Midterm

Abstract

In this task, I apply the **K-Nearest Neighbors (KNN)** algorithm, refining several aspects to improve accuracy. This includes thoughtful feature selection, weighted scoring, and systematic preprocessing strategies.

Feature Choice

The training data contains the following features (excluding the target "score"): `ProductId`, `UserId`, `HelpfulnessNumerator`, `HelpfulnessDenominator`, `Time`, `Summary`, `Text`, and `Id`.

1. **Feature Removal:** The `Id` field is dropped since it only serves as a unique identifier for reviews and offers no meaningful information for prediction.
2. **Helpfulness Ratio (Helpness):** A new feature, **Helpness = HelpfulnessNumerator / HelpfulnessDenominator**, is added. Reviews with similar Helpness scores tend to have similar overall ratings. This feature is also used as a **weight factor** in the KNN algorithm to prioritize reviews deemed more helpful. Reviews with higher Helpness contribute more to the prediction.
3. **Time as a Feature:** Time is considered relevant since reviews posted within the same period may reflect similar opinions or trends.
4. **Product and User Preferences:** Review patterns suggest that users tend to score products consistently. Therefore, both `ProductId` and `UserId` are included. However, since these features are identifiers, they are used to **filter neighbors** rather than calculate distances directly. The algorithm selects neighbors with matching `ProductId` first. If sufficient matches are unavailable, it expands to neighbors with matching `UserId`, and finally to other reviews if needed.
5. **Textual Features (Summary and Text):** Although extracting meaningful features from review text is essential (e.g., "good" being closer to "excellent" than to "bad"), it poses challenges with complex expressions (e.g., "not good at all" vs. "not bad"). As deep learning is not applied in this task, keyword-based distance calculation will be explored in future work.

Algorithm Details

Feature Preprocessing:

- Both **Helpness** and **Time** are normalized to prevent large numerical ranges from disproportionately influencing the model’s predictions. Without normalization, the algorithm would heavily favor time-based similarities.

Choosing K:

- Based on experimental results, **k = 38** yields the best performance. However, increasing k beyond this value negatively impacts accuracy, especially in datasets dominated by scores of 5.0. This reflects a **severe bias** in the data distribution. Selecting too many neighbors increases the chance of including multiple 5.0 scores, reducing prediction accuracy for reviews with lower scores.

Weights:

- The weighted score is computed as:**weighted_score = (normalized_helpfulness × 2.5) + 1**
    - Even if a review has no helpfulness votes (possibly due to being newly posted), it still contributes to the prediction.

Results

The experiments were conducted under consistent conditions, varying only the **data size** and **k** values. 1/4 of each dataset was used as the test set.

Datasize \ k	18	28	38	48	68
6,000	0.520	0.520	0.520	–	–
14,000	–	–	0.535	0.530	0.530
28,000	–	–	0.535	0.530	0.530

Effect of ProductId and UserId on Accuracy

This experiment examines whether filtering by ProductId and UserId affects the accuracy.

Size	**Use Helpfulness**	**Matched ProductId (avg)**	**Accuracy**	**Matched both ProductId & UserId?**
8,000	No	0.809	0.516	No
8,000	No	0.809	0.526	ProductId
32,000	No	3.07	0.527	ProductId
32,000	No	3.07	0.533	ProductId & UserId

Impact of MinMax Scaling

This experiment compares the effect of using MinMax scaling on accuracy.

Size	MinMax Scaling	Accuracy
8,000	Yes	0.518
8,000	No	0.516

Conclusion

This task demonstrates the importance of feature engineering, preprocessing, and appropriate neighbor selection when applying the KNN algorithm to biased datasets.