Machine Learning

Activity recognition using performance metrics

Thodoris Goulas

Abstract

- Establish and evaluate a model, predicting performed activity type, based on a comprehensive set of performance metrics (features)

- Activities classes-labels (categorical type):
  - Running
  - Biking
  - Swimming
  - Hiking
  - Cardio workout
  - Strength training

- Classification algorithms will be used

Training Data

- Garmin Connect application will be used, exporting training data in json file


```json
{
    "activeSets": 1,
    "activityId": 5843171487,
    "activityType": "strength_training",
    "aerobicTrainingEffect": 0.0,
    "aerobicTrainingEffectMessage": "NO_AEROBIC_BENEFIT_18",
    "anaerobicTrainingEffect": 0.0,
    "anaerobicTrainingEffectMessage": "NO_ANAEROBIC_BENEFIT_0",
    "atpActivity": false,
    "autoCalcCalories": false,
    "avgFractionalCadence": 0.0,
    "avgHr": 68.0,
    "avgSpeed": 0.0,
    "beginTimestamp": 1605558640000,
    "calories": 155.03073999999998,
    "deviceId": 3972498287,
    "distance": 0.0,
    "duration": 1172858.0322265625,
    "elapsedDuration": 1172858.0322265625,
    "elevationCorrected": false,
    "eventTypeId": 9,
    "favorite": false,
    "lapCount": 1,
    "manufacturer": "GARMIN",
    "maxFractionalCadence": 0.0,
    "maxFtp": 197.0,
    "maxHr": 87.0,
    "maxTemperature": 22.0,
    "minTemperature": 17.0,
    "movingDuration": 1172858.0322265625,
    "name": "Strength",
    "parent": false,
    "pr": false,
    "purposeful": false,
    "rule": "subscribers",
    "sportType": "TRAINING",
    "startTimeGmt": 1605558640000.0,
    "startTimeLocal": 1605565840000.0,
    "steps": 8.0,
    "summarizedDiveInfo": {},
    "summarizedExerciseSets": [
        {
            "category": "UNKNOWN",
            "duration": 1172858.0322265625,
            "reps": 4,
            "sets": 1,
            "volume": 0
        }
    ],
    "timeZoneId": 473,
    "totalReps": 4,
    "totalSets": 1,
    "userProfileId": 81201032,
    "uuidLsb": -7410265071826486760,
    "uuidMsb": 1621105547867541023
},
```

Data Set

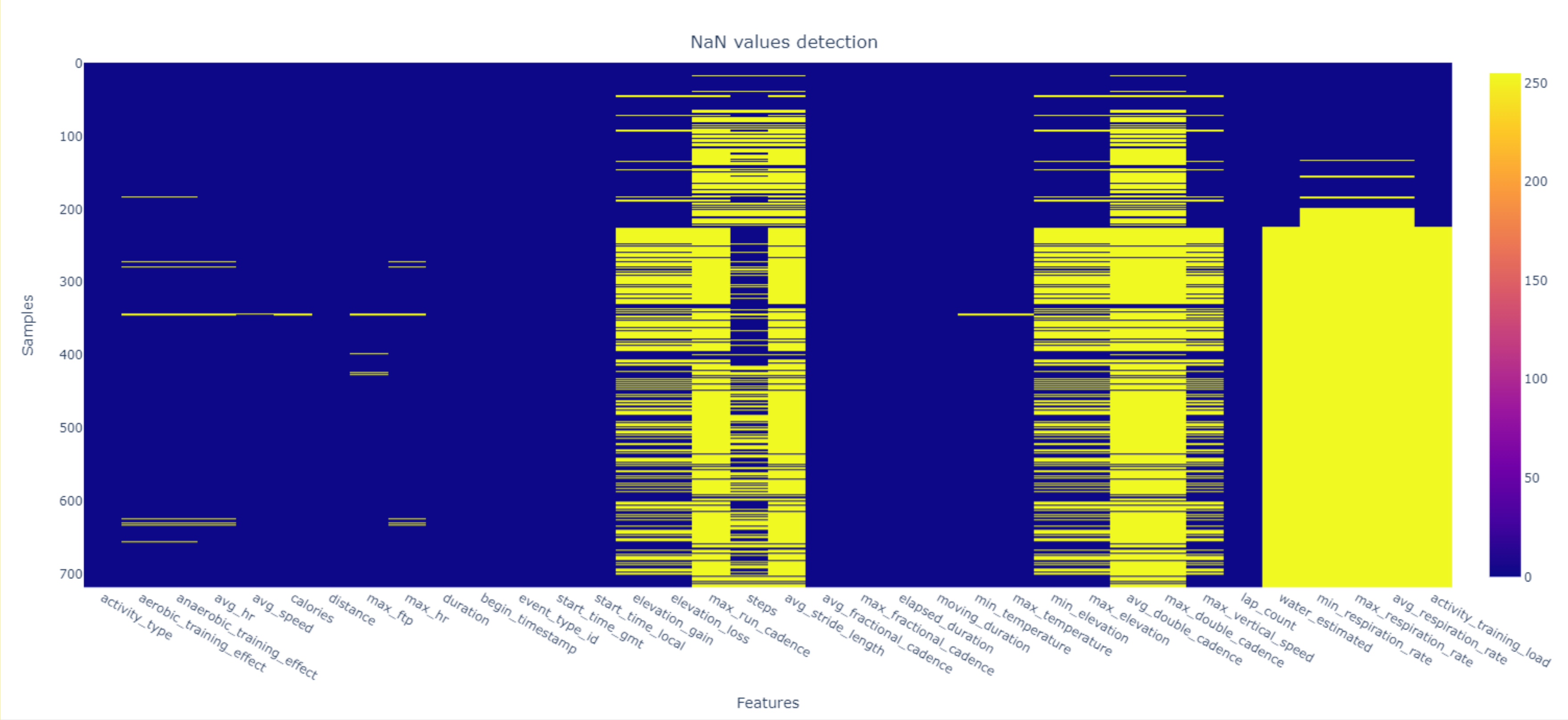
- Data set consisted by about 750 activities will be used as training data
- This data set collected by personal use of Garmin Connect application for over a year
- 10 k-fold stratified cross-validation was used.
- Final train-test scores calculated by the average of each fold score.
- **sklearn.model_selection.GridSearchCV** library was used

