POLITECNICO
MILANO 1863

Data Information and Quality

Project ID: 46
Data Quality: Dimensionality & Duplication
ML Task: Classification

Sara Limooee, 10886949
Seyedpeyman Alavi, 10876737

Winter 2024

1- Abstract

In the world of data-driven decision-making, making good machine learning models relies on having high quality data. In this project, we are going to focus on two key aspects of data quality: dimensionality and duplication, examining their impact on classification tasks with various classification algorithms such as "Decision Tree", "Logistic Regression", "KNN", "Random Forest", "AdaBoost", "MLP".

Dimensionality refers to the number of features or records in a dataset, affecting model complexity and efficiency. Duplication involves redundant information, impacting the integrity of datasets. Our goal is to check how these two factors affect ML algorithm performance.

1- Setup choices

For doing the described tasks above, we are going to proceed with these assumptions:
Dimensionality

2-1- 10 experiments with a fixed number of samples and a varying number of features.
2-2- 10 experiments with a fixed number of features and varying number of samples.

Duplication:

2-1- For one experiment: varying % duplicated rows, for example from 5% to 50% (with an increasing step of 5%) of duplicated rows.
2-2- 10 experiments with different % similarity: for example, creating and adding non-exact duplicated rows, based on a fixed similarity measure.

2-1- For the first experiment, we are going to discuss the choices made for the data collection phase. Since here we want to examine dimensionality, as explained in the project guidelines file, the data preparation phase is requested for: Completeness, Accuracy, Duplication. So, there is no need to do anything for this phase. For this part, we fix the number of samples as 1000 and change the number of features for 10 experiments such that in every experiment, the number of features will be 5 plus a variable in range of (10, 110, 10). Then, we will call each classification algorithm for each of these new datasets.

```
for n_additional_features in range(10, 110, 10):
    # A: DATA COLLECTION
    X, y = make_dataset_for_classification(
        n_samples = 1000,
        n_features = 5 + n_additional_features,
        n_informative = 5,
        n_redundant = 0,
        n_repeated = 0,
        n_classes = 2,
        n_clusters_per_class = 2,
        weights = None,
        flip_y = 0.01,
        class_sep = 1.0,
        hypercube = True,
        seed = SEED
    )
```

2-2- For this experiment, same as the previous one, we are going to discuss the choices made for the data collection phase.
For this part, we fix the number of features as 5 and change the number of samples for 10 experiments such that in every experiment, the number of features will be 1000 plus a variable in range of (10, 110, 10). Then, we will call each classification algorithm for each of these new datasets.

```
for n_additional_samples in range(10, 110, 10):
    # A: DATA COLLECTION
    X, y = make_dataset_for_classification(
        n_samples = 1000 + n_additional_samples,
        n_features = 5,
        n_informative = 5,
        n_redundant = 0,
        n_repeated = 0,
        n_classes = 2,
        n_clusters_per_class = 2,
        weights = None,
        flip_y = 0.01,
        class_sep = 1.0,
        hypercube = True,
        seed = SEED
    )
```

2-3- For the third experiment, we are going to do one experiment and for this one, keep the similarity percentage the same while changing the number of duplicated rows added to the dataset. In another word, we create a base dataset with 1000 samples and 5 features and then call the "add_non_exact_duplicates(X, y, percentage_added)" function on the base dataset to retrieve sets with different number of duplicated rows. The percentage_added variable is in range of (0.05, 0.5, 0.05). The details and implementation of mentioned function is in the next section.

```
# DATA COLLECTION
X, y = make_dataset_for_classification(
    n_samples = 1000,
    n_features = 5,
    n_informative = 5,
    n_redundant = 0,
    n_repeated = 0,
    n_classes = 2,
    n_clusters_per_class = 2,
    weights = None,
    flip_y = 0.01,
    class_sep = 1.0,
    hypercube = True,
    seed = 2023
)

results_for_each_algorithm_before_deduplication = []
results_for_each_algorithm_after_deduplication  = []
for algorithm in CLASSIFICATION_ALGORITHMS:

    results_single_algorithm_1 = []
    results_single_algorithm_2 = []
    for percentage_added in np.arange(0.05, 0.55, 0.05).tolist():
        # DATA POLLUTION
        X_duplicated, y_duplicated = add_non_exact_duplicates(X, y, percentage_added)
```

