# Exercise 1: Classification

Rois Lea Michaela (11810807) Bhardwaj Aman (12333472)
Haubenburger Gabriel (11840531)

2024-10-23

## Introduction

### Data Set 1: Wine

The wine data set is a comparatively small data set we chose for classification. It does not contain any missing values. There are 178 instances with 13 numerical attributes. The 14th attribute contains the target variable, which is the class (type of Wine, there are three different ones: 1, 2 and 3) which is used for classification. We have 71 instances of class 2, 59 of class 1 and 48 of class 3, so classes are not extremely disparate in size. All the attributes contain only values of the ratio data type (numeric), which does not restrict the use of mathematical operations on them in possible pre-processing steps. Overall the data set seems well behaved and should require little pre-processing, since we only have well distributed numerical values.

### Data Set 2: Sick

The sick dataset contains thyroid disease records provided by the Garavan Institute and J. Ross Quinlan in 1987. It comprises 3772 instances with 30 features: 7 numeric and 23 symbolic. The dataset is labeled with two classes: "sick" (positive) and "negative." There are 6064 missing values, distributed across all instances. Since one of the features is mono-valued and the other contains only missing values, both can be dropped during the preprocessing. This dataset is useful for binary classification tasks and reflects real-world challenges like missing data.

### Data Set 3: Congressional Voting

The voting dataset consists of 218 samples with 17 features and 2 classes representing the two different parties democrat and republican. The dataset contains missing values ("unknown") and categorical variables ("y" and "n").

### Data Set 4: Amazon Reviews

The reviews dataset is comparable large, consisting of 750 samples, 10 000 numerical features and 50 classes that represent the names of the authors who wrote the reviews. It contains missing values.

# Classifiers

## Multi-layer Perceptron (MLP)

MLP is an extension of the simple perceptron. Instead of a single artificial neuron, it consists of (many) perceptrons arranged in (possibly multiple) hidden layers. The input for each layer comes from the output from the previous layer.

MLP is purely feed-forward, so the output of a given layer never feeds back into itself or a previous layer.

MLP trains using Backpropagation, utilizing gradient descent to reduce the loss starting from the final layer.

For Classification, MPL provides probability estimates for the classes, and can thus also easily be used for Multi-Labeling.

We used the MLPClassifier implementation of the scikit learn library. While this model has many parameters, we chose to just focus on tuning the four we believe to be most important, namely the solver (the version of gradient descent used), the number of hidden layers and their neurons, the activation function of neurons and alpha (which represents the learning rate).

## Decision Trees

A Decision Tree is a supervised machine learning algorithm used for both classification and regression tasks. It works by recursively splitting the data based on feature values, creating a tree-like structure with decision nodes and leaf nodes. The tree's root node represents the feature that best divides the data, while internal nodes represent further splits based on other features, and leaf nodes give the final prediction (either class labels for classification or values for regression). The splits at each node are determined using criteria like Gini Index, Entropy, or Mean Squared Error (MSE), depending on whether the task is classification or regression. Decision trees are powerful for capturing non-linear relationships in the data and can be easily visualized, making them a valuable tool in machine learning.

## Gaussian Process Classifier

The Gaussian Process Classifier (GPC) is a probabilistic, non-parametric machine learning model designed for classification tasks. This type of classifier models the distribution of potential outcomes based on a Gaussian process prior over the latent functions that map inputs to outputs. After incorporating the observed data, this results in a posterior distribution. The probability / likelihood of class membership for given inputs is estimated by applying a so called link function, such as the logistic of probit function, to the latent Gaussian process. Since GPC does not only predict the class membership, but also provides a measure of uncertainty in the predictions, the classifier is particularly useful for tasks where understanding prediction confidence is crucial. The model can manage non-linear, complex decision boundaries and adjust to various data distributions. However, especially when dealing with large datasets, the use of GPC can be computationally intensive.

### Kernel Functions

The choice of kernel functions $k(x, x')$ encodes prior assumptions about the form of the latent function $f(x)$:

- RBF Kernel: smooth, continuous functions
- Matern Kernel: functions with varying smoothness
- Rational Quadratic Kernel: used for modelling functions with multiple length scales

# Data Exploration and Pre-Processing

- **Feature Distributions:** The distributions of the features were explored for all four datasets. For the Sick and Reviews dataset the Principal Component Analysis (PCA) was used in order to reduce the dimensionality by transforming the original high-dimensional data into a smaller set of uncorrelated components while preserving as much variance - and therefore information - in the data as possible. The components of PCA are called Principal Components (PC1 is the linear combination of original features that explains the maximum variance in the data and PC2 is orthogonal to PC1 and corresponds to the second-most variance). Investigating the distributions helped to understand some of the underlying statistical properties. An example of a feature distribution from the wine dataset (Alcohol) is shown in Fig.1.

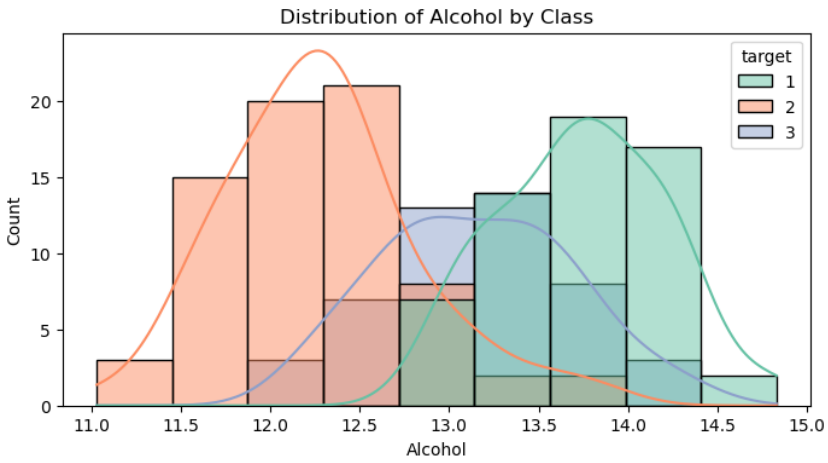



Figure 1: Wine: Distribution of the feature alcohol by wine class

- **Pairwise Feature Relationships:** Pairwise relationships between features were examined to understand correlations and dependencies within the dataset. Fig.2 gives an example of the pairwise feature relationships for the wine dataset. Determining the relationships helped to understand how features interact and whether some features are redundant or highly correlated.
- **Feature Correlation Heatmaps:** In addition, correlation heatmaps (one example for the wine dataset is shown in Fig.3) were generated to visually analyze the linear relationships between the features. This simplified identifying potential correlations, allowing for better-informed decisions about feature selection and scaling.

