multiple Decision Trees through bagging, where each tree is trained on a random subset of the training data.

b. Training a single shallow Decision Tree with reduced depth to minimize overfitting.

c. Using a single Decision Tree but regularizing it by pruning unnecessary branches.

d. Combining multiple Decision Trees through boosting, where each tree corrects the errors of the previous one.

Q. 8 Which of the following is **not** a hyperparameter of a Random Forest Classifier? d

a. `n_estimators`

b. `max_features`

c. `min_samples_leaf`

d. `learning_rate`

Q. 9 Given a highly imbalanced dataset for a binary classification problem, with 99% of positive labels, which of the following evaluation metrics is **not suitable** for assessing the model's performance? c

a. Precision

b. Recall

c. Accuracy

d. F1-score

Q. 10 Which of the following statements about the AUC (Area Under the Curve) is **not true**? d

a. AUC evaluates the model's performance across all possible decision thresholds.

b. AUC quantifies the likelihood that a randomly chosen positive sample is assigned a higher score than a randomly chosen negative sample.

c. AUC is the area under the ROC curve, where the ROC plots the True Positive Rate (TPR) against the False Positive Rate (FPR).

d. AUC is sensitive to class imbalance and may significantly degrade in performance when one class dominates.

Section 2: Node Classification Problem on the Cora dataset (80 marks)

The Cora dataset represented in figure 1 is a benchmark dataset widely used for graph-based machine learning tasks. It represents a citation network where each node corresponds to a research paper and edges indicate citation relationships. Each paper is labeled into one of seven distinct classes corresponding to specific topics or fields of research.

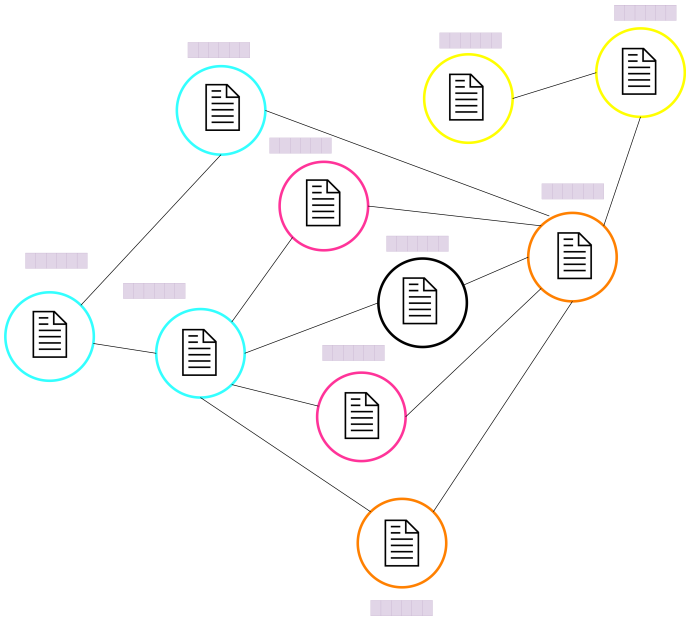

Figure 1: The Cora Dataset

The feature representation of each node is a 1433-dimensional binary Bag-of-Words (BoW) vector. Let $x_i \in \{0,1\}^{1433}$ denote the BoW feature vector for the ith paper, where each dimension indicates the presence (1) or absence (0) of a specific word in the paper. For example, if the vocabulary contains the words *machine*, *learning*, and *graphs*, and a paper contains only *machine* and *graphs*, its BoW vector would have 1s in the positions corresponding to *machine* and *graphs*, and 0 elsewhere. These vectors provide a sparse representation of the textual content of the papers.

The dataset consists of 2708 nodes and 10556 edges, with each node having 1433 binary features. It is divided into training, validation, and test subsets.

In this exam, we will explore four key aspects of graph-based machine learning. Each exercise builds on the concepts of representation learning, classification, and optimization to solve practical problems.

