Group Information

- **Group:** 89
- **Module:** 3
- **Name:** Xinyan Liu, No Personal Number, Bilateral Exchange Agreements Program
- **Email:** `liuxinya@chalmers.se`
- **Allowed to be finished by one person:** Yes

Declaration

I hereby declare that I have actively participated in solving every exercise. All solutions are entirely my own work, without having copied or referred to other solutions.

1 Reading and reflection

The paper *Hidden Technical Debt in Machine Learning Systems* discusses how machine learning systems, while powerful and quick to develop, often accumulate significant hidden technical debt, making their long-term maintenance challenging and expensive. The authors draw parallels between software engineering's concept of technical debt and the unique challenges posed by ML systems. They argue that ML systems inherently face more risks due to their reliance on data, dynamic behaviors, and integration into complex environments. Key issues explored include boundary erosion, where ML systems blur traditional software abstractions; entanglement, where changes to one feature or component inadvertently impact others; and the challenge of managing data dependencies, which are harder to track and stabilize than code dependencies. The paper also identifies anti-patterns like "glue code" (excessive supporting code to integrate ML libraries) and "pipeline jungles" (overly complicated data preprocessing flows), both of which increase system rigidity and hinder scalability. Additionally, the authors highlight risks from feedback loops, undeclared consumers, configuration issues, and external world changes, all of which can degrade system reliability if not addressed.

One major takeaway is the importance of designing ML systems with maintainability in mind, rather than prioritizing short-term performance improvements. The authors suggest strategies like monitoring data dependencies, establishing robust abstractions, and fostering a culture that values simplification and maintainability alongside accuracy. Another critical insight is that ML systems are not just about code; they require substantial infrastructure, testing, and configuration to handle changes in the external world, such as distribution shifts in data.

Another takeaway is that technical debt in ML systems is less about the models themselves and more about the surrounding processes and infrastructure. This paper emphasizes the need for a balance between research and engineering, as well as the importance of building tools and abstractions to manage complexity. It's a reminder that long-term success in ML requires a focus on system-level robustness, not just algorithmic performance.

2 Implementation

The implementation involves a custom classifier that utilizes the KMeans clustering algorithm for classification tasks. The classifier is trained on labeled data from Beijing and Shenyang, and tested on labeled data from Guangzhou and Shanghai. The goal is to predict the binary label `PM_HIGH` based on the input features.

2.1 Code Workflow

The code follows these steps:

1. Load labeled datasets from four cities: Beijing, Shenyang, Guangzhou, and Shanghai.
2. Combine Beijing and Shenyang data as the training set, and Guangzhou and Shanghai data as the test set.
3. Initialize a custom `Classifier` with 2 clusters.
4. Train the classifier on the training set.
5. Validate the classifier on the training set and compute accuracy.
6. Test the classifier on the test set and compute accuracy.
7. Print the training and test accuracy.

2.2 Class and Functions

2.2.1 `__init__`

- Initializes the classifier with a specified number of clusters (default is 2).
- Sets up a KMeans model and a placeholder for centroid labels.

2.2.2 `fit`

- Fits the KMeans model to the training data `X`.
- Assigns labels to each cluster centroid based on the majority label in the training data `y`.

2.2.3 `predict`

- Predicts the cluster assignment for new data points using the trained KMeans model.
- Assigns the label corresponding to the centroid of the inferred cluster.

2.2.4 `score`

- Computes the accuracy score by comparing the true labels `y_true` with the predicted labels `y_pred`.

2.3 Data Preparation

- The training set is created by combining Beijing and Shenyang data.
- The test sets are Guangzhou and Shanghai data.
- Features are extracted by dropping the `PM_HIGH` column, and labels are extracted from the `PM_HIGH` column.

2.4 Training and Testing

- The classifier is trained on the training set using the `fit` method.
- Training accuracy is computed by predicting labels for the training set and comparing them with the true labels, which is the same as validation accuracy, as we do not split training set into training set and validation set
- Test accuracy is computed by predicting labels for the test set and comparing them with the true labels.

2.5 Evaluation of the System

2.5.1 Training Accuracy (Beijing and Shenyang)

Training Accuracy: 0.7250. A training accuracy of 73% indicates that the classifier correctly predicted the labels for 73% of the training data points. This level of accuracy suggests that the KMeans-based classifier is capable of capturing some underlying patterns in the training data. However, it also highlights that there is room for improvement, as the inherent limitations of KMeans in handling supervised classification tasks and potential overlap or noise in the feature space may hinder optimal class separation.