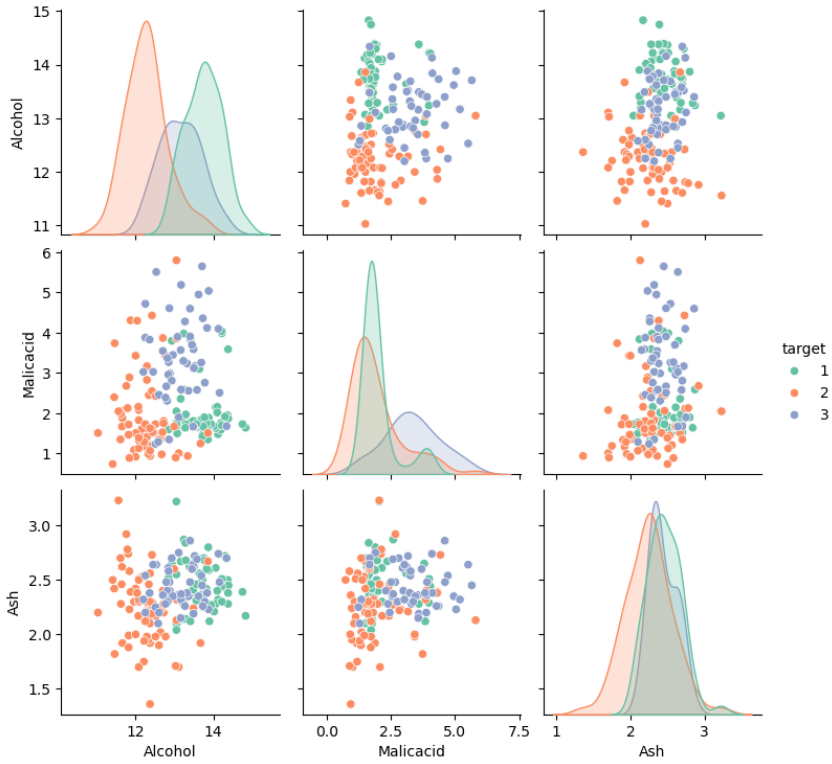



Figure 2: Wine: Relationships between the three features Alcohol, Malicacid and Ash

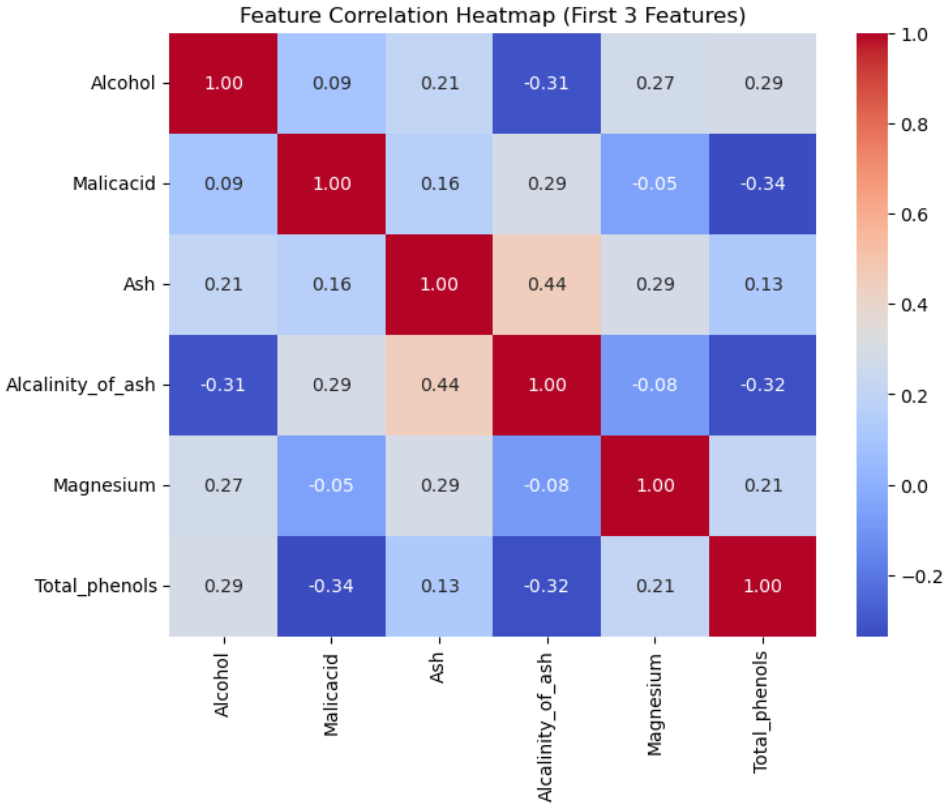



Figure 3: Wine: Feature correlation heatmap

- **Scaling the features:** Since some classifier models (e.g., Gaussian Process) are sensitive to the scale of input features, different feature scaling methods were tested to ensure that the model performs optimally. The following scalers were applied:

  - **StandardScaler**: This scaler standardizes the features by removing the mean and scaling to unit variance. It is suitable for features with a normal distribution.
  - **MinMaxScaler**: This scaler scales the features to a specific range (typically [0, 1]). It is particularly useful when the features have different units or when a uniform scale is important.
  - **RobustScaler**: This scaler scales the features using statistics that are robust to outliers (i.e., median and interquartile range). It was tested to check if outliers in the data affect the model performance.

- **Missing values:** So, first of all, the missing values in the datasets were handled. The wine and Reviews datasets did not have any missing values. The datasets Voting and Sick had missing values as "Unknown" and "?" respectively. Also, the Sick dataset had one column in which all the values were missing ("?") and one more column in which all the values were the same. So, we dropped these columns and as for the remaining missing values, the columns which had categorical values were imputed with Mode ('most_frequent') and columns which had numerical values were imputed with Median.

- **Encoding:** The wine dataset had no column names so the column names were added from 0-13 (0 being the target variable). All the data in wine was numerical. In the Reviews dataset, the target variable "Class" had categorical classes which were encoded to numerical. The voting dataset had its data as 'y' and 'n' which were encoded as '1' and '0'. The target variable "class" had two classes, "democrat" and "republican", which were encoded as '0' and '1' respectively. Finally, in the Sick dataset, the Sex column was encoded '0' and '1' for 'F' and 'M'. The other values 'f' and 't' were encoded as '0' and '1'. The column 'referral source' having SVHC, other, SVI, STMW, SVHD were encoded 0, 1, 2, 3, and 4 respectively. The target variable 'Class' having two classes 'negative' and 'sick' were encoded as '0' and '1'.