- In the first exercise, we will focus on generating dense, semantic-rich D-dimensional representations for each paper in the dataset. Using the GloVe algorithm, the sparse binary Bag-of-Words vectors will be replaced with embeddings learned from the co-occurrence patterns of words. This exercise emphasizes the importance of capturing semantic relationships in textual data.

- The second exercise will involve generating graph-based embeddings and performing classification. First, we will use the Node2vec algorithm to create embeddings that capture the structural properties of the citation network. These embeddings will then be used as features for a Random Forest classifier to predict the class labels of the nodes in the Cora dataset. This exercise combines unsupervised graph representation learning with supervised node classification.

- In the third exercise, we will employ a Graph Neural Network (GNN) to learn embeddings for the nodes. Unlike the Node2vec approach, the GNN will directly optimize the embeddings using cross-entropy loss while simultaneously performing node classification. This step integrates graph structure and feature information to achieve higher classification accuracy.

- Finally, in the fourth exercise, we will address the stable matching problem. Using the Gale-Shapley algorithm, reviewers will be assigned to research papers based on predefined preferences.

Exercise 1 (Learning Word and Document Embeddings with GloVe)**.**

In this exercise, we will explore the GloVe approach for generating dense, semantic-rich word embeddings. The goal is to use the co-occurrence patterns of words in the corpus to learn D-dimensional embedding vectors for each word. These embeddings will then be used to derive an embedding vector for each research paper.

We create the corpus by combining all the research papers in the dataset. From this corpus, we extract a vocabulary and assign an index to each word. We will use the term "word i" to refer to the word associated with index $i \in \{1, \dots, \mathcal{V}\}$ in this vocabulary. A co-occurrence matrix X is then constructed, where each entry X_{ij} represents the co-occurrence count of word i and word j within a given context size.

1. Complete the blank in the Algorithm 2 to compute the co-occurrence matrix X from the corpus:

Algorithm 1 Getting the Co-Occurrence Matrix

Require: sequences (list of lists of integers), context_size
Ensure: X (the co-occurrence matrix)
Initialize matrix $X \in \mathbb{M}_{\mathcal{V},\mathcal{V}}(\mathbb{R})$ with zeros
... ▷ Fill in the blank
return X

Algorithm 2 Getting the Co-Occurrence Matrix

Require: sequences (list of lists of integers), context_size
Ensure: X (the co-occurrence matrix)
Initialize matrix $X \in \mathbb{M}_{\mathcal{V},\mathcal{V}}(\mathbb{R})$ with zeros
for sequence in sequences **do**
  for word $w[i]$ of index i in sequence **do**
    for word $w[j]$ of index j in the context of $w[i]$ **do**
      $X[w[i], w[j]] \leftarrow X[w[i], w[j]] + \frac{1}{|i-j|}$
    end for
  end for
end for
  return X

2. Derive the loss function for the GloVe model, starting from the desired approximation:

$$\log X_{ij} \approx W_i^T \tilde{W}_j + b_i + \tilde{b}_j.$$

where W_i and $\tilde{W}_j$ are the embedding vectors for the word i and the context word j, respectively, and b_i and $\tilde{b}_j$ are bias terms.

The loss function can then be written as follows:

$$J(\theta) = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(\log X_{ij} - W_i^T \tilde{W}_j - b_i - \tilde{b}_j)^2$$

We will minimize this loss function with respect to all parameters: W, $\tilde{W}$, b, and $\tilde{b}$.

Two training methods can be applied: alternating least squares (ALS) and gradient descent.