2.6 Validation Accuracy (Beijing and Shenyang)

Validation Accuracy: 0.7250. Since the training data is directly used to evaluate the model, the training accuracy also serves as an implicit validation metric for the Beijing and Shenyang datasets. A validation accuracy of 73% indicates that while the model fits the data reasonably well, it may not fully capture the underlying structure, suggesting that further refinement might be needed to better separate overlapping classes.

2.6.1 Test Accuracy (Separately on Guangzhou and Shanghai)

Test Accuracy on Guangzhou: 0.9364 and **Test Accuracy on Shanghai: 0.9016**. Evaluating the classifier separately on each city reveals that the model achieves a high accuracy on both, with a slightly better performance on Guangzhou (93.64%) compared to Shanghai (90.16%). These high test accuracies indicate that the classifier generalizes exceptionally well to the unseen data from these regions, suggesting that the feature distributions in Guangzhou and Shanghai may be more distinct or less noisy than those in the training data.

```
Train Accuracy (Beijing and Shenyang): 0.7250
Test Accuracy Guang Zhou: 0.9364
Test Accuracy Shanghai: 0.9016
```

3 Discussion

3.1 Observed Results

- **Training and Validtion Accuracy** (Beijing and Shenyang): 72.50%
- **Test Accuracy on Guangzhou**: 93.64%
- **Test Accuracy on Shanghai**: 90.16%

It is counterintuitive to see a higher accuracy on unseen test data compared to the training data, which is different from most cases.

3.2 Possible Reasons for Higher Test Accuracy

Several factors could explain why test accuracy exceeds training accuracy:

a) **Differences in Data Distributions:** For example, the training data, collected from Beijing and Shenyang, might be inherently more noisy or exhibit greater overlap between classes. In contrast, the test data from Guangzhou and Shanghai may feature clearer separations, making classification easier. .

b) **Usage in KMeans:** The KMeans algorithm is designed for clustering rather than classification. Its application here involves assigning labels based on the majority vote within clusters. This method

inherently assumes that the data naturally separates into two compact and well-distinguished clusters. If this assumption does not fully hold in the training data due to higher noise or overlapping class distributions, the training accuracy can be lower. On the other hand, if the test data better satisfies these assumptions, this can manifest as a higher accuracy.

3.3 Assumptions During Development

During the model development process, several assumptions were implicitly made:

- **Data is Naturally Clustered into Two Distinct Groups:** We assumed that the underlying data could be partitioned into two distinct clusters that reflect the two classes.
- **Similarity in Data Distributions:** It was assumed that the training and test data share similar statistical properties such that performance on the training data would be indicative of performance during deployment.
- **Clean and Well-Separated Data:** The method of assigning labels via majority voting in each cluster presumes that clusters are relatively clean and that the majority label is representative of the entire cluster.

In practice, these assumptions may not hold. The training data might have more noise, feature overlap, or other complexities that the model could not capture, whereas the test data might have a simpler or more idealized structure.

3.4 Potential Improvements

To address the observed discrepancies and to further improve classifier performance, several strategies could be considered:

- **Enhanced Feature Engineering:** Conduct more rigorous feature selection and normalization to reduce noise and improve class separability. Advanced techniques such as dimensionality reduction or feature scaling methods (e.g., PCA) could be useful.
- **Model Refinement:** Explore alternative clustering methods or hybrid models that can better handle the supervised task. For example, integrating additional classifiers (e.g., SVM or Random Forest) may bolster performance.
- **Balancing Data Distributions:** Collect more representative training data or perform data augmentation to ensure that the training set better reflects the feature distributions observed in the test sets.
- **Parameter Tuning:** Adjust the parameters of the KMeans algorithm (e.g., initialization methods, number of iterations) to improve its fit to the training data.

Implementing these measures could potentially increase the training accuracy while maintaining or even improving the generalization performance on unseen data.

3.5 Conclusion

While it is typical for a deployed system to exhibit a lower accuracy than what is observed during training, our system indicates a reverse trend where testing on Guangzhou and Shanghai yields higher accuracy than training on Beijing and Shenyang. The discrepancy appears to be driven by differences in data distribution, sampling bias, and inherent limitations of using KMeans for a classification task. Addressing these issues through improved feature engineering, model refinement, and better data management could bridge the gap between training and test performance, thus reinforcing the reliability and robustness of the system.