Meta-Learning for Coreset Selection: Identifying Optimal Techniques Across Diverse Datasets

Colin Ward
Department of Computer Science
Wake Forest University
Winston-Salem, NC 27109
wardcj21@wfu.edu

Ahmed Ali
Department of Computer Science
Wake Forest University
Winston-Salem, NC 27109
aliae21@wfu.edu

Abstract

This project conducts a comprehensive empirical evaluation of various coreset selection techniques across 76 datasets spanning diverse domains, including finance, healthcare, and technology. By extracting detailed metadata from classification (binary and multi-class) and regression tasks, we developed a meta-model to predict the optimal coreset selection method for a given dataset. Our methodology involves rigorous preprocessing, feature extraction, and the application of multiple coreset techniques, coupled with robust evaluation metrics like ROC AUC and MSE. We identified distinct performance trends, and insights. Leveraging decision trees and Random Forest models, we achieved interpretable results and significant improvements in predictive accuracy through data augmentation. This work bridges theoretical advances with practical applications, offering actionable insights and a scalable framework for optimizing coreset selection strategies tailored to specific data characteristics and analytical goals.

1 Introduction

The exponential growth of data across various domains has posed significant challenges in data processing and analysis. For instance, ten years ago, the New York Stock Exchange (NYSE) generated approximately 5 terabytes of data daily, a volume that has since expanded with the inclusion of various financial instruments such as fixed-income securities, derivatives, and ETFs. Processing such vast amounts of data necessitates substantial computational resources, making it imperative to develop efficient data reduction techniques.

Coreset selection emerges as a pivotal strategy to mitigate these challenges by reducing dataset sizes while preserving their essential characteristics. Introduced by Agarwal in 2005, coreset selection has gained traction in large-scale data analysis, particularly within machine learning and artificial intelligence domains. The primary objective is to create a smaller, representative subset of the original data that maintains its integrity, thereby optimizing computational efficiency without compromising model performance.

Originally focused on financial datasets, our project has expanded to encompass a diverse range of datasets across multiple domains, including healthcare, retail, technology, and more. This expansion aims to evaluate the generalizability and robustness of various coreset selection methods in different contexts. By analyzing a broad spectrum of datasets, we seek to develop a meta-model that predicts the most effective coreset selection technique based on dataset features and specific task requirements, whether classification or regression. This comprehensive approach ensures that our findings are applicable to a wide array of real-world scenarios, enhancing the utility and scalability of coreset selection in diverse data analysis tasks.

1.1 Problem Statement

Despite the theoretical advancements in coreset selection techniques, there remains a significant knowledge gap regarding which methods are most effective based on specific data characteristics. Practitioners often face uncertainty in selecting the appropriate coreset strategy for their unique datasets and analytical objectives. This lack of empirical guidance hinders the optimal utilization of coreset methods, potentially leading to suboptimal model performance or inefficient use of computational resources.

1.2 Motivation

Our motivation for this project stems from our desire to help bridge the knowledge gap of when different coreset selection techniques are most effective. We believe that through our systematic evaluation of these techniques on 75+ datasets from multiple domains, we have generated comprehensive meta-data for our meta-model. This meta-model will provide valuable insights into how practitioners can best use coreset selection strategies in their problems.

1.3 Challenges

Conducting this extensive evaluation presents several challenges. Managing and preprocessing a large number of diverse datasets requires robust and scalable engineering solutions to ensure consistency and comparability across experiments. Additionally, the computational demands of applying multiple coreset selection methods to extensive datasets require efficient algorithm implementations. Developing a meta-model that accurately correlates dataset features with coreset effectiveness involves intricate machine learning tasks, including feature engineering and model validation. Lastly, ensuring reproducibility and scalability of experiments across numerous datasets and methods demands careful engineering and documentation.

2 Related work

Coreset selection techniques have been extensively studied as a means to reduce large datasets while preserving essential characteristics for model training. Various methods have been proposed, focusing on theoretical and empirical insights.

The foundational work by Feldman et al. (2020) introduced sensitivity sampling as a framework for coreset construction, which has seen widespread use due to its versatility across clustering objectives and beyond. Sensitivity sampling selects points proportional to their importance, which provides a robust mechanism to handle well-clusterable datasets (Bansal et al., 2024). This approach has inspired significant advances in coreset methodology, particularly for k-means and related clustering problems.