  Feature selection was also applied using Variance Threshold on Reviews dataset to remove features with low variance. Finally, the plots for all the datasets comparing Missing Values, Encoding (No. of Categorical and Numerical Columns), Distribution of Target Column and Feature Count before and after pre-processing steps were generated. Here are the plots:

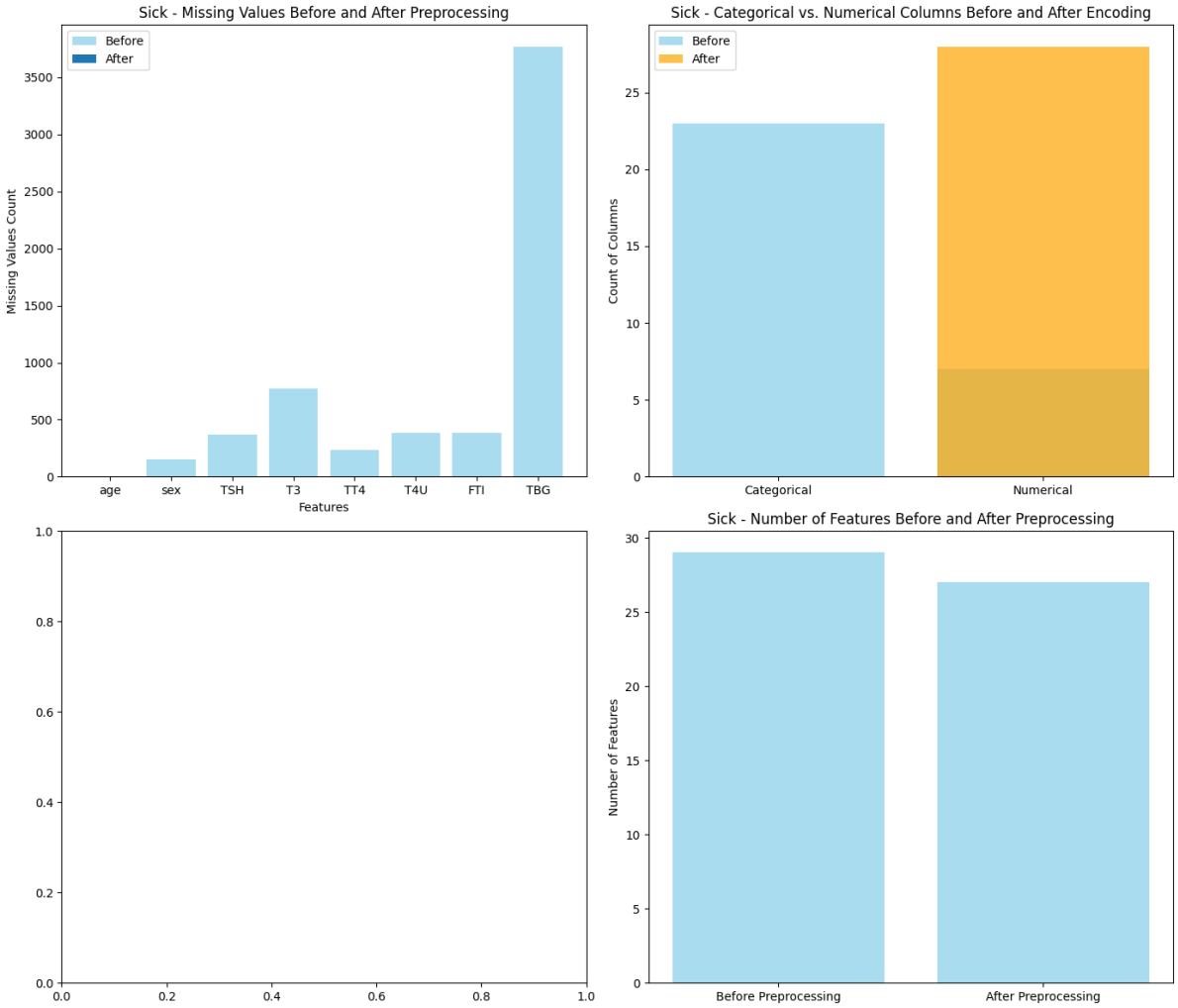



Figure 4: Sick dataset pre-processing comparison

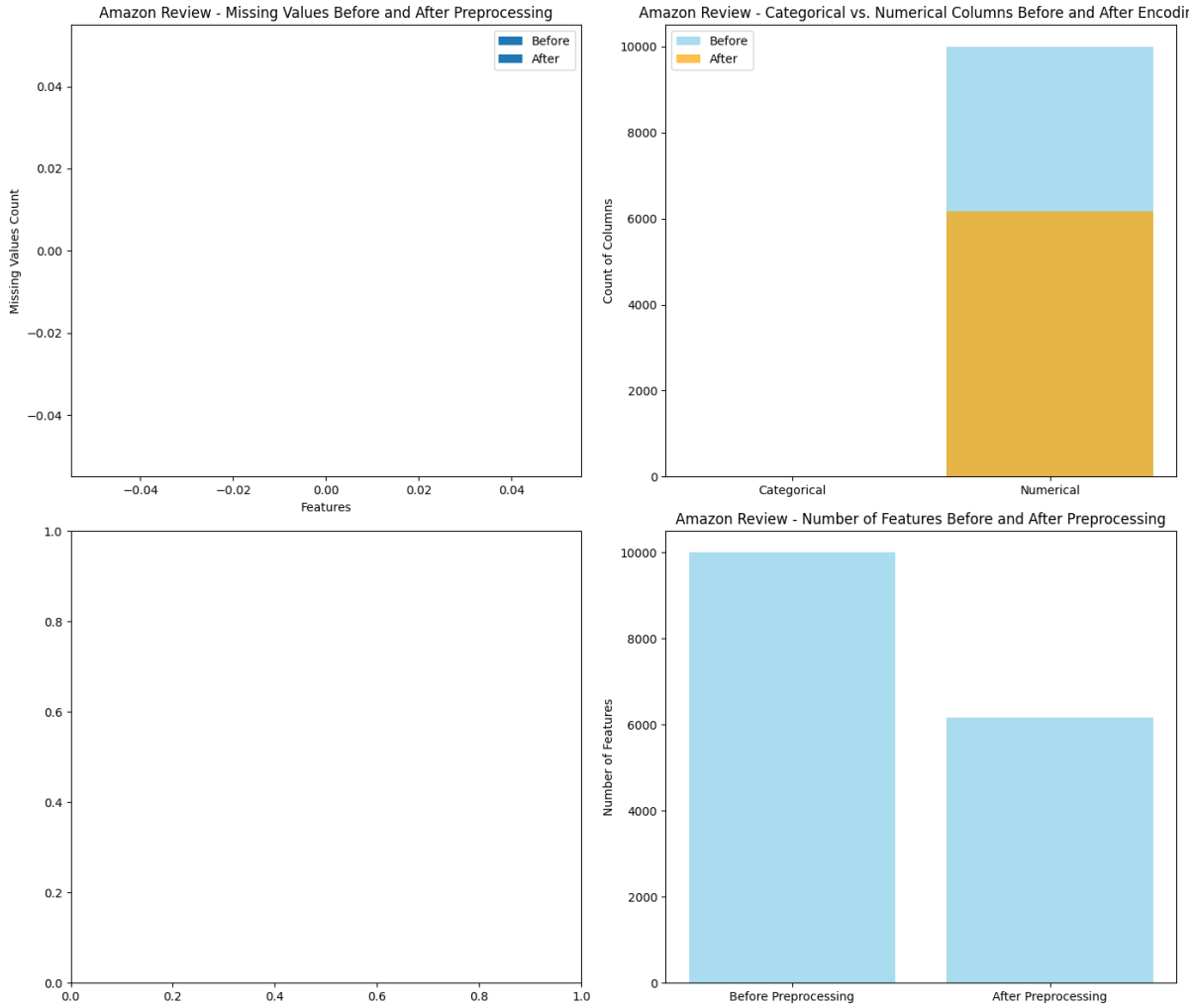



Figure 5: Reviews dataset pre-processing comparison

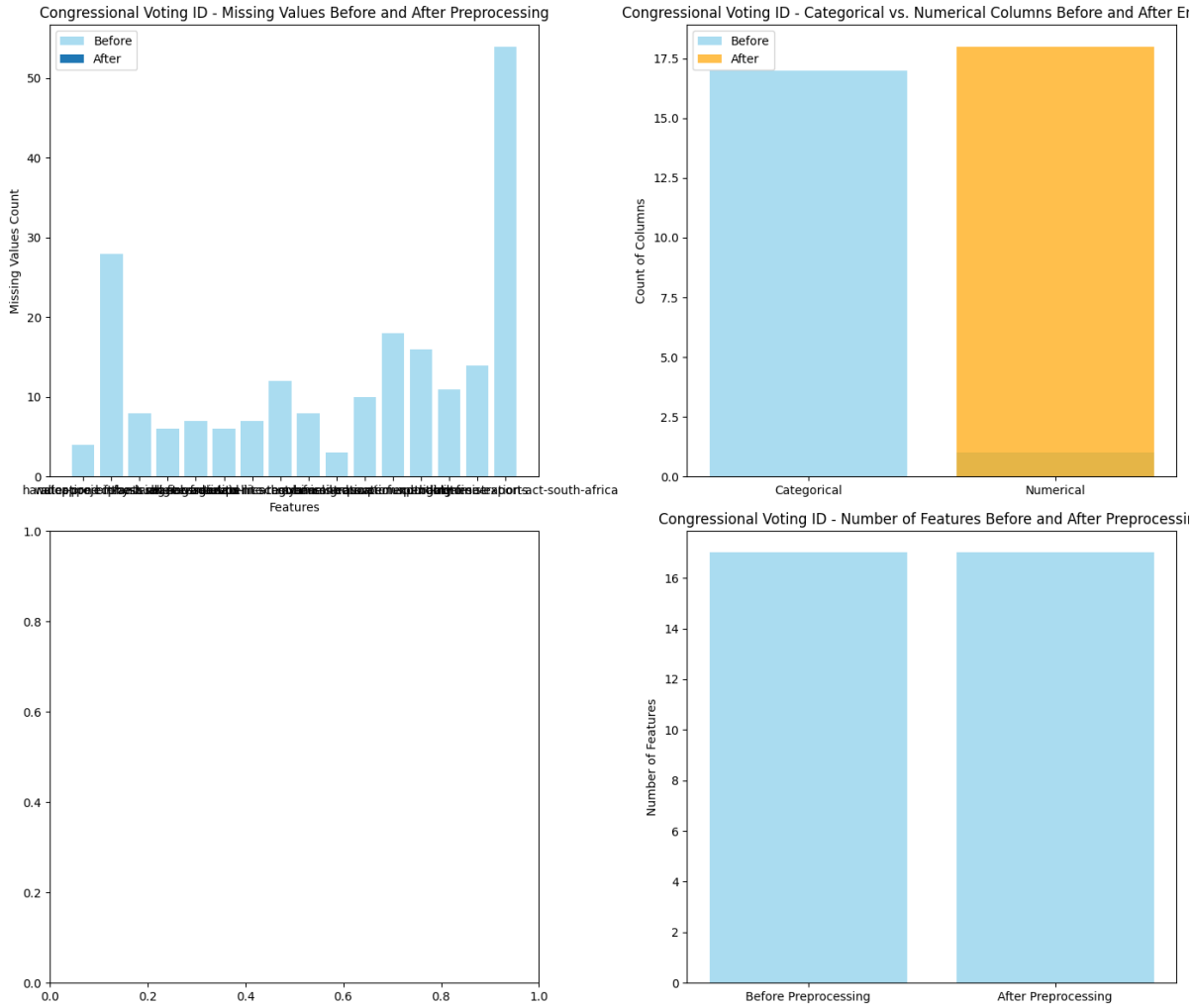