Let us consider $i \in \{1 \dots, \mathcal{V}\}$, the gradient of the loss function with respect to the parameters W_i is provided to you:

$$\nabla_{W_i} J(W_i) = -2 \sum_{j'=1}^{\mathcal{V}} f(X_{ij'}) \left(\log X_{ij'} - W_i^T \tilde{W}_{j'} - b_i - \tilde{b}_{j'} \right) \tilde{W}_{j'} \quad (2)$$

3. Determine $W_i^{(t+1)}$, the updated value of W_i at iteration $t+1$ using the Alternating Least Squares method, as a function of the other parameters $(\tilde{W}_j^{(t)})_{1 \le j \le \mathcal{V}}, b_i^{(t)}$, and $(\tilde{b}_j^{(t)})_{1 \le j \le \mathcal{V}}$ at iteration t.

$$\begin{aligned}
&\nabla_{W_i^{(t+1)}} J(W_i) = 0 \\
&\iff -2 \sum_{j'=1}^{V} f(X_{ij'}) \left(\log X_{ij'} - {W_i^{(t+1)}}^T \tilde{W}_{j'}^{(t)} - b_i^{(t)} - \tilde{b}_{j'}^{(t)} \right) \tilde{W}_{j'}^{(t)} = 0 \\
&\iff \sum_{j'=1}^{V} f(X_{ij'}) \left(\log X_{ij'} - b_i^{(t)} - \tilde{b}_{j'}^{(t)} \right) \tilde{W}_{j'}^{(t)} = \sum_{j'=1}^{V} f(X_{ij'}) {W_i^{(t+1)}}^T \tilde{W}_{j'}^{(t)} \tilde{W}_{j'}^{(t)} \\
&\iff \sum_{j'=1}^{V} f(X_{ij'}) \left(\log X_{ij'} - b_i^{(t)} - \tilde{b}_{j'}^{(t)} \right) \tilde{W}_{j'}^{(t)} = \left(\sum_{j'=1}^{V} f(X_{ij'}) {\tilde{W}_{j'}^{(t)}}^T \tilde{W}_{j'}^{(t)} \right) W_i^{(t+1)} \\
&\iff W_i^{(t+1)} = \left(\sum_{j'=1}^{V} f(X_{ij'}) \tilde{W}_{j'}^{(t)} {\tilde{W}_{j'}^{(t)}}^T \right)^{-1} \\
&\qquad \cdot \left(\sum_{j'=1}^{V} f(X_{ij'}) (\log X_{ij'} - b_i^{(t)} - \tilde{b}_{j'}^{(t)}) \tilde{W}_{j'}^{(t)} \right)
\end{aligned}$$

Use the hint:

$$\forall a, b \in \mathbb{R}^n, \quad (a^T b)\, b = (b\, b^T)\, a.$$

4. Complete the blank in Algorithm 4 for training the GloVe model using gradient descent:

Algorithm 3 Training the GloVe Model Using Gradient Descent

Require: X, learning rate η, number of epochs N_{epochs}
Ensure: Trained parameters W, $\tilde{W}$, b, $\tilde{b}$
Initialize W, $\tilde{W}$, b, $\tilde{b}$ randomly
for $t = 1, \ldots, N_{\text{epochs}}$ **do**
    ... ▷ Fill in the blank
end for
return W, $\tilde{W}$, b, $\tilde{b}$

Algorithm 4 Training the GloVe Model Using Gradient Descent

Require: X, learning rate η, number of epochs N_{epochs}
Ensure: Trained parameters W, $\tilde{W}$, b, $\tilde{b}$
Initialize W, $\tilde{W}$, b, $\tilde{b}$ randomly
for $t = 1, \ldots, N_{\text{epochs}}$ **do**
    for $i = 1, \ldots, V$ **do**
        $W_i \leftarrow W_i - \eta \nabla_{W_i} J(W, \tilde{W}, b, \tilde{b})$
    end for
    for $j = 1, \ldots, V$ **do**
        $\tilde{W}_j \leftarrow \tilde{W}_j - \eta \nabla_{\tilde{W}_j} J(W, \tilde{W}, b, \tilde{b})$
    end for
    for $i = 1, \ldots, V$ **do**
        $b_i \leftarrow b_i - \eta \nabla_{b_i} J(W, \tilde{W}, b, \tilde{b})$
    end for
    for $j = 1, \ldots, V$ **do**
        $\tilde{b}_j \leftarrow \tilde{b}_j - \eta \nabla_{\tilde{b}_j} J(W, \tilde{W}, b, \tilde{b})$
    end for
end for
return W, $\tilde{W}$, b, $\tilde{b}$