Yang et al. (2024) proposed decision boundary reconstruction, which focuses on selecting data points nearest to the decision boundary, ensuring the model is trained on the most fragile data points. This method emphasizes the importance of fragile regions in data representation, a concept that is complementary to clustering-based techniques such as those introduced by Chai et al. (2023), which select coresets by analyzing distances within clusters to represent diverse data regions effectively.

TAGCOS, introduced by Zhang et al. (2024), extends this idea by leveraging gradient clustering for instruction tuning datasets. TAGCOS uses sample gradients to group data into clusters, then applies a greedy algorithm for efficient coreset selection, achieving near-optimal performance on a small subset of the data.

Another recent method, STAFF, proposed by Zhang et al. (2024), focuses on task-specific fine-tuning of large language models (LLMs). STAFF uses speculative execution to efficiently allocate selection budgets, outperforming other methods such as GraNd and EL2N in terms of accuracy and selection overhead, especially at high pruning rates.

Reservoir sampling, introduced by Vitter (1985), provides a practical and computationally efficient method for random sampling when the dataset size is unknown. This method remains particularly relevant for streaming data applications and serves as a baseline for randomized selection strategies.

Cochran's (1946) comparative analysis of systematic and stratified random sampling underscores the effectiveness of stratified methods, particularly for datasets with specific variance structures. This foundational work informs many modern coreset sampling techniques that aim to balance computational simplicity with representativeness.

Active learning literature, surveyed extensively by Settles (2009), highlights the importance of uncertainty sampling and data diversity. These principles underpin many coreset strategies, particularly for tasks requiring high accuracy with limited data.

Bachem et al. (2017) explored practical coreset constructions using importance sampling, focusing on scalable techniques for k-means, PCA, and other machine learning problems. Their work highlights the scalability of coresets, achieving sublinear sizes while maintaining theoretical performance guarantees.

Feldman's 2020 survey on coreset selection contrasts the theoretical benefits of various techniques, emphasizing the need for empirical evaluations. Despite these theoretical advances, many studies lack systematic experimental validation across diverse datasets, particularly in high-dimensional domains like finance.

While the advancements in coreset selection demonstrate significant progress, there remains a lack of empirical studies that systematically evaluate these methods across diverse datasets. Most existing works focus on theoretical guarantees or task-specific optimizations, often overlooking generalizability and practical guidance for practitioners dealing with datasets that exhibit unique challenges, such as imbalanced classes, mixed feature types, or high dimensionality.

Our project aims to address this gap by conducting a comprehensive empirical evaluation of various coreset selection techniques. By exploring both classification and regression tasks, we seek to identify not only the most effective methods but also the specific dataset characteristics that influence their performance. Furthermore, the development of a meta-model to recommend optimal coreset techniques based on dataset features differentiates our work from prior studies. This bridges the divide between theoretical advances and real-world applicability.

3 Methodology

The first step in our project's pipeline is to collect our metadata on the given dataset and task. Each dataset and associated model will serve as one datapoint in our metadata. The features included in the metadata are:

- dataset_name: Name of the dataset (e.g., creditcard.csv, bankrupcy.csv).
- task_type: Type of task associated with the dataset, either Binary Classification or Regression.
- num_instances: Number of instances (rows) in the dataset.
- num_features: Total number of features (columns) in the dataset.
- num_numerical_features: Number of numerical features.
- num_categorical_features: Number of categorical features.
- feature_type: Indicates the type of features in the dataset (e.g., Numerical, Mixed).
- num_classes: Number of classes in classification tasks (applicable for classification datasets only, empty for regression).
- class_balance: A dictionary indicating the class distribution for classification datasets. Empty for regression
- imbalance_ratio: Ratio of instances between the most frequent and least frequent classes. Empty for regression
- dimensionality: Calculated as num_features / num_instances.
- mean_correlation & max_correlation: Measures of correlation between features.
- feature_redundancy: Measure of feature redundancy in the dataset.
- Statistical moments (mean_of_means, variance_of_means, mean_of_variances, variance_of_variances): Mean and variance of feature means and variances across features.