Figure 6: Voting dataset pre-processing comparison

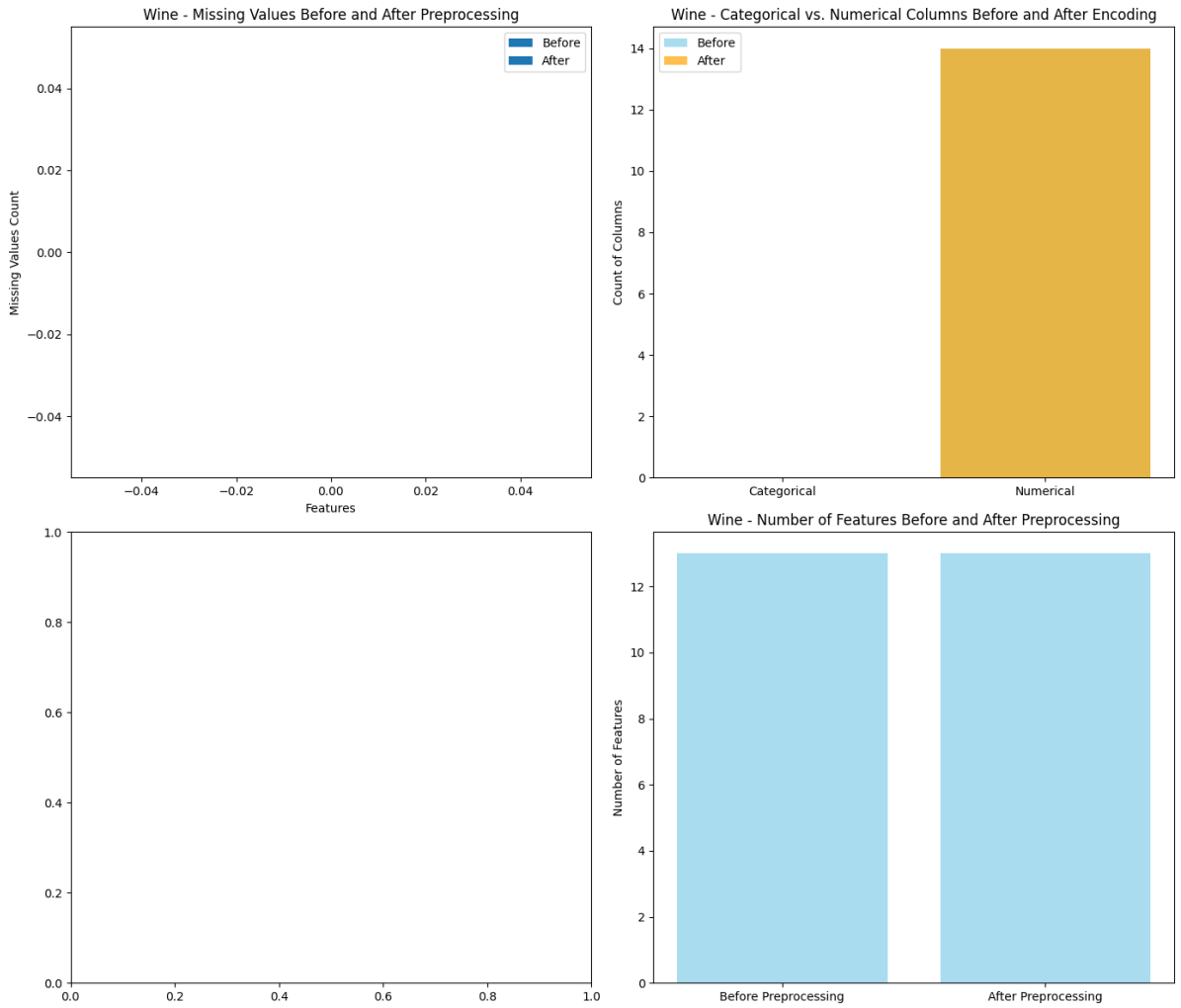



Figure 7: Wine dataset pre-processing comparison

# Findings

## Data Set 1: Wine

### MLP

Table 1: Wine: MLP

| Scaler | Solver | $\alpha$ | Activation | Hidden Layers | Mean Accuracy | Std. Dev. |
|---|---|---|---|---|---|---|
| StandardScaler | 'lbfgs' | 1e1 | 'identity' | (15, 15) | 0.95 | 0.05 |
| MinMaxScaler | 'lbfgs' | 1e-4 | 'logistic' | (15, 2) | 0.98 | 0.05 |
| RobustScaler | 'sgd' | 1e-8 | 'identity' | (100,) | 0.97 | 0.05 |

Holdout accuracies were 1.0, 0.96 and 0.98, respectively.

The parameters were chosen via Grid Search on the following parameter grid:

```
parameters = {
    "activation": ('identity', 'logistic', 'tanh', 'relu'),
    "solver": ('lbfgs', 'sgd', 'adam'),
    "hidden_layer_sizes": ((15,2), (100,), (15,15), (20,3)),
    "alpha": np.logspace(-8, 3, 12),
}
```

### Decision Trees

TBA

### Gaussian Process Classifier

Table 2: Wine: GPC

| Scaler | Kernel | Optimizer | Length Scale | Mean Accuracy | Std. Dev. |
|---|---|---|---|---|---|
| Std.Sc. | RBF | 'fmin_l_bfgs_b' | 1.0 | 0.98 | 0.04 |
| MinMaxSc. | Matern | None | 1.0 | 0.97 | 0.05 |
| RobustSc. | Rat.Quad. | None | 1.5 | 0.98 | 0.04 |

