Report of models’ findings and takeaways in Big Data Mining Midterm Project

Ta Cong Thuan

Nguyen Hoang Kien

Vi Nguyen Ngoc Chau

## 

# Table of Contents

1. Decision Tree & Random Forest
2. K-means
3. Linear Regression
4. Logistic Regression
5. Matrix Factorization
6. [Naïve](https://docs.google.com/document/d/1vkClbJjCRK17i0pOjtVv308D41o6PbnUqfpyMe5jSYA/edit?pli=1#heading=h.47l7i2sxf84m) Bayes

# 

## 

1. **Decision Tree & Random Forest**
   1. Decision Tree
      1. *Tasks*

* **Initialization** (\_\_init\_\_): Complete attribute initialization.
* **Node Initialization**: Define threshold and feature.
* Entropy & IG
* **Rule Finding**: Generate potential split thresholds for each feature.
* **Split Selection** (next\_split):
  + Find features and thresholds using information gain.
  + Implement feature and threshold selection strategies for Random Forest.
* **Tree Building** (build\_tree):
  + Construct the root node and handle base cases for stopping recursion.
  + Split data for left and right subtrees.
* **Fitting** (fit): Build the tree using training data.
* **Individual Prediction** (ind\_predict): Traverse the tree to predict a label.
* **Batch Prediction** (predict): Call ind\_predict for each instance.
  + 1. *Results & Findings*  
       Train Error = 1.0 -> Perfectly classified all the training instances.  
       Test Error ~= 0.621875 -> Captured some patterns in the wine quality data.
* Tuning like the maximum depth of the tree, minimum samples allowed for split, or number of features at each split.
* Data exploration: It's helpful to explore the data (e.g., distribution of features, presence of outliers) to identify potential issues that could affect the model's performance.
  1. Random Forest
     1. *Tasks*
* **Initialization** (\_\_init\_\_): Complete attribute initialization.
* **Training** (fit):
  + Create n\_trees instances with appropriate hyperparameters.
  + Train each tree on the training data.
* **Individual Prediction** (ind\_predict):
  + Obtain predictions from all trees for the given instance.
  + Determine the final prediction using majority voting (random)
* **Batch Prediction**: Call ind\_predict for each instance in the batch.
  + 1. *Results & Findings*

**Test Accuracy** = 0.709375 (Greater than 0.69) -> capture some underlying patterns in the wine quality data and generalize better to unseen data.

* Experimenting with different numbers of trees could potentially improve the performance.
* max\_depth and n\_features that you can tune to potentially improve the test accuracy.

1. **K-means**
   1. K-means
      1. *Tasks*

* **Initialization**: Sets up the number of clusters (k), training iterations, and randomly assigns cluster centers from the data.
* **Training** (fit): Iteratively assigns data points to the closest cluster based on distance, then recalculates the cluster centers based on their assigned points. This repeats until the clusters stabilize or a maximum number of iterations is reached.
* **Individual Prediction**: Assigns a new data point to the closest cluster based on distance calculations.
* **Batch Prediction**: Efficiently assigns multiple new data points to clusters by potentially vectorizing the distance calculations.
  + 1. *Result and Finding*

Different results every time run K Means (centers are randomly initialized.

* 1. Gaussian
     1. *Task*
* **Initialization**: KMeans sets initial means, mixing probabilities are equal, and covariance matrices are identity.
* **Training** (EM loop):
  + E-Step: Calculates data point probabilities for each cluster.
  + M-Step: Updates GMM parameters based on probabilities.
  + Likelihood: Monitors model fit during training.
* **Individual Prediction**:
  + To predict the cluster for a new data point:
  + Calculate the probability of the data point belonging to each cluster using the Gaussian function and mixing probabilities.
  + Assign the data point to the cluster with the highest probability.
* **Batch Prediction**:
  + Similar to individual prediction, but done for multiple new data points at once.
  + Can be optimized for efficiency by vectorizing calculations for all data points.
    1. *Result and Finding*

-> Both custom and scikit-learn GMM implementations produced **similar cluster centers** and scores, despite **minor** variations. This suggests they captured the data structure well. Slight differences might stem from initialization, numerical precision, or optimization details in the EM algorithm.

1. [**Linear Regression**](https://docs.google.com/document/d/1vkClbJjCRK17i0pOjtVv308D41o6PbnUqfpyMe5jSYA/edit?pli=1#heading=h.9m3t8rgu7rx0)
   1. *Tasks*

* **Initialization**: Sets learning rate, iterations, early stopping, and intercept usage. Optionally allows initializing weights.
* **Training**: Stores training data, handles intercept, initializes weights, and calls gradient descent for training.
* **Gradient Descent**: Iterates, calculates gradient, updates weights, implements early stopping, adjusts learning rate, tracks loss, and returns the model.
* **Prediction**: Takes new data, handles intercept for consistency, calculates predictions, and returns them.
  1. *Result & Findings*

Here's a breakdown of the situation different coefficients (the loss is very close to each other like 805)

* Test 1: Used a custom linear regression implementation with coefficients close to [19.7472888 30.00037853] [20, 30]. Achieved a loss not shown explicitly but likely similar to Test 2 based on the provided information.
* Test 2: Used scikit-learn's LinearRegression with coefficients close to [-0.18, 2.05, 0.36]. Achieved a loss of 0.25.