- Skewness (mean_skewness) & Kurtosis (mean_kurtosis): Higher-order statistical descriptors of feature distributions.
- outlier_percentage: Proportion of data points classified as outliers.
- data_sparsity: Proportion of missing or sparse data points in the dataset.

Next, we apply all coreset selection techniques. We use each of them on a Logistic Regression model for classification and a Linear Regression model for regression. We chose to use simple models for their ability to apply to a wide array of problems. Then, we select the technique that yields the highest accuracy. For accuracy, we use the ROC AUC for our classification datasets and MSE for our regression datasets. We collect these evaluation metrics along with some other unique metrics solely for further insights and comparison. And, based on the net gain in accuracy over the baseline, we select which coreset selection technique was best. This then becomes our output label for the metadata. A brief description of the label:

- best_coreset_method: The target variable, indicating the best coreset selection method for the dataset based on evaluation. This is the only The full list of classes are: clustering, gradient, importance, kmeans, random, reservoir, stratified (only for classification), uncertainty (only for classification), leverage (only for regression), farthest (only for regression)

After constructing our metadata across or three scripts for binary classification, regression, and multi-class classification, with various datasets and associated models, we train a model on the metadata to select the optimal coreset selection technique. As such, a user could upload their dataset and the task they intend to perform, and then our meta-model would recommend the optimal coreset selection technique. We also implemented evaluation through stratified splits on the meta-model. To achieve higher accuracy, we performed data augmentation on the metadata to increase the number of instances. For this we built a separate model. These two models implemented Random Forest Classification.

Further, recall that our project aims to create a tool that chooses the best coreset selection technique and figure out why a particular method is optimal. This is why we also implemented a meta-model that uses a Decision Tree. A decision tree most closely mimics the logic of a human brain, and thus, our results are more interpretable relative to the other versions. The explainability of our results is just as important as the recommendation.

4 Implementation and Experiments

The implementation of our project is structured around several key functions, each designed to execute specific tasks within the workflow. All of these functions are reused across our various scripts to ensure consistency. The functions also incorporate adjustments based on the script's task.

4.1 Data Preprocessing

The **preprocess_data** function serves as the initial step in preparing raw datasets for analysis. It systematically cleans and transforms raw data by identifying numerical and categorical columns, handling missing values by imputing means for numerical data and modes for categorical data, encoding categorical variables using One-Hot Encoding, and scaling numerical features using StandardScaler. Additionally, it processes the target variable. This function ensures that all datasets are uniformly processed, facilitating consistent model training and evaluation across different datasets.

4.2 Feature Extraction

The **extract_dataset_features** function is designed to analyze a given dataset and extract a comprehensive set of characteristics that can influence coreset selection methods. This is our key function that is used to collect metadata for the given dataset. It begins by capturing fundamental properties such as the dataset name, task type, number of instances, and number of features, distinguishing between numerical and categorical data types. The function evaluates class balance for binary classification tasks, calculating the number of classes and the imbalance ratio. It also assesses the dataset's dimensionality and examines the correlation among numerical features to identify redundancy. Additionally, the function computes statistical properties like means, variances, skewness, and kurtosis

of numerical features, detects the presence of outliers using the IQR method, and measures data sparsity by determining the proportion of zero elements. All these extracted features are organized into a dictionary, enabling the function to serve as reusable metadata across various scripts, thereby facilitating informed decision-making in coreset selection processes.

4.3 Coreset Selection Techniques

We have implemented 11 coreset selection functions (inlcuding a function that selects no coreset) using diverse sampling strategies:

- **no_coreset_selection:**
  Acts as a baseline by returning the entire training dataset without any sampling, allowing for direct comparison between models trained on the full dataset and those trained on selected subsets.
- **random_sampling_coreset:**
  Selects a random subset of the training data using NumPy's random choice, ensuring an unbiased sample from the entire dataset. Is only used for both classification tasks.
- **stratified_sampling_coreset:**
  Preserves class distribution by performing stratified sampling, selecting a subset of the data while maintaining the original class proportions, which is crucial for maintaining model performance in imbalanced datasets.
- **kmeans_clustering_coreset:**
  Utilizes K-Means clustering to partition the dataset into clusters a fraction of the dataset size, then selects one random sample from each cluster to form the coreset, ensuring comprehensive coverage of the feature space.
- **uncertainty_sampling_coreset:**
  Identifies and selects the top samples where the Logistic Regression model is least certain about the predictions, focusing on informative data points near the decision boundary to enhance model learning. It is only used for both classification tasks.
- **importance_sampling_coreset:**
  Assigns weights to samples based on their importance, calculated as the absolute difference between predicted probabilities and 0.5, and performs weighted sampling to prioritize more influential data points in the coreset. For regression, uses residuals as importance weights.
- **reservoir_sampling_coreset:**
  Implements the reservoir sampling algorithm to maintain an unbiased and random sample of the training data, suitable for large or streaming datasets due to its memory efficiency.
- **gradient_based_coreset:**
  Trains a Logistic Regression model to compute the log loss for each sample, then selects the top percentage of high-loss samples while ensuring class representation, thereby focusing on data points that significantly impact model learning. For regression, selects samples with highest squared error loss.
- **clustering_based_coreset:**
  Employs MiniBatchKMeans clustering to efficiently cluster large datasets and selects one random sample from each cluster, ensuring comprehensive coverage of the feature space while maintaining computational efficiency.
- **farthest_point_sampling_coreset:**
  Implements a coreset selection technique tailored for regression datasets by maximizing the diversity of the selected samples. It randomly selects an initial data point and iteratively adds the data point that is farthest from the current coreset based on Euclidean distance, ensuring that the chosen subset represents the entire dataset well. It is only used for regression tasks.
- **leverage_sampling_coreset:**
  Employs leverage scores to identify and select the most influential samples from a regression dataset. By calculating the hat matrix and extracting its diagonal elements, the function determines the leverage scores, which indicate the importance of each data point in the

feature space. It then selects the top samples with the highest leverage scores to form the coreset. It is only used for regression tasks.

4.4 Model Training and Evaluation

The **train_and_evaluate** function orchestrates the application of coreset selection methods, model training, and performance evaluation. It receives training and testing datasets along with the specified coreset selection method, applies the corresponding sampling strategy to obtain the coreset, and trains a Logistic Regression model on this subset (or Linear Regression). The function then predicts on the test set, computes evaluation metrics such as confusion matrix, classification report, ROC AUC score, precision, recall, and F1 score, and visualizes the ROC curve (Or MSE, MAE, R2 for regression). These metrics are added into a csv file for comparison across different coreset methods. This comprehensive evaluation enables a clear understanding of how each coreset strategy impacts model performance.

4.5 Experimental Workflow

The **main** functions serve as the orchestrator of the entire experimental process. They begins by extracting dataset features using the extract_dataset_features function and preprocessing the data through preprocess_data. The data is then split into training and testing sets with stratification to maintain class distributions. The function iterates through all defined coreset selection methods, applying each to the training data and evaluating the resultant model's performance using train_and_evaluate. Performance metrics from each method are stored in a results list. The main function identifies the best-performing coreset method based on the highest ROC AUC score for classification tasks and the lowest Mean Squared Error (MSE) for Regression tasks. It then compiles a comprehensive table combining dataset features with the best method's performance metrics. Additionally, the function allows for the generation of visualizations, including bar charts that compare the ROC AUC and MSE scores across different coreset methods, enhancing the interpretability of results. To activate this functionality set SHOW_PLOTS flag to true.