Each research paper $n \in \{1, \ldots, V\}$ is represented by a feature vector $\mathbf{x}_n \in \mathbb{R}^D$, which is computed as the average of the GloVe embedding vectors W_{w_k} for all words w_k in the paper:

$$\mathbf{x}_n = \frac{1}{|\mathcal{W}_n|} \sum_{k \in \mathcal{W}_n} W_{w_k},$$

where $\mathcal{W}_n$ denotes the set of word indices in research paper n.

5. Explain two advantages of using the GloVe-based feature vectors over the initial Bag-of-Words (BoW) feature vectors.

(a) **Semantic Relationships:** GloVe captures semantic relationships and word contexts by leveraging co-occurrence statistics, mapping similar words closer in the embedding space. BoW lacks this semantic understanding.

(b) **Efficiency:** GloVe produces dense, low-dimensional vectors, reducing memory and computational costs, whereas BoW representations are sparse and high-dimensional.

Exercise 2 (Node Classification using Graph Based Embedding and Tree Based Models)**.**

In this exercise, we will focus on the Node2vec algorithm to create graph-based embedding vectors for nodes.

After training the graph-based embeddings, we will apply a Random Forest classifier to map these embeddings to one of the seven class labels in the dataset.

A key part of generating node embeddings in the Node2vec algorithm involves sampling neighborhoods of nodes using biased random walks. The bias is controlled by two hyperparameters:

- **Return parameter** (p): Controls the likelihood of immediately revisiting a node in the walk.
- **In-out parameter** (q): Controls the likelihood of visiting nodes closer or farther away from the starting node.

Refer to Figure 2. Suppose the current node is w, the starting node of the random walk is u, and the neighbors of w are S_1, S_2, S_3, and S_4.

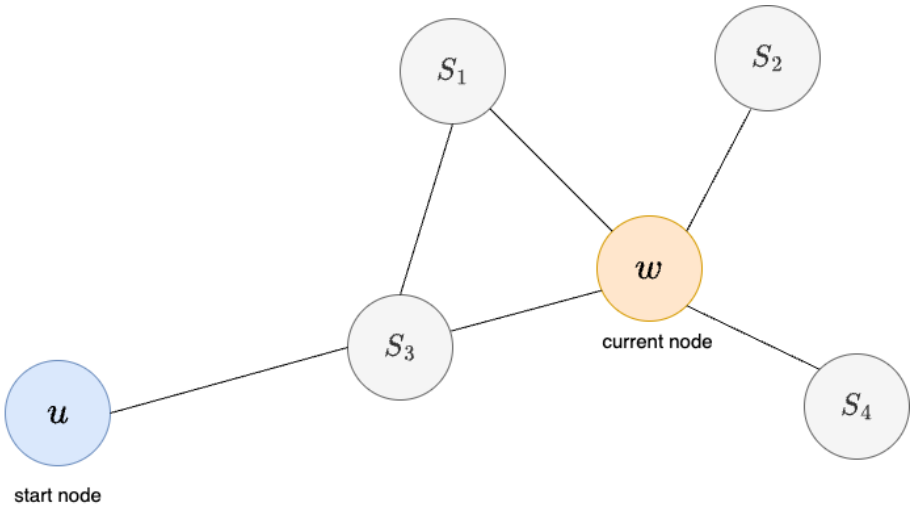

Figure 2: Sampling a neighbor in a biased Random Walk

1. Calculate the probabilities π_1, π_2, π_3, and π_4, where π_i is the probability of sampling neighbor S_i in a biased random walk starting from node u (the starting node) and currently at node w. Use the hyperparameters p and q and the given context shown in Figure 2.