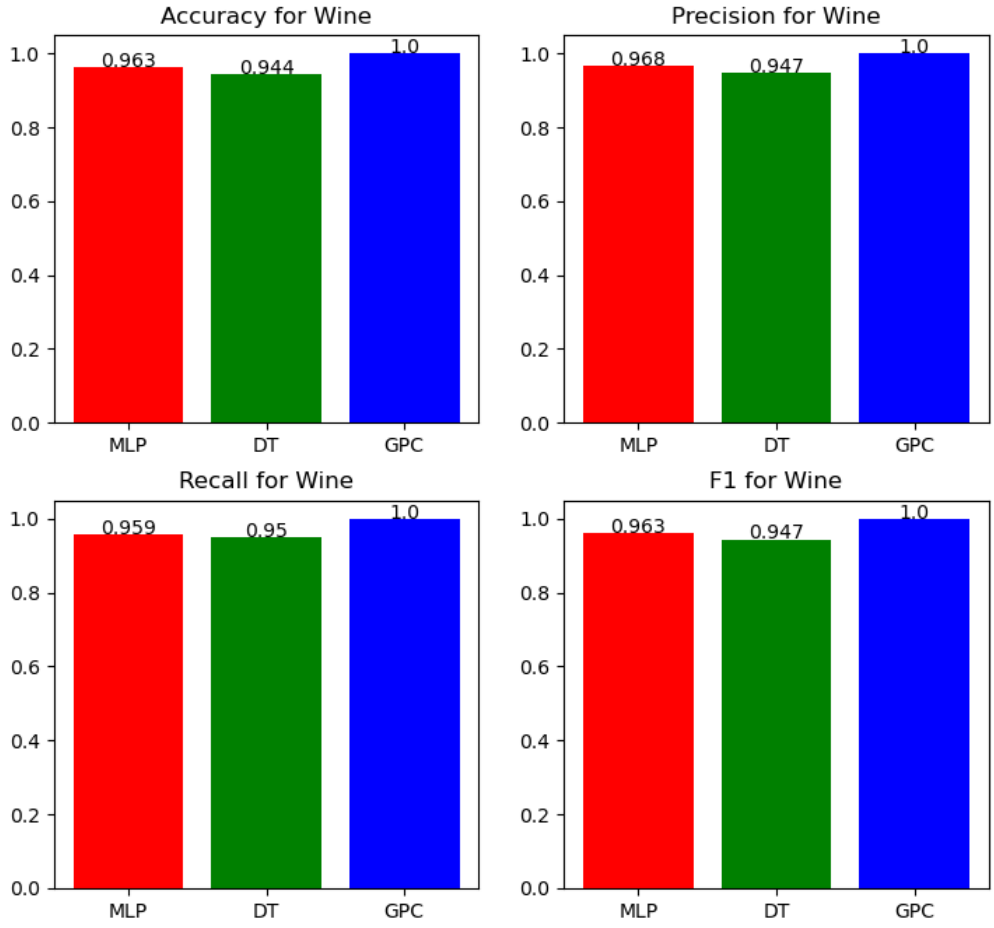
Holdout accuracies were 0.98 for each of the three different combinations.

The parameters were chosen via RandomizedSearchCV on the grid

```
parameters = {
    "kernel": ('RBF', 'Matern', 'RationalQuadratic'),
    "length_scale : (0.1, 1.0, 1.5, 2.0),
    "optimizer": (None, 'fmin_l_bfgs_b')
}
```

### Comparison

We took the best performing models (in regards to accuracy) for each classifier on the Wine data set and compared them in regard to their accuracy, precision, recall and F1 scores. The result is shown in the graph below.




Training times were 0.059,0.003 and 0.278 seconds for MLP, DT and GPC, respectively.

## Data Set 2: Sick

### MLP

Holdout accuracies were 0.958, 0.973, 0.96, 0.983 and 0.966, respectively.

The parameters were chosen via RandomizedSearchCV on the grid

```
parameters = {
    "activation": ('identity', 'logistic', 'tanh', 'relu'),
    "solver": ('lbfgs', 'sgd', 'adam'),
    "hidden_layer_sizes": ((15,2), (100,), (15,15), (20,3), (50, 50),
        (20, 20, 20), (100,100,100)),
    "alpha": np.logspace(-10, 4, 15),
```

Table 3: Sick: MLP

| Scaler | Solver | $\alpha$ | Activation | Hidden Layers | Mean Accuracy | Std. Dev. |
|---|---|---|---|---|---|---|
| None | lbfgs | 1e-1 | identity | (15, 15) | 0.96 | 0.01 |
| StandardScaler | lbfgs | 1e0 | relu | (15,15) | 0.97 | 0.01 |
| MinMaxScaler | lbfgs | 1e-1 | tanh | (50, 50) | 0.97 | 0.01 |
| RobustScaler | lbfgs | 1e0 | relu | (15,15) | 0.98 | 0.01 |
| MaxAbsScaler | adam | 1e-4 | relu | (100,100,100) | 0.97 | 0.02 |

```
}
```

due to the larger data set size making a full grid search too costly.

### Decision Trees

TBA

### Gaussian Process Classifier

Table 4: Sick: GPC

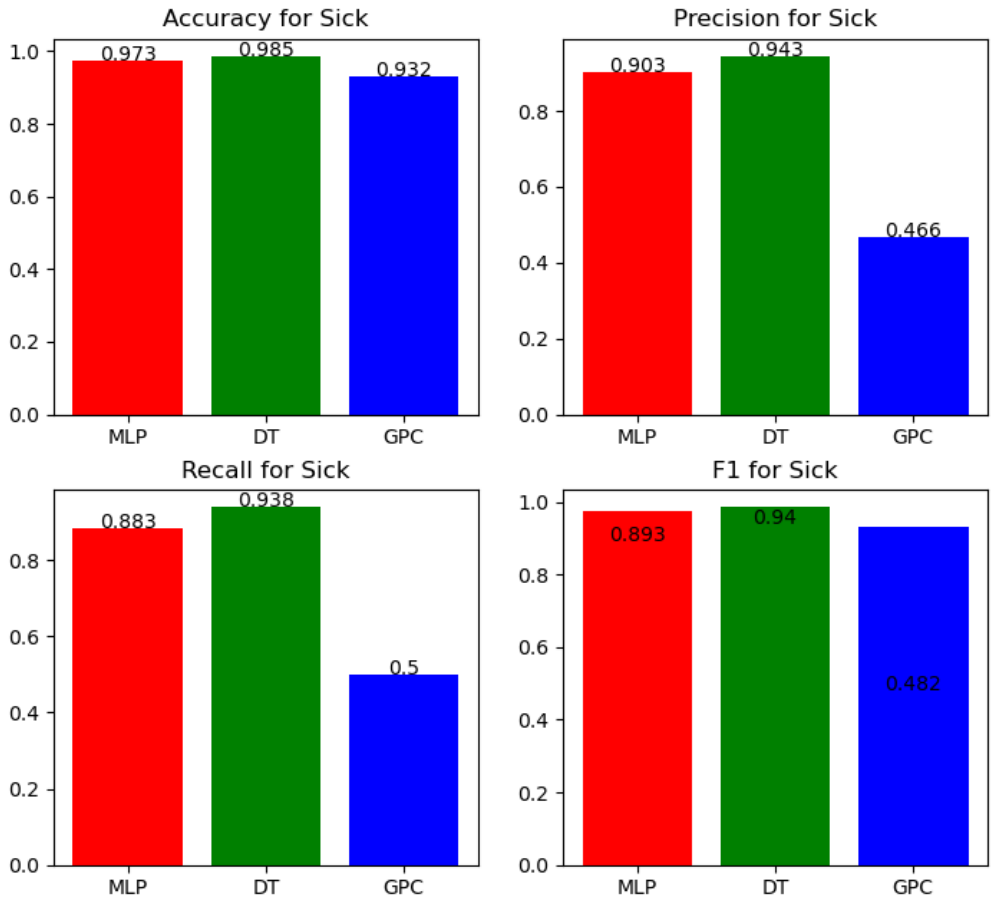
| Scaler | Kernel | Optimizer | Length Scale | Mean Accuracy | Std. Dev. |
|---|---|---|---|---|---|
| StdSc. | RBF | 'fmin_l_bfgs_b' | 1 | 0.95 | 0.00 |
| MinMaxSc. | Matern | 'fmin_l_bfgs_b' | 0.5 | 0.95 | 0.00 |
| RobustSc. | Rat.Quad. | 'fmin_l_bfgs_b' | 1.5 | 0.96 | 0.00 |