2-4- For the last experiment, we will create 10 different datasets with same number of duplicated rows and different similarity percentages. For this purpose, we defined the function “add_non_exact_duplicates_2” which gets the similarity percentage which is a float number between 0 and 1 as input and adds default (200) number of duplicated rows to the base dataset with 1000 samples and 5 features.

```
# DATA COLLECTION
X, y = make_dataset_for_classification(
    n_samples = 1000,
    n_features = 5,
    n_informative = 5,
    n_redundant = 0,
    n_repeated = 0,
    n_classes = 2,
    n_clusters_per_class = 2,
    weights = None,
    flip_y = 0.01,
    class_sep = 1.0,
    hypercube = True,
    seed = 2023
)

results_for_each_algorithm_before_deduplication = []
results_for_each_algorithm_after_deduplication  = []

for algorithm in CLASSIFICATION_ALGORITHMS:

    results_single_algorithm_1 = []
    results_single_algorithm_2 = []

    for percentage_similarity in np.arange(0.05, 1, 0.1):
        # DATA POLLUTION
        X_duplicated, y_duplicated = add_non_exact_duplicates_2(X, y, percentage_similarity)
```

3- Pipeline Implementation

As also explained in the previous section, data preparation phase is requested for: Completeness, Accuracy, Duplication. So, the pipeline for each of the two dimensions checked is as below:

Dimensionality:

The first and second phase is the data collection and data pollution which we must create the requested issues in the dataset, dimensionality, by calling the function “make_dataset_for_classification” with parameters explained in previous section. Then we will analyze and evaluate the effect of dimensionality on both features and number of samples.

Duplication:

For this task, we first create a base dataset. Then call a data pollution function to create the requested issue, which is duplication. After that we must analyze and evaluate it once without data preparation phase and once after we performed data preparation. For that purpose and for all the experiments requested, we have defined 3 functions which we will explain in detail in the following:

- add_non_exact_duplicates():
  This function adds different number of non-exact duplicates to the base dataset X, y. percentage_added is a float number between 0 and 1 which indicates the number of duplicates. Based on that, we create non-exact duplicates. By non-exact duplicates we mean that we do not copy the record, but we change the values by adding a random number from normal distribution with std equals to 0.1 and mean equals to 0.
  This function keeps the similarity percentage of each record and its duplicate the same and only varies in the number of duplicates.

```python
def add_non_exact_duplicates(X: np.array, y: list, percentage_added: float) -> tuple:
    X_duplicates, y_duplicates = list(), list()

    selected_indices = np.random.choice(len(X), int(len(X) * percentage_added), replace = False)

    for index in selected_indices:
        X_duplicates.append( X[index] + np.random.normal(0, 0.1, size = X[index].shape) )
        y_duplicates.append( y[index] )

    X_duplicated = np.vstack([X, np.array(X_duplicates)])
    y_duplicated = np.concatenate([np.array(y), np.array(y_duplicates)])

    np.random.shuffle(X_duplicated)
    np.random.shuffle(y_duplicated)

    return X_duplicated, y_duplicated
```

- add_non_exact_duplicates_2():
  This function adds same number of non-exact duplicates to the base dataset X, y with different similarity percentages. The similarity percentage here is defined as the number of features to be changed in the duplicated record. For this purpose, we choose default (200) number of records randomly and choose some of the features again randomly based on the similarity percentage * number of features. Then we create non-exact duplicates by adding a random number from normal distribution to the selected features.