   The normalized probabilities are calculated as follows:

$$Z = 1 + \frac{1}{q} + \frac{1}{p} + \frac{1}{q}.$$

   The normalized probabilities are:

$$\pi_1 = \frac{1}{Z}, \quad \pi_2 = \frac{\frac{1}{q}}{Z}, \quad \pi_3 = \frac{\frac{1}{p}}{Z}, \quad \pi_4 = \frac{\frac{1}{q}}{Z}.$$

   Let $u, v \in V$ denote nodes in the graph, where V is the set of all nodes. For each node u, let $\mathbf{Z}_u \in \mathbb{R}^D$ be its graph-based embedding vector. The embeddings are designed such that graph similarity between nodes u and v is reflected as similarity in the embedding space between $\mathbf{Z}_u$ and $\mathbf{Z}_v$.

2. If the similarity between nodes u and v is defined as the probability of visiting node v on a random walk starting from node u, what should be the similarity in the embedding space between $\mathbf{Z}_u$ and $\mathbf{Z}_v$?

   The similarity in the embedding space between $\mathbf{Z}_u$ and $\mathbf{Z}_v$ is given by:

$$\text{Similarity}(\mathbf{Z}_u, \mathbf{Z}_v) = \frac{\exp(\mathbf{Z}_u^\top \mathbf{Z}_v)}{\sum_{n \in V} \exp(\mathbf{Z}_u^\top \mathbf{Z}_n)}.$$

   We denote $\mathcal{N}_R(u)$ the set of nodes sampled via random walks starting from u based on the biased strategy R

3. Derive the unsupervised loss function that translates graph similarity into embedding similarity.

   To translate graph similarity into embedding similarity, we aim to maximize the probability of visiting neighbors in the graph using embedding-based similarities. The loss function is derived as the negative log-likelihood of the neighbors in the embedding space:

$$\mathcal{L}(\theta) = -\sum_{u \in V} \sum_{v \in \mathcal{N}_R(u)} \log \left(\frac{\exp(\mathbf{Z}_u^\top \mathbf{Z}_v)}{\sum_{n \in V} \exp(\mathbf{Z}_u^\top \mathbf{Z}_n)} \right).$$

   This loss function ensures that embeddings of graph neighbors are similar while maintaining the overall graph structure in the embedding space.

   After obtaining the embeddings, we wish to apply a Random Forest model to classify each paper into one of the 7 targets.

4. Derive the expression for the information gain at a decision tree node.

   Use the following notations:

   - D_p: The dataset at the parent node.
   - D_{left}, D_{right}: The datasets at the left and right child nodes.
   - N_p: The total number of samples at the parent node.
   - N_{left}, N_{right}: The total number of samples in the left and right child nodes.
   - $I(D)$: The impurity measure of a dataset D.

5. We aim to tune the hyperparameters n_estimators and max_depth of the Random Forest using Grid Search with cross-validation. Complete the blanks in Algorithm 6:

Algorithm 5 Hyperparameter Tuning with Cross Validation

Require: Training data X, labels y, hyperparameter grid H, number of folds k
Ensure: Best hyperparameter combination

```
Initialize best_score ← 0, best_params ← None
for each (n_estimators, max_depth) in H do
    scores ← []
    for fold = 1 to k do
        Split X and y into training and validation sets: (X_train, y_train, X_val, y_val)
        Train a Random Forest with n_estimators and max_depth on (X_train, y_train)
        Compute validation score and store: scores.append(evaluate(model, X_val, y_val))
    end for
    avg_score ← ...                                   ▷ Fill in the blank
    if avg_score > best_score then
        best_score ← avg_score
        ...                                           ▷ Fill in the blank
    end if
end for
return best_params
```

Algorithm 6 Hyperparameter Tuning with Cross Validation

Require: Training data X, labels y, hyperparameter grid H, number of folds k
Ensure: Best hyperparameter combination