-> In linear regression, with a high-dimensional feature space (multiple features), the loss function can have multiple local minima. These minima represent different models with varying coefficients that can achieve similar overall loss on the training data. In this case, both models likely captured the underlying linear relationship between wine density, alcohol content, and quality. Despite having different weightings (coefficients) for each feature, they achieved comparable performance on the training data as measured by the loss function.

1. **Logistic Regression**
   1. *Tasks*

* Initialization: Stores training data and sets weights/bias to zeros.
* Gradient Calculation: Calculates gradients for weights and bias using sigmoid and error.
* Gradient Descent: Iterates, updates weights/bias with learning rate, implements early stopping.
* Prediction: Standardized test data if needed, calculates individual probabilities using weights and features, returns all predictions.
  1. *Result & Findings*
* count : 774/1319~=0.58
* seems to have achieved a decent number of correct classifications on the training data (774). However, this metric is unreliable because the model was trained on the same data it's being evaluated on.
* To assess the model's true performance, techniques like k-fold cross-validation or a separate test set are necessary.

1. [**Matrix Factorization**](https://docs.google.com/document/d/1vkClbJjCRK17i0pOjtVv308D41o6PbnUqfpyMe5jSYA/edit?pli=1#heading=h.ooihd3ix3wc1)
   1. Matrix Factorization
      1. *Task*

* **Initialization**: Sets up learning rate, training iterations, and other factors to control the model's behavior.
* **Training**: Learns from user ratings (user, movie, rating). It creates two key factors (P and Q) and refines them based on the differences between predicted ratings and actual ratings.
* **Individual Prediction**: Predicts a rating for a single new user-movie combination that wasn't included in the training data.
* **Batch Prediction**: Predicts ratings for a list of new user-movie combinations efficiently.
  + 1. *Result and Findings*

Result:

CPU times: user 2min 36s, sys: 628 ms, total: 2min 37s

Wall time: 2min 39s >< Wall time: 997 µs

→ The execution time in the second case (2min 39s) is roughly 1,600,000 times slower than the first case (997 µs).

(error/ len(test))\*\* (0.5): 1.4001861948855225 >< provided time 2.5386249744910048

→ A lower RMSE (like 1.400...) suggests better prediction accuracy on a dataset

* 1. Matrix Factorization with bias
     1. *Task*
* Initialization: Sets up learning rate, training iterations, and other factors to control the model's behavior.
* Training: Learns from user ratings (user, movie, rating) considering user bias, movie bias, and a global average rating. It refines factors (P, Q) capturing user and movie preferences, along with user and movie biases.
* Individual Prediction: Predicts a rating for a single new user-movie combination that wasn't included in the training data, considering biases.
* Batch Prediction: Predicts ratings for a list of new user-movie combinations efficiently, accounting for biases.
  + 1. *Result and Findings*

Result:

CPU times: user 4min 23s, sys: 13.3 s, total: 4min 36s

Wall time: 4min 29s >< Wall time: 0 ns

→ + The first process executes much slower than the second.

* The 0 ns time for the second process is likely inaccurate and doesn't reflect actual execution time. It might be a placeholder or an error.

(error/ len(test))\*\* (0.5): nan ><2.5386249744910048

—> not finding the RMSE

1. [**Naïve Bayes**](https://docs.google.com/document/d/1vkClbJjCRK17i0pOjtVv308D41o6PbnUqfpyMe5jSYA/edit?pli=1#heading=h.47l7i2sxf84m)
   1. *Tasks*

* **Training**: Calculates class probabilities (prior) and feature probabilities (likelihood) for efficient classification.
* **Prediction**:
  + Consider all class labels and their probabilities based on the feature vector.
  + Chooses the class with the highest probability (considering both prior and likelihood).
  + Handles small probabilities (potentially using logarithms) and unknown feature values (with chosen strategy).
* **Predicts for multiple instances** by applying the single-instance prediction logic to each data point.
  1. *Result & Findings*

Overall Accuracy: ~0.884 -> can effectively learn the patterns within the data and make accurate predictions for new instances.

* The accuracy of 88.4% is on the test set, which is a good indicator of how the model might perform on unseen data. However, it's important to be cautious about generalizing this performance to other datasets or real-world scenarios.
* The code doesn't explore hyperparameter tuning or feature engineering techniques. These approaches could potentially improve the model's performance.
* It's not shown how the model handles unknown feature values. The implementation might need a strategy for dealing with unseen feature values during prediction.

## 