Holdout accuracies were 0.94, 0.94 and 0.95, respectively.
The parameters were chosen via RandomizedSearchCV on the grid

```
parameters = {
    "kernel": ('RBF', 'Matern', 'RationalQuadratic'),
    "length_scale : (0.001, 0.1, 1.0, 1.5, 2.0, 2.5),
    "optimizer": [None, 'fmin_l_bfgs_b']
}
```

### Comparison

We took the best performing models (in regards to accuracy) for each classifier on the Sick data set and compared them in regard to their accuracy, precision, recall and F1 scores. The result is shown in the graph below.




Training times were 0.348,0.068 and 14.392 seconds for MLP, DT and GPC, respectively.

## Data Set 3: Congressional Voting

### MLP

Table 5: Congress: MLP

| Scaler | Solver | $\alpha$ | Activation | Hidden Layers | Mean Accuracy | Std. Dev. |
|---|---|---|---|---|---|---|
| StandardScaler | 'lbfgs' | 1e-10 | 'logistic' | (20, 20, 20) | 0.97 | 0.04 |
| MinMaxScaler | 'lbfgs' | 1e-4 | 'logistic' | (15, 2) | 0.95 | 0.05 |
| RobustScaler | 'lbfgs' | 1e-0 | 'identity' | (15,2) | 0.97 | 0.04 |

Holdout accuracies were 0.955, 0.97 and 0.97, respectively.

The parameters were chosen via Grid Search on the same grid as for the Wine data set.

Further, the default SimpleImputer and OrdinalEncoder were used, to replace missing values with the column mean and encode nominal values, respectively.

### Decision Trees

TBA

### Gaussian Process Classifier

Table 6: Congress: GPC

| Scaler | Kernel | Optimizer | Length Scale | Mean Accuracy | Std. Dev. |
|---|---|---|---|---|---|
| Robust.Sc. | RBF | 'fmin_l_bfgs_b' | 0.1 | 0.97 | 0.05 |
| MinMaxSc. | Matern | None | 0.5 | 0.96 | 0.05 |
| Robust.Sc. | Rat.Quad. | 'fmin_l_bfgs_b' | 1.0 | 0.98 | 0.05 |

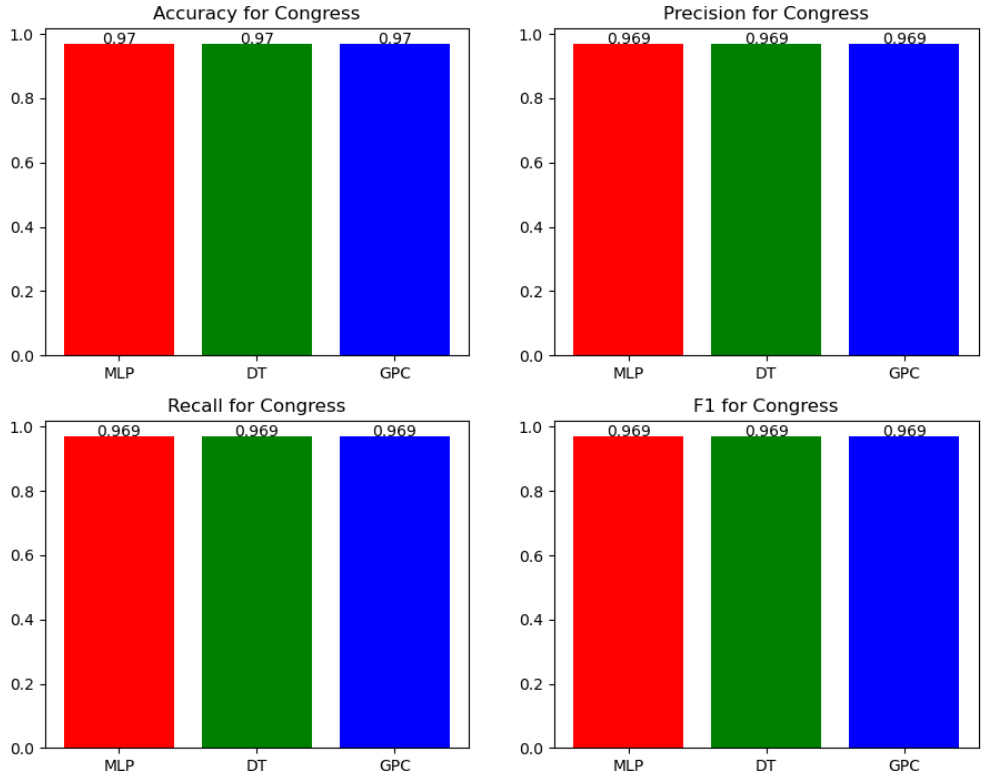
Holdout accuracies were 0.95, 0.94 and 0.96, respectively.

The parameters were chosen via RandomizedSearchCV on the grid

```
parameters = {
    "kernel": ('RBF', 'Matern', 'RationalQuadratic'),
    "length_scale : (0.1, 1.0, 1.5, 2.0),
    "optimizer": (None, 'fmin_l_bfgs_b')
}
```

### Comparison

We took the best performing models (in regards to accuracy) for each classifier on the Congress data set and compared them in regard to their accuracy, precision, recall and F1 scores. The result is shown in the graph below.




Training times were 0.028, 0.0135 and 0.097 seconds for MLP, DT and GPC, respectively.