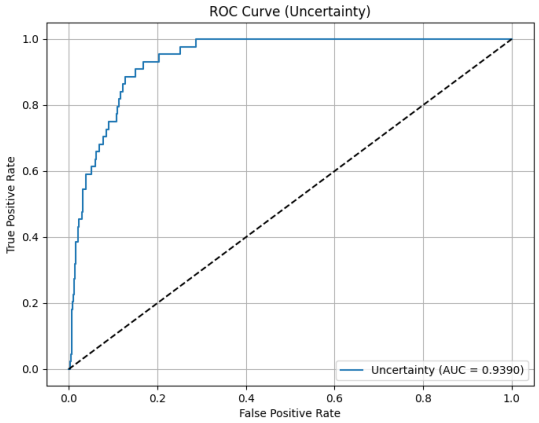

Figure 1: Shows an example performance of one classification task using ROC AUC

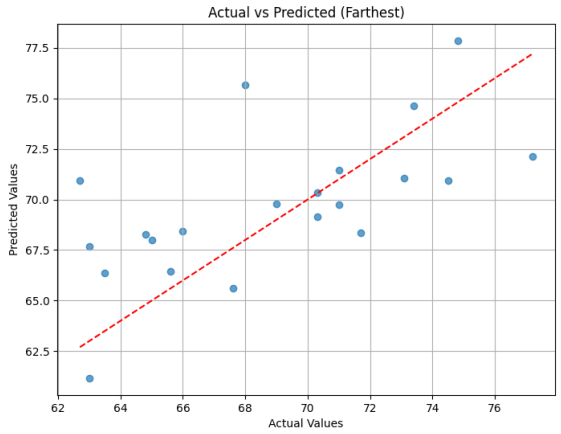

Figure 2: Shows an example performance of one regression task using MSE

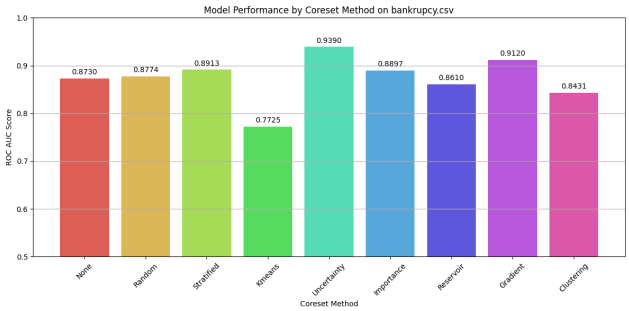

Figure 3: Shows an example of the summary of performances by Coreset Selection Method using ROC AUC

4.6 Experimentation Results

In total, we have collected 76 datasets, 32 of which were focused on the task of Binary Classification, 26 focused on Regression, and 18 on Multi-class classification. Of the 76, we reserved 10 for testing purposes and kept 66 for training. The resulting best coreset selection technique by task type are as follows:

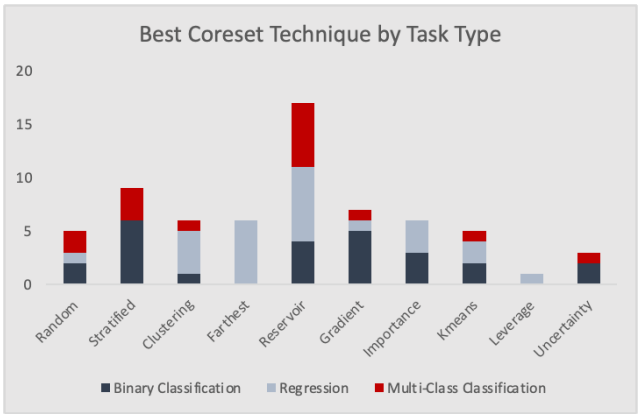

Figure 4: Distribution of coreset selection techniques

The figure shows that Reservoir was the most popular coreset selection method across the three task types. Farthest was popular for regression tasks, while Stratified was popular for Binary Classification tasks. Seeing these results are quite concerning especially given the uneven distribution of coreset selection methods. After experimenting with 76 datasets across our three task types, we understood the value of class imbalance. Still, we continued as outlined in the implementation pipeline and we let the numbers speak for themselves.

One version of our implementation made use of a Decision Tree model, and the resulting tree looks as follows:

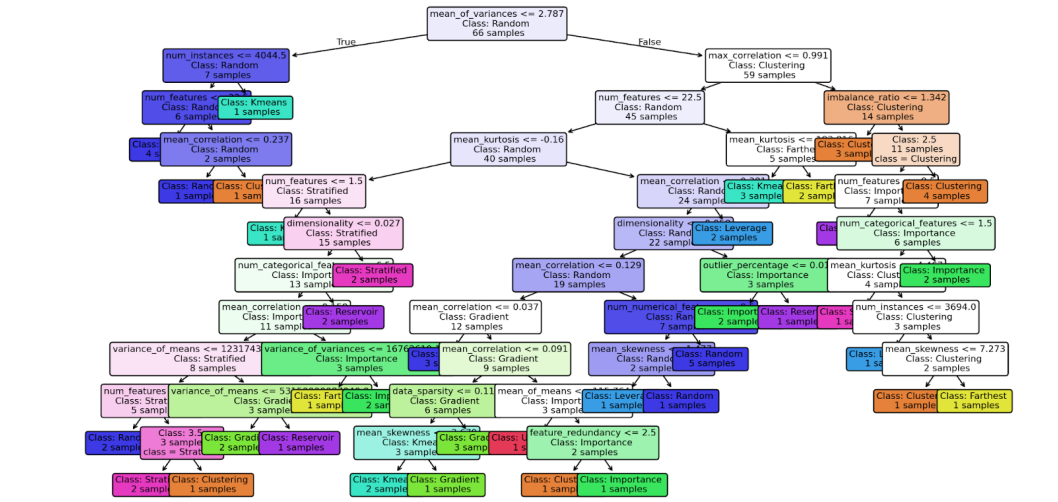

Figure 5: The resulting tree

As we can see, the tree first splits based on the mean of variances, if true the tree is traversed to the left, if false the tree is traversed to the right. Each node contains information regarding (1) the condition of splitting, (2) the majority class within the node, and (3) the number of samples that arrived at that node. The process of traversal continues until we reach a leaf, each leaf contains information about the best coreset selection technique, and the number of samples at that particular coreset.

The Decision Tree model achieved a disappointing accuracy of 40%. The model was especially biased towards Reservoir, which is expected given that Reservoir showed up on 25% of all observations.

We knew that the biggest limitation to our model is the number of datasets and our class imbalance, which introduced overfitting problems. To battle this, we performed data augmentation to double the size of our metadata. To perform this, we created a "twin" for every row. Each set of twins has the same inputs and outputs, but changes were made to the metadata. We introduced noise such that the copied cell is randomly distributed around the original value.

When running the decision tree model on the augmented dataset, a 99% accuracy was achieved, which was more problematic than impressive. Soon, we found out that we have to switch our testing method after augmentation. This is because we encountered the case where one twin would be in the testing sample while the other twin was in the training sample. Which was effectively equivalent to having non-disjoint sets of training and testing data. So, we tagged every twin, and ensured that twins stay close to each other.

The results improved after the data augmentation step and achieved an accuracy of around 50%. Still, there is much work to be done. So we continued investigating.

Using a decision tree provided us with good insights and interpretable results into what specific features of our meta-model influence the decision of selecting the best coreset. However, the model itself isn't the most sophisticated, and has some serious limitations. Namely its propensity to overfitting and bias towards the majority class. So, we decided to graduate to the Random Forest ensemble model. We figured that a Random Forest model will keep the interpretability of the decision tree, but create different trees that reach drastically different conclusions which helps counter overfitting.

The Random Forest model using the augmented data was tuned until we reached equilibrium at 73 trees and a maximum depth of 5. The performance of the Random Forest model was a 92% accuracy rate at a 90/10 training/testing split and an 86% accuracy at a 80/20 split. The Random Forest model that attempted to use just the orignal metadata took a creative approach to help reduce overfitting by using a transformer to add more weight to the task type feature. But its results were not nearly as good as the model using the augmented data. It saw roughly 40% accuracy rate. This version of the model was set up to make predictions on user defined data sets as well. Based on those predictions, it appeared the model was overfitted to the majority class; Reservoir.

4.7 Additional Insights

An analysis of the evaluation metrics data reveals additional insights into the performance of coreset methods based on specific metrics:

- **Multi-Class Classification**: None of the coreset methods distinctly outperformed others in metrics like ROC AUC, precision, recall, or F1 score. This suggests that either the methods struggled uniformly with the multi-class task or the evaluation did not capture significant differences. We think this might have been due to struggles with the Coresets to capture representitive data for datasets with a large number of classes.
- **Binary Classification**: Reservoir sampling performed best in terms of precision and F1 score, reflecting its reliability in accurately predicting the positive class. This may suggest that lack of complexity in our datasets allows for less complex coreset selection methods to excel. Also, Gradient sampling excelled in recall, indicating its strength in identifying true positives.
- **Regression**: K-means sampling minimized R2 score error, suggesting it was effective at representing the dataset for this task. However, the regression data also highlighted limitations across methods in achieving strong results.