Initialize best_score $\leftarrow 0$, best_params $\leftarrow$ None
for each (n_estimators, max_depth) in H **do**
  scores $\leftarrow []$
  for fold $= 1$ to k **do**
    Split X and y into training and validation sets: $(X_{\text{train}}, y_{\text{train}}, X_{\text{val}}, y_{\text{val}})$
    Train a Random Forest with $n_{\text{estimators}}$ and max_depth on $(X_{\text{train}}, y_{\text{train}})$
    Compute validation score and store: `scores.append(evaluate(model, X_val, y_val))`
  end for
  avg_score $\leftarrow \frac{\sum \text{scores}}{k}$ ▷ Fill in the blank
  if avg_score $>$ best_score **then**
    best_score $\leftarrow$ avg_score
    best_params $\leftarrow$ (n_estimators, max_depth) ▷ Fill in the blank
  end if
end for
return best_params

Exercise 3 (Graph Neural Networks for Node Embedding and Classification)**.**

In this exercise, we will use Graph Neural Networks (GNNs) to learn graph-based embedding vectors for nodes in the graph.

These embeddings will be learned by leveraging the GloVe feature vectors associated with each research paper and the graph structure. The embeddings will then be used for node classification into the seven class labels.

The GNN operates on the principle of message passing, where information from a node's neighbors is aggregated and combined iteratively to update the node's embedding.

1. Complete the blanks in Algorithm 8 for message passing:

Algorithm 7 Message Passing Framework

Require: Graph $G = (V, E)$, node features $\{\mathbf{x}_v \mid v \in V\}$, number of iterations K, $f_{\text{aggregate}}$, f_{update}
Ensure: Final node embeddings $\{\mathbf{h}_v^{(K)} \mid v \in V\}$

Initialize embeddings: ... ▷ Fill in the blank
for $k = 1$ to K **do**
  for each node $v \in V$ **do**
    ... ▷ Fill in the blank
  end for
end for
return $\{\mathbf{h}_v^{(K)} \mid v \in V\}$ =0

Algorithm 8 Message Passing Framework

Require: Graph $G = (V, E)$, node features $\{\mathbf{x}_v \mid v \in V\}$, number of iterations K, $f_{\text{aggregate}}$, f_{update}
Ensure: Final node embeddings $\{\mathbf{h}_v^{(K)} \mid v \in V\}$

Initialize embeddings: $\mathbf{h}_v^{(0)} \leftarrow \mathbf{x}_v$ for all $v \in V$ ▷ Fill in the blank
for $k = 1$ to K **do**
  for each node $v \in V$ **do**
    Compute aggregated messages:

$$\mathbf{a}_v^{(k)} \leftarrow f_{\text{aggregate}}\left(\{\mathbf{h}_u^{(k-1)} \mid u \in \mathcal{N}(v)\}\right)$$

    Update node embedding:

$$\mathbf{h}_v^{(k)} \leftarrow f_{\text{update}}(\mathbf{a}_v^{(k)}, \mathbf{h}_v^{(k-1)})$$

  end for
end for
return $\{\mathbf{h}_v^{(K)} \mid v \in V\}$

The node embeddings are learned using a supervised loss function, specifically cross-entropy loss, defined as:

$$\mathcal{L}(\theta) = -\sum_{v \in V_{\text{train}}} \sum_{c=1}^{C} y_v^c \log \hat{y}_v^c$$

Where:

- y_v^c: Ground-truth label (one-hot encoded) for node v.

- $\hat{y}_v^c = \text{softmax}\left(W_{\text{out}}\mathbf{h}_v^{(K)}\right)$: Predicted probability of class c, computed from the node embedding.

After training, the performance of the GNN model will be compared to the Node2vec and Random Forest method. Below are the confusion matrices for the two methods:

- **Confusion Matrix for Node2vec and Random Forest Model**

  The confusion matrix below shows the classification results for the Node2vec and Random Forest model. The rows represent the actual labels, and the columns represent the predicted labels (1 to 7).