## Data Set 4: Reviews

### MLP

Table 7: Reviews: MLP

| Scaler | Solver | $\alpha$ | Activation | Hidden Layers | Mean Accuracy | Std. Dev. |
|---|---|---|---|---|---|---|
| None | 'lbfgs' | 1e1 | 'identity' | (15, 15) | 0.54 | 0.05 |
| StandardScaler | 'adam' | 1e-1 | 'tanh' | (100,) | 0.61 | 0.06 |
| MinMaxScaler | 'adam' | 1e-1 | 'tanh' | (15, 2) | 0.63 | 0.04 |
| RobustScaler | 'lbfgs' | 1e1 | 'relu' | (100,) | 0.59 | 0.07 |

Holdout accuracies were 0.498, 0.596, 0.636 and 0.596, respectively.
The parameters were chosen via RandomizedSearchCV on the grid

```
parameters = {
    "activation": ('identity', 'logistic', 'tanh', 'relu'),
    "solver": ('lbfgs', 'sgd', 'adam'),
    "hidden_layer_sizes": ((15,2), (100,), (15,15), (20,3), (50, 50),
        (20, 20, 20), (100,100,100)),
    "alpha": np.logspace(-10, 4, 15),
```

```
}
```

due to the larger data set size making a full grid search too costly.

### Gaussian Process Classifier

In a first approach, normal GPC was used for the classification task, resulting in a cross-validation accuracy of 0.59 and a holdout accuracy of 0.52 (Kernel: DotProduct(sigma_0=1), Scaler: MinMaxScaler) However, since the most significant computational challenge in Gaussian process classification tasks is inversion of the covariance matrix $K_{ij} = k(x_i, x_j)$ (with a time complexity of $\mathcal{O}(n^3)$, where $n$ defines the number of training points), the computation took some days to complete. Therefore, GPC is not feasible for large datasets. In order to reduce the computational costs, an approximation method, namely Sparse Gaussian Classification, was used.

Table 8: Reviews: Sparse GPC

| Scaler | Kernel | n_iter | Samples | Mean Accuracy | Std. Dev. |
|---|---|---|---|---|---|
| MaxAbsScaler | RBF | 100 | 100 | 0.01 | 0.006 |
| StandardScaler | Matern | 200 | 700 | 0.01 | 0.006 |
| MinMaxScaler | Matern | 300 | 700 | 0.01 | 0.006 |

The parameters for Sparse GPC were chosen via RandomizedSearchCV on the grid:

```
parameters = {
    "kernels" = ('RBF', 'Matern')
    "scalers" = (StandardScaler, MinMaxScaler, RobustScaler, MaxAbsScaler)
    "n_iterations" = (100, 200, 300, 500, 1000)
    "number_samples" = (100, 500, 700)
    )
}
```

In the sparse Gaussian Process Classifier implementation, the use of a limited number of inducing points (e.g., 700) significantly reduces the computational cost by approximating the full covariance matrix. However, this also limits the model's capacity to capture complex patterns in the data, leading to a lower cross-validation accuracy. Furthermore, when attempting to increase the number of inducing points to 1000 for better approximation, the kernel computation became infeasible due to the high memory and computational demands of handling larger subsets of data, resulting in a kernel crash. This highlights the trade-off between computational efficiency and model accuracy in sparse GPCs.

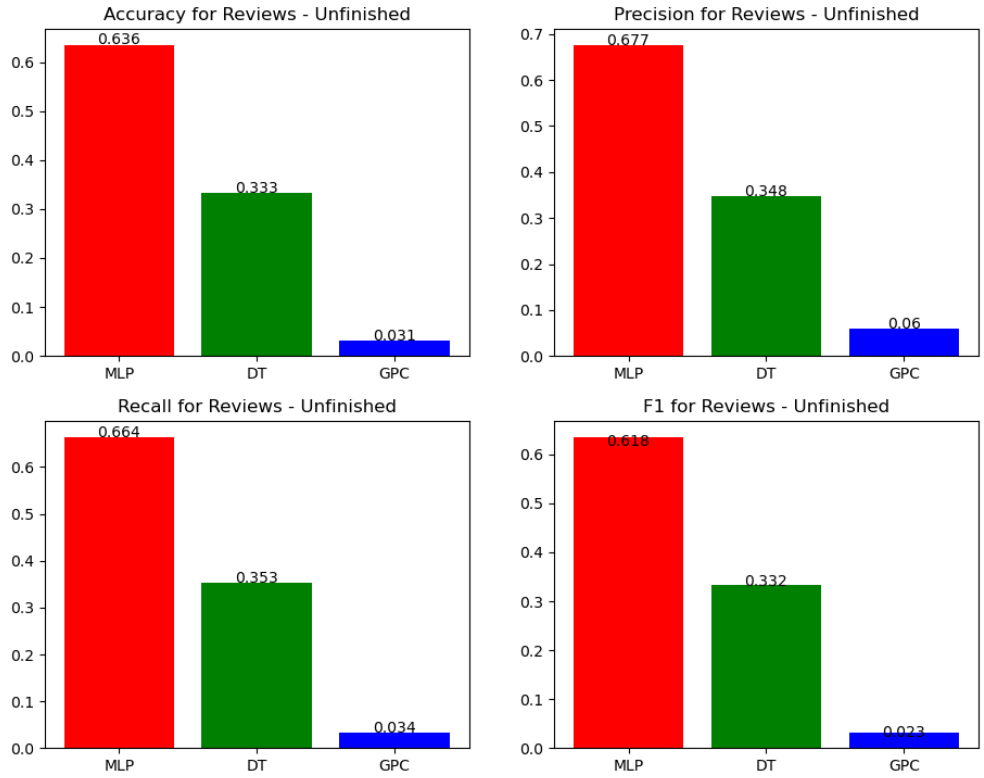
### Decision Trees

TBA

### Comparison

We took the best performing models (in regards to accuracy) for each classifier on the Congress data set and compared them in regard to their accuracy, precision, recall and

F1 scores. The result is shown in the graph below. Note that an older version of the sparse GPC was used in this comparison.




Training times were 26.730, 2.368 and 27.180 seconds for MLP, DT and GPC, respectively.

# Conclusion

All three classifiers we tried performed very similar for the first three data sets in regard to accuracy. They also mostly coincided in the other scores measured, aside from GPC, which had markedly lower precision and recall scores on the sick data set.

We can see that DTs were consistently the fastest to train, followed by MLPs and then GPCs. For larger instances especially, training times for GPCs and larger MLPs skyrocketed, while DTs and small MLPs remained manageable.

From this we can conclude that GPC seem to be well suited for smaller data sets, where it achieved good results in reasonable time. DT achieved excellent results on most data sets together with the shortest training times, but struggled to model the reviews data set. Finally, MLP performed mostly middle of the road, but achieved the best results on the reviews data set.