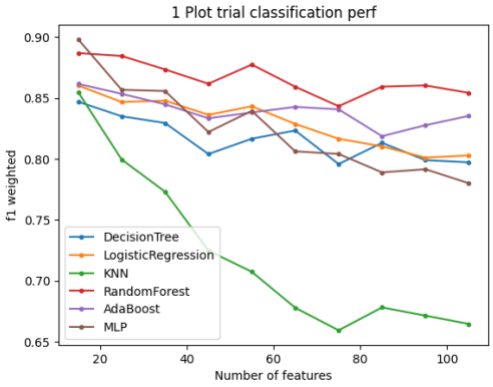
```python
# it has fixed number of percentage_added
def add_non_exact_duplicates_2(X: np.array, y: list,  percentage_similarity: float, num_added: int = 200) -> tuple:

    X_duplicates, y_duplicates = list(), list()

    selected_indices = np.random.choice(len(X), num_added, replace = False)

    for index in selected_indices:
        selected_features = np.random.choice(len(X[0]), int(len(X[0]) * (1-percentage_similarity)), replace = False)

        new_X_index = X[index]
        new_X_index[selected_features] = new_X_index[selected_features] + np.random.normal(
            0, 0.1, size = new_X_index[selected_features].shape
        )

        X_duplicates.append( new_X_index )
        y_duplicates.append( y[index] )

    X_duplicated = np.vstack([X, np.array(X_duplicates)])
    y_duplicated = np.concatenate([np.array(y), np.array(y_duplicates)])

    np.random.shuffle(X_duplicated)
    np.random.shuffle(y_duplicated)

    return X_duplicated, y_duplicated
```

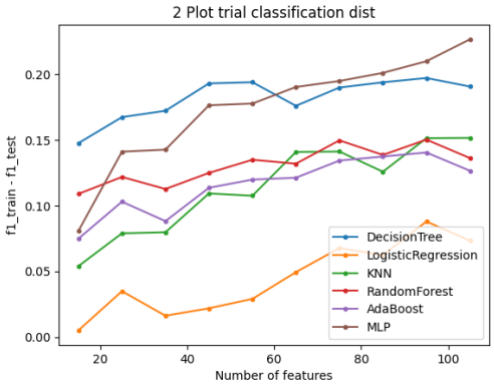
- remove_duplicates():
  This function identifies and eliminates duplicate records within a dataset. Using the record linkage library, the function creates an index of all possible pairs of records and compares them. The function uses the "Levenshtein" similarity method to compare string values of each feature. This method computes a similarity score, and pairs with a similarity score exceeding a specified threshold (0.8 in this case) are considered potential duplicates.

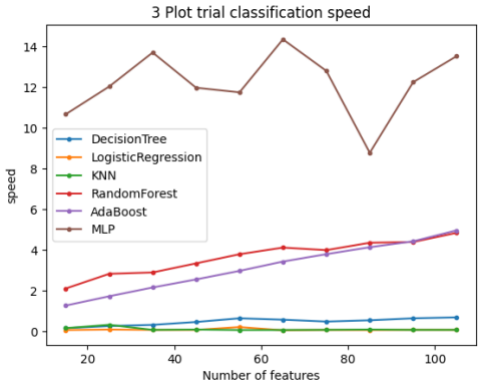
```python
def remove_duplicates(X_duplicated: np.array, y_duplicated: np.array):
    # Combine X and y into a DataFrame for record linkage
    data = pd.DataFrame(np.column_stack((X_duplicated, y_duplicated)), columns=[f'X{i}' for i in range(X_duplicated.shape[1])] + ['y'])

    # Convert float values to strings
    data = data.astype(str)

    # Record linkage to detect and remove duplicates
    indexer = recordlinkage.Index()
    indexer.full()
    pairs = indexer.index(data)

    compare_cl = recordlinkage.Compare()

    # Adjust the labels in the string method to match the column labels
    for i in range(X_duplicated.shape[1]):
        compare_cl.string(f'X{i}', f'X{i}', method='levenshtein', threshold=0.8)

    features = compare_cl.compute(pairs, data)

    # Assuming a threshold of 0.8 for levenshtein similarity, you can adjust this based on your needs
    potential_duplicates = features[features.sum(axis=1) > 0.8].index.get_level_values(1)

    # Remove the detected duplicates
    data_no_duplicates = data.drop(index=potential_duplicates)

    # Separate X and y again
    X_no_duplicates = data_no_duplicates.iloc[:, :-1].values
    y_no_duplicates = data_no_duplicates['y'].values

    return X_no_duplicates, y_no_duplicates
```