These findings suggest that while certain methods excel in specific metrics, there may be opportunities to explore hybrid approaches or refine techniques to improve underrepresented metrics, particularly in challenging tasks like multi-class classification.

4.8 Engineering Considerations

The codebase is engineered with flexibility in mind, allowing for the seamless integration of different datasets. Key engineering strategies include:

- **Modular Design:**
  Functions are compartmentalized based on their roles. This modularity allows for easy updates and additions of new coreset selection techniques or preprocessing steps as needed.
- **Reproducibility:**
  A fixed random seed (RANDOM_STATE = 42) ensures consistent results across different runs, enhancing the reliability of experimental findings.
- **Efficiency:**
  Leveraging efficient libraries such as scikit-learn for machine learning tasks and minimizing computational overhead through optimized sampling techniques ensures that the project can handle large-scale datasets effectively.
- **Balance:** Sizes for the coresets are defined fractions that are constant for each task but may vary depending on the script. They were designed to balance the computational demand of each coreset selection technique without sacrificing accuracy.

- **Extensibility:**
  The framework is designed to accommodate additional coreset selection methods and datasets with minimal modifications, supporting the project's objective of analyzing a large number of datasets.

4.9 Current Code and Collected Data

https://github.com/colinward77/Meta-Learning-for-Coreset-Selectio.

5 Project Summary

Survey and Selection of Coreset Selection Techniques:

- Performed a literature review to identify existing coreset selection methods.
- Selected 10 diverse coreset selection techniques, including traditional (e.g., random sampling, stratified sampling) and advanced approaches (e.g., gradient-based, clustering-based, uncertainty sampling, leverage sampling).

Dataset Collection and Preprocessing

- Assembled a diverse collection of 76 datasets from Kaggle related to multiple domains such as finance, healthcare, retail, and technology.
- Implemented standardized preprocessing pipelines to handle missing values, encode categorical variables, and scale numerical features.

Feature Extraction and Metadata Creation:

- Determined relevant dataset characteristics to extract, including but not limited to task type (binary classification, regression, or multi-class classification), number of instances and features, data type distributions, class balance, missing values, dimensionality ratios, and feature correlations.
- Implemented feature extraction algorithm to compile the extracted features into a structured metadata format which are then collected in a csv file for the meta-model. This same algorithm was repurposed to allow the meta-model to make predictions on user-defined datasets.

Coreset Selection, Model Training, and Evaluation:

- Implemented and Applied the selected coreset selection techniques to all collected datasets.
- Used a Logistic Regression model for binary classification and multi-class classification tasks and Linear Regression model for regression tasks. Used separate standardized metadata extraction environments for the three task types.
- Evaluated model performance on each coreset with accuracy, precision, recall, F1-score, ROC AUC metrics or MSE, MAE, and R2 score.
- Recorded the performance metrics for all coreset selection techniques in separate evaluation metric csv files.

Meta-Model Development and Tuning:

- Developed a simple Decision Tree Model on the compiled metadata and then performed data augmentation on metadata to improve the resulting accuracy.
- Developed a Random Forest Classifier Model on compiled metadata to predict the most suitable coreset selection method for user defined datasets. Used transformer to alter feature weights. Did not use data augmentation
- Developed a second Random Forest Classifier Model on compiled metadata which had top performing accuracy and used data augmentation.

5.1 Roadmap

- **Week 7**: Conducted literature review and survey of coreset selection technique.
- **Week 8**: Selected original financial datasets, coreset techniques to use, and then refined project plan
- **Week 9**: Determined important dataset characteristics and developed code to extract metadata.
- **Week 10**: Developed full metadata and coreset performance code for binary classification and collected metadata on 10 datasets.
- **Week 11**: Midterm presentation and report initial findings
- **Week 12-13**: Refactored code to extract metadata from datasets used for regression and multi-class classification tasks and compiled more metadata.
- **Week 14**: Developed a decision tree model trained on our metadata and performed data augmentation.
- **Week 15:** Created two new meta-models using Random Forest Classifier each with unique attempts to improve accuracy, and tuned all 3 models.
- **Week 16:** Final presentation of framework, findings, and conclusions.