Data Visualization

- 34 features were extracted for each example
- NaN value heapmap, correlation matrix and feature boxplots were implemented
- **plotly** library used

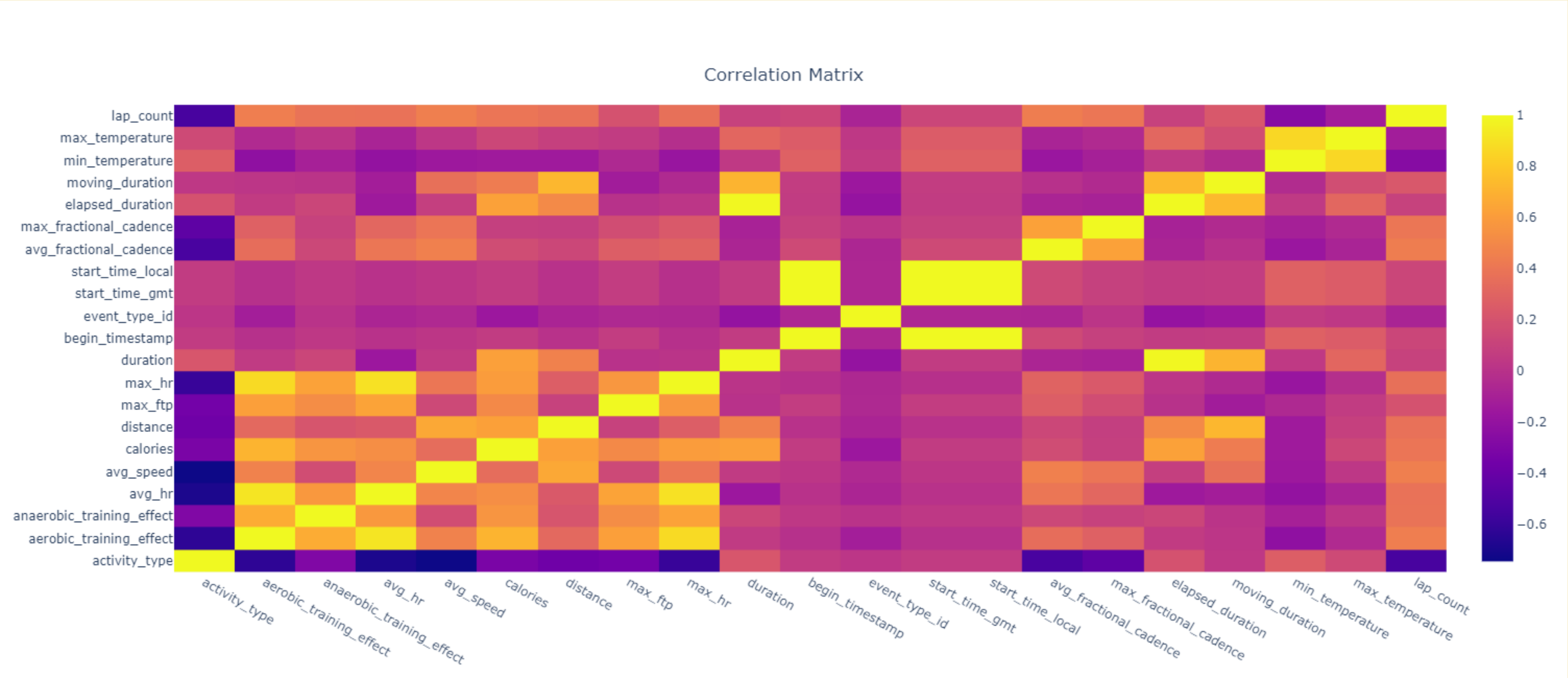
Data Visualization

- Removed features with multiple NaN values



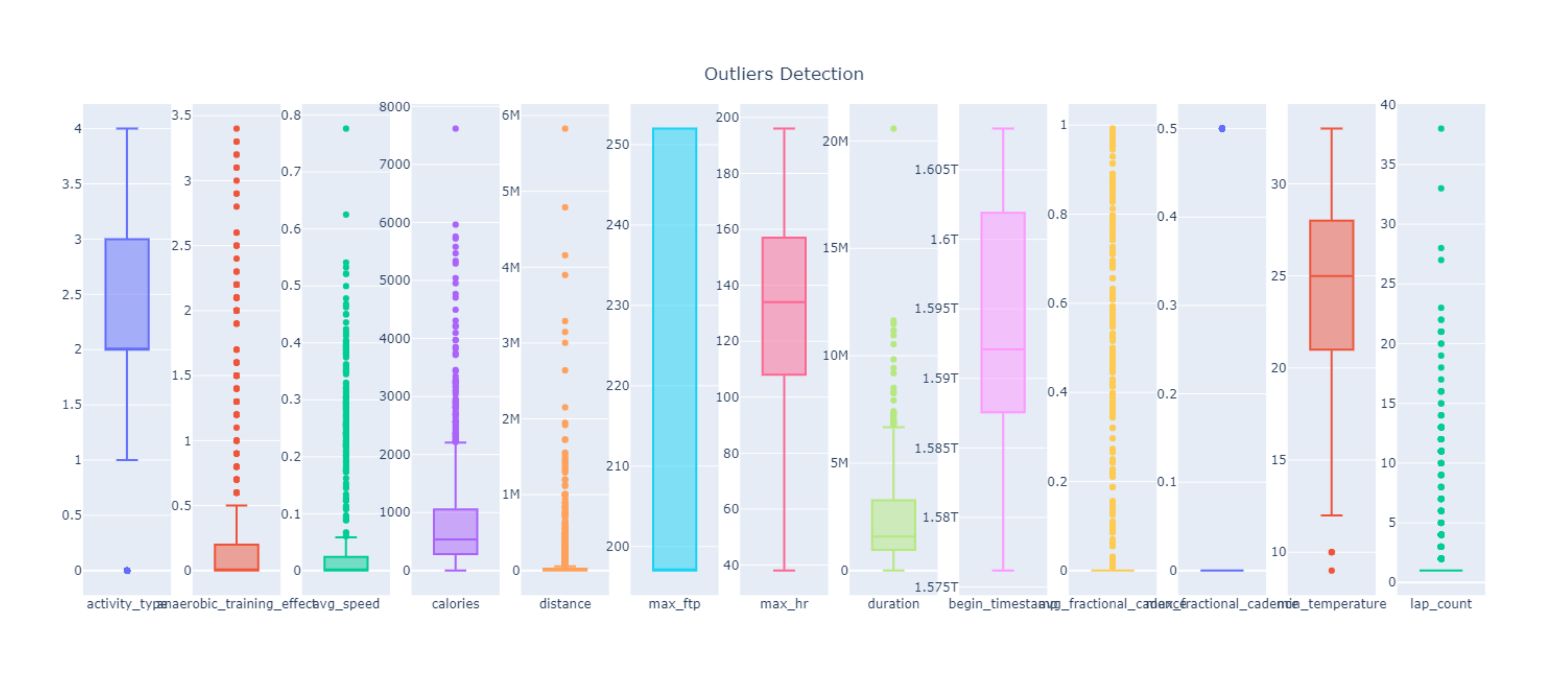
Data Visualization

- Removed features with low correlation with activity type
- Removed features highly correlated with others



Data Visualization

- Boxplots provide useful information about feature values distribution and outliers detection



Data Preprocessing

- Removed samples containing NaN values

1. changes range
2. preserves the shape of the original distribution
3. doesn't reduce the importance of outliers

- ssing library used to normalize the extracted
- preprocessing algorithms:

reduces the effects of outliers

1. makes the mean of the distribution 0
2. about 68% of the values will lie be between -1 and 1
3. distorts the relative distances between the feature values

1. applied in rows and not in columns
2. each row values have a unit norm (if squared and summed, the total would equal 1)
3. transform all the features to values between -1 and 1