4- Results

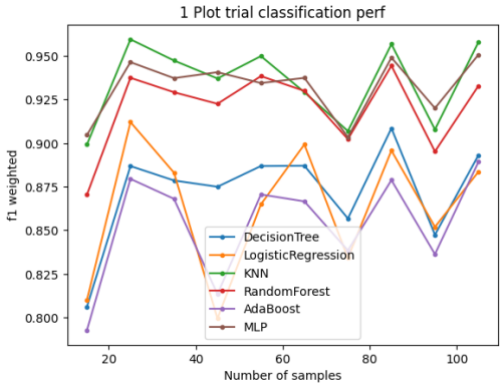
- Dimensionality Reduction on Number of Features

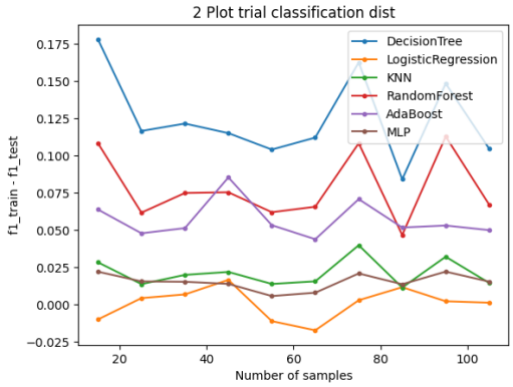


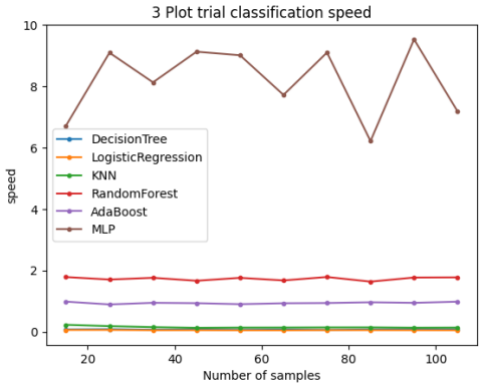




- Dimensionality Reduction on Number of Samples



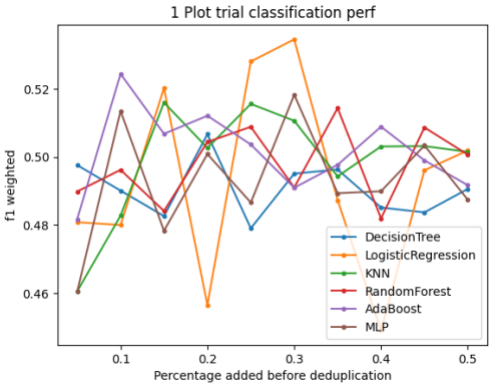


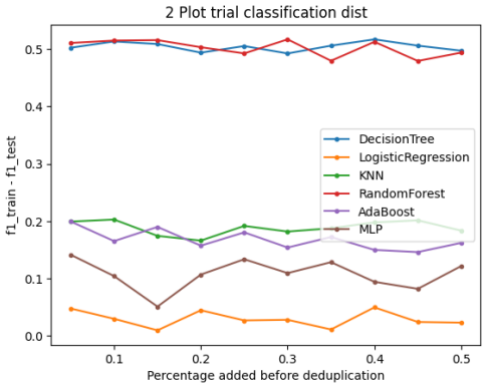


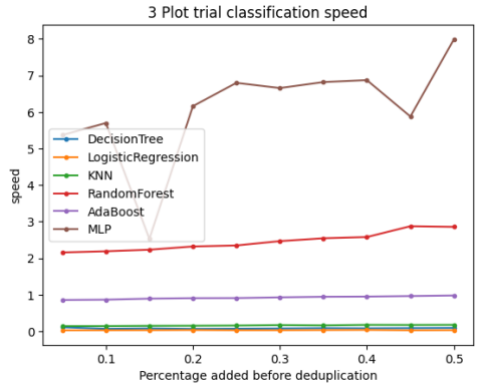
Result:

The impact of dimensionality on machine learning classification algorithms is substantial across various performance measures. In terms of speed, higher dimensionality generally results in increased computational complexity during both training and inference, leading to longer processing times.
In high-dimensional spaces, the concept of the "curse of dimensionality" becomes relevant. It refers to various phenomena where the distance between points becomes less meaningful as the number of dimensions increases. In high-dimensional spaces, the data points tend to become sparser, and the notion of proximity can lose its significance. This can affect distance-based algorithms (e.g., k-nearest neighbors) as the concept of "closeness" becomes less meaningful in high-dimensional spaces.
In addition, the models with high dimensionality datasets are more likely to overfit which affects the overall performance of machine learning classification algorithms.

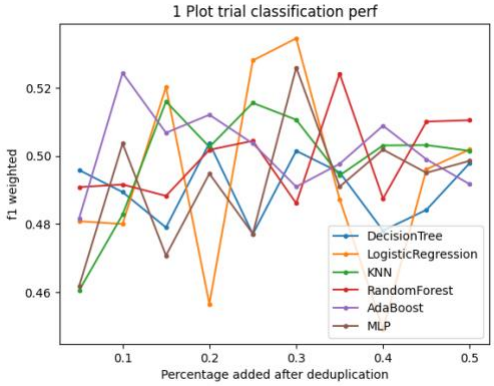
- Duplication Removal with same similarity & different number of duplicates
  - Before data preparation