5.2 Division of work

While Colin and Ahmed have collaborated extensively and work on project tasks together as much as possible, they each have focused on specific areas within the code development stages.

Ahmed took the lead in implementing the majority of the coreset selection methods, ensuring that each sampling technique was accurately integrated and functioning within the classification codebase. Meanwhile, Colin handled the remaining critical tasks, including data preprocessing, feature extraction, training Logistic Regression models, and evaluating their performance metrics. Additionally, Ahmed spearheaded refining our project plan and the midterm presentation while Colin integrated matplotlib into the codebase to help produce visualizations for the presentation and compiled the current metadata. Also, both Colin and Ahmed jointly conducted literature reviews and selected datasets. This division of responsibilities allowed for an even and efficient progression of the classification tasks.

After expanding our scope, Colin worked mainly on collecting regression datasets and building the code base for the task genre, Ahmed worked on multi-class classification. Colin took charge of the process of collecting the metadata itself, but the search for datasets to use was split 50-50. Colin developed the majority of our meta-model framework, and Ahmed introduced data augmentation. Ahmed's forest introduction to the data augmentation included the development of the simple decision tree. Colin expanded his original model to use Random Forest after Ahmed's sucess and added a transformer for feature weighting while Ahmed developed the other final meta-model using Random Forest and the augmented data. Although Colin's final model saw worse accuracy than Ahmed's, it was the only version of the meta-model that allowed for users to define there own datasets and receive their own predictions.

For further distribution of coding, see comments in the code's functions.

Ultimately, both Colin and Ahmed worked hard and enabled one another to succeed in this project. We both believe the work to be equal when taking into consideration the intangibles buried between the lines of the idea-generation, drafting, optimization, integration, and communication processes. Each of us played to their own strengths and built upon each other. Teamwork, above all else, is the superlative element responsible for the success of this project.

References

[1] Groenfeldt, Tom. "At NYSE, the Data Deluge Overwhelms Traditional Databases." *Forbes*, Forbes Magazine, 15 Apr. 2024, www.forbes.com/sites/tomgroenfeldt/2013/02/14/at-nyse-the-data-deluge-overwhelms-traditional-databases/.

[2] Yang, Shuo, et al. "Mind the Boundary: Coreset Selection via Reconstructing the Decision Boundary" *OpenReview*, 6 June 2024, openreview.net/forum?id=hWng0GXeE4.

[3] Chai, Chengliang, et al. "Efficient Coreset Selection with Cluster-Based Methods." *ACM Conferences*, 4 Aug. 2023, dl.acm.org/doi/10.1145/3580305.3599326.

[4] Feldman, Dan. "Introduction to Core-Sets: An Updated Survey." *arXiv.Org*, 18 Nov. 2020, arxiv.org/abs/2011.09384.

[5] Zhang, Xiaoyu, et al. "SPECULATIVE CORESET SELECTION FOR TASK-SPECIFIC FINE-TUNING." *arXiv.org*, 2 Oct. 2024, arxiv.org/abs/2410.01296.

[6] Zhang, Jipeng, et al. "TAGCOS: Task-Agnostic Gradient Clustered Coreset Selection for Instruction Tuning Data." *arXiv.org*, 21 July 2024, arxiv.org/abs/2407.15235.

[7] Cochran, W. G. "Relative Accuracy of Systematic and Stratified Random Samples for a Certain Class of Populations." *The Annals of Mathematical Statistics*, vol. 17, no. 2, 1946, pp. 164–177, https://www.jstor.org/stable/pdf/2236036 .

[8] Settles, Burr. *Active Learning Literature Survey.* Technical Report #1648, University of Wisconsin–Madison, 2009, https://minds.wisconsin.edu/bitstream/handle .

[9] Vitter, Jeffrey Scott. "Random Sampling with a Reservoir." *ACM Transactions on Mathematical Software*, vol. 11, no. 1, 1985, pp. 37–57, https://dl.acm.org/doi/pdf/10.1145/3147.3165 .

[10]Bachem, Olivier, Mario Lucic, and Andreas Krause. "Practical Coreset Constructions for Machine Learning." *arXiv.org*, 4 June 2017, arxiv.org/abs/1703.06476.