  - MinMaxScaler: su … an from each value and then divides by the ran
  - … then dividing by the interquartile … — 25% value)
  - StandardScaler: subtracts the mean and then scaling to unit variance
  - Normalizer: L2 normalization
  - Hybrid: Combination of previous algorithms

Data Preprocessing

- After data preprocessing the following features were selected:
  - anaerobic_training_effect
  - avg_speed
  - calories
  - distance
  - max_ftp
  - max_hr
  - Duration
  - begin_timestamp
  - avg_fractional_cadence
  - max_fractional_cadence
  - min_temperature
  - lap_count
- Dataset size: 705 samples

Classification Algorithms

- Logistic Regression
- K-Nearest Neighbors
- Naive Bayes
- Decision Trees
- Support Vector Machine
- (Single-layer) Perceptron

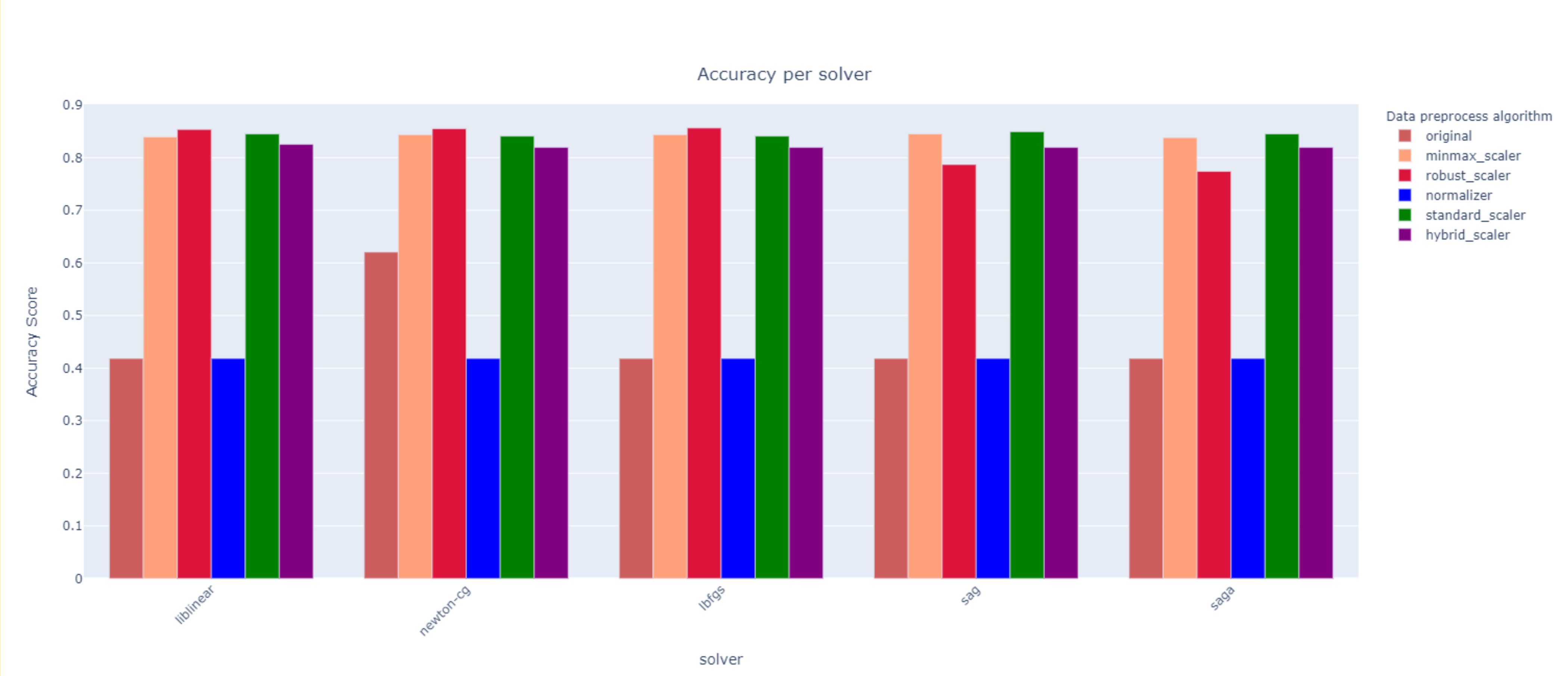
Logistic Regression

- the probabilities describing the possible outcomes of a single trial are modelled using a logistic function
- several classes involved ➡ multiclass logistic regression was preformed
- **sklearn.linear_model** library was used
- tric was used for trained model performance evaluation
- eters:
  - *solver*: all available solvers compared
  - C: the penalty parameter, which represents misclassification or error term – ed C = 0.1, 1, 10, 100