Actual \Predicted	1	2	3	4	5	6	7
1	15	2	1	1	0	0	1
2	1	12	0	3	2	0	2
3	0	1	14	3	0	0	2
4	2	0	1	11	0	5	1
5	1	0	0	0	18	0	1
6	0	0	7	0	1	12	0
7	0	1	1	5	0	0	13

- **Confusion Matrix for the GNN Model**

  The confusion matrix below shows the classification results for the GNN model. The rows represent the actual labels, and the columns represent the predicted labels (1 to 7).

Actual \Predicted	1	2	3	4	5	6	7
1	19	0	0	0	0	0	1
2	0	20	0	0	0	0	0
3	0	0	18	2	0	0	0
4	0	0	0	17	2	0	1
5	0	1	1	0	17	0	1
6	0	2	0	2	0	16	0
7	0	0	0	0	1	1	18

2. Calculate the accuracy of each model.

   The accuracy is defined as:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}.$$

   For Model 1:

$$\text{Accuracy}_1 = \frac{15+12+14+11+18+12+13}{140} = 0.679.$$

   For Model 2:

$$\text{Accuracy}_2 = \frac{19+20+18+17+17+16+18}{140} = 0.893.$$

3. Calculate the precision of each model.

   The precision is defined as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}.$$

   For Model 1:

Label	TP	FP	FN	Precision
1	15	4	5	0.789
2	12	4	8	0.750
3	14	10	6	0.583
4	11	12	9	0.478
5	18	3	2	0.857
6	12	5	8	0.706
7	13	7	7	0.650
Final Precision				0.688

   For Model 2:

Label	TP	FP	FN	Precision
1	19	0	1	1.000
2	20	3	0	0.870
3	18	1	2	0.947
4	17	4	3	0.810
5	17	3	3	0.850
6	16	1	4	0.941
7	18	3	2	0.857
Final Precision				0.896

4. Calculate the recall of each model.

   The recall is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}.$$

For Model 1:

Label	TP	FP	FN	Recall
1	15	4	5	0.7500
2	12	4	8	0.6000
3	14	10	6	0.7000
4	11	12	9	0.5500
5	18	3	2	0.9000
6	12	5	8	0.6000
7	13	7	7	0.6500
Final Recall				0.679

For Model 2:

Label	TP	FP	FN	Recall
1	19	0	1	0.9500
2	20	3	0	1.0000
3	18	1	2	0.9000
4	17	4	3	0.8500
5	17	3	3	0.8500
6	16	1	4	0.8000
7	18	3	2	0.9000
Final Recall				0.893

5. Calculate the F1 score of each model.

   The F1 score is defined as:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

   The results for each model are:

$$\text{F1}_1 = 0.683, \quad \text{F1}_2 = 0.894.$$

Exercise 4 (Stable Matching for Reviewers and Papers)**.**

In this exercise, we will use the Gale-Shapley algorithm to find a stable matching between a set of reviewers (or researchers) and research papers. Each researcher has a list of preferences over the papers, and each paper has a list of preferences over the researchers. Preferences are expressed as scores ranging from 0 to 1, where a higher score indicates a higher preference.

Let $R = \{R_1, R_2, R_3, R_4, R_5\}$ denote the set of researchers and $P = \{P_1, P_2, P_3, P_4, P_5\}$ denote the set of papers. The preferences of each researcher R_i for each paper P_j are represented by a score $S_{ij} \in [0, 1]$. Similarly, the preferences of each paper P_j for each researcher R_i are represented by a score $\tilde{S}_{ji} \in [0, 1]$.