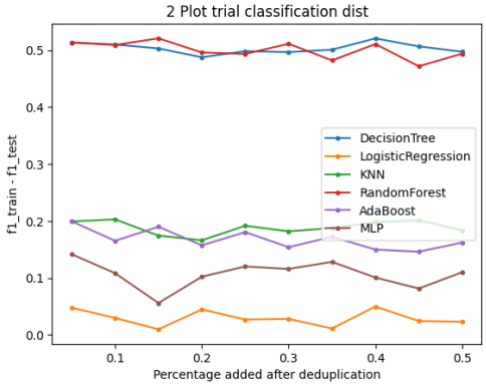


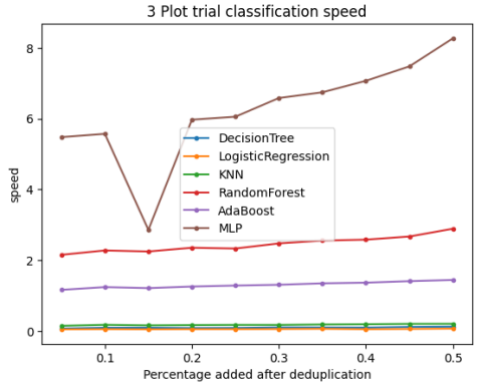




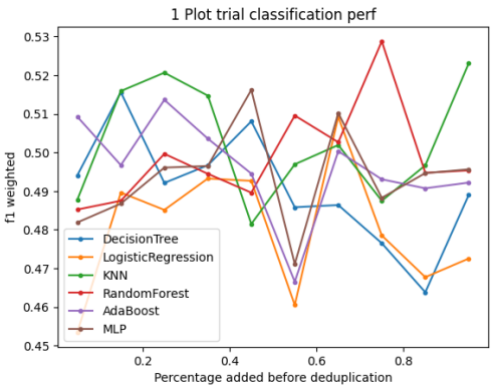
  - After data preparation

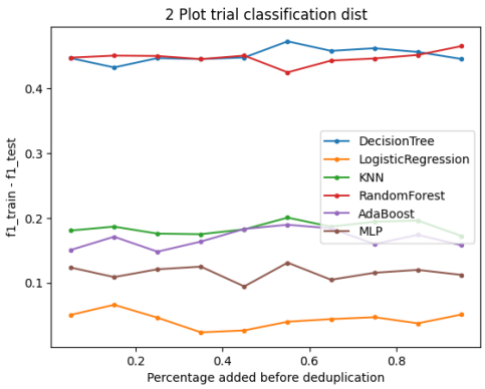


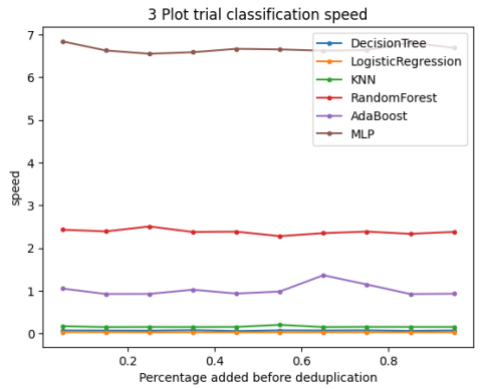




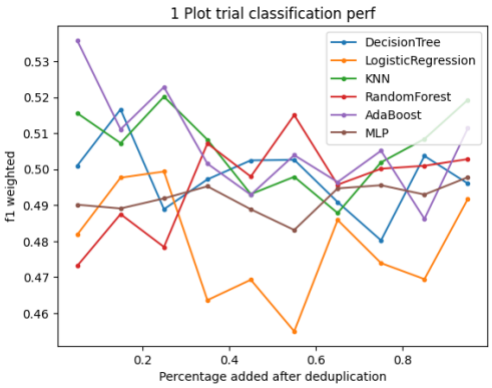
- Duplication Removal with different similarity & same number of duplicates
  - Before data preparation

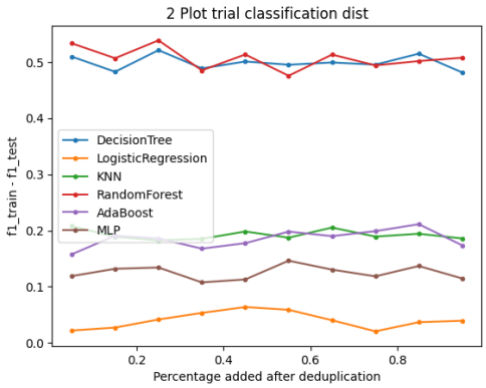


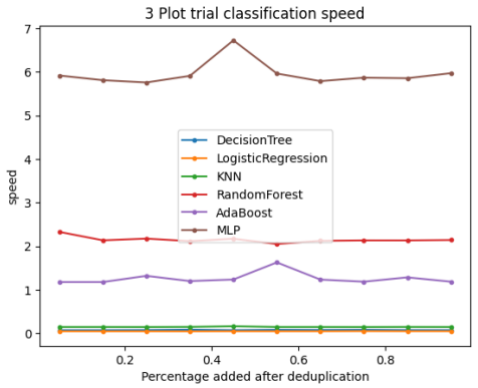




- After data preparation







Result:

The impact of duplication on machine learning classification algorithms, including aspects of speed, distance measure, and overall performance, varies based on factors such as the classification algorithm and dataset. Increasing the number of duplicates may affect training speed and leads to redundant computations while higher similarity percentages could contribute to slower training. Duplicates influence distance-based algorithms, such as k-nearest neighbors (KNN), affecting decision boundaries and may cause overfitting.