Below are example preference scores for researchers and papers:

- **Preferences of Researchers for Papers**

Researcher	P_1	P_2	P_3	P_4	P_5
R_1	0.9	0.7	0.8	0.6	0.5
R_2	0.6	0.9	0.5	0.8	0.7
R_3	0.7	0.6	0.9	0.5	0.8
R_4	0.8	0.7	0.6	0.9	0.5
R_5	0.5	0.8	0.7	0.6	0.9

- **Preferences of Papers for Researchers**

Paper	R_1	R_2	R_3	R_4	R_5
P_1	0.8	0.9	0.7	0.6	0.5
P_2	0.5	0.8	0.9	0.7	0.6
P_3	0.9	0.6	0.8	0.5	0.7
P_4	0.6	0.7	0.5	0.9	0.8
P_5	0.7	0.5	0.8	0.6	0.9

1. Define instability in the context of stable matching. Does the Gale-Shapley algorithm always result in a stable matching? Explain.

   Definition of Instability: In the context of stable matching, an *instability* occurs when there exists a pair (R_i, P_j) such that:

   - Researcher R_i prefers paper P_j over their currently assigned paper, and
   - Paper P_j prefers researcher R_i over their currently assigned researcher.

   Does the Gale-Shapley Algorithm Always Result in a Stable Matching? Yes, the Gale-Shapley algorithm always results in a stable matching.

2. Derive the rankings for each researcher over papers and for each paper over researchers based on the given preference scores.

   The rankings are derived by sorting the preference scores in descending order for each researcher and each paper. **Rankings of Researchers over Papers:**

Researcher	Ranking of Papers
R_1	$P_1 > P_3 > P_2 > P_4 > P_5$
R_2	$P_2 > P_4 > P_5 > P_1 > P_3$
R_3	$P_3 > P_5 > P_1 > P_2 > P_4$
R_4	$P_4 > P_1 > P_2 > P_3 > P_5$
R_5	$P_5 > P_2 > P_3 > P_4 > P_1$

Rankings of Papers over Researchers:

Paper	Ranking of Researchers
P_1	$R_2 > R_1 > R_3 > R_4 > R_5$
P_2	$R_3 > R_2 > R_4 > R_5 > R_1$
P_3	$R_1 > R_3 > R_5 > R_2 > R_4$
P_4	$R_4 > R_5 > R_2 > R_1 > R_3$
P_5	$R_5 > R_3 > R_1 > R_4 > R_2$

3. Complete the blanks in Algorithm 10:

Algorithm 9 Gale-Shapley Algorithm

Input: Lists of preferences (researchers, papers)
Output: Stable matching

All researchers start as free
while ∃ free researcher R who hasn't proposed to all papers **do**
  Pick such a researcher R
  Let P be the next paper on R's list
  if P is free **then**
    ... ▷ Fill in the blank
  else if P prefers R to current researcher R' **then**
    ... ▷ Fill in the blank
  else
    ... ▷ Fill in the blank
  end if
end while

Algorithm 10 Gale-Shapley Algorithm

Input: Lists of preferences (researchers, papers)
Output: Stable matching

All researchers start as free
while ∃ free researcher R who hasn't proposed to all papers **do**
  Pick such a researcher R
  Let P be the next paper on R's list
  if P is free **then**
    Assign R to P
  else if P prefers R to current researcher R' **then**
    Assign R to P, and make R' free
  else
    R remains free and moves to the next paper on their list
  end if
end while

4. Apply the Gale-Shapley algorithm to the provided preferences and derive the final matching as pairs {(Researcher, Paper)}.

   The final stable matching using the Gale-Shapley algorithm is:

$$\{(R_1, P_1), (R_2, P_2), (R_3, P_3), (R_4, P_4), (R_5, P_5)\}.$$

5. Discuss the optimality of the resulting couples compared to other valid stable matchings. Is the solution optimal for both researchers and papers? Why or why not?

6. Discuss the optimality of the resulting couples compared to other valid stable matchings. Is the solution optimal for both researchers and papers? Why or why not?

   The Gale-Shapley algorithm produces a stable matching, but the optimality depends on which side of the matching is making the proposals. In this case, researchers are the proposing side, so the resulting matching is optimal for researchers. Each researcher is matched with their best valid partner among all stable matchings. Papers are the receiving side, they receive the least preferred valid partner among all stable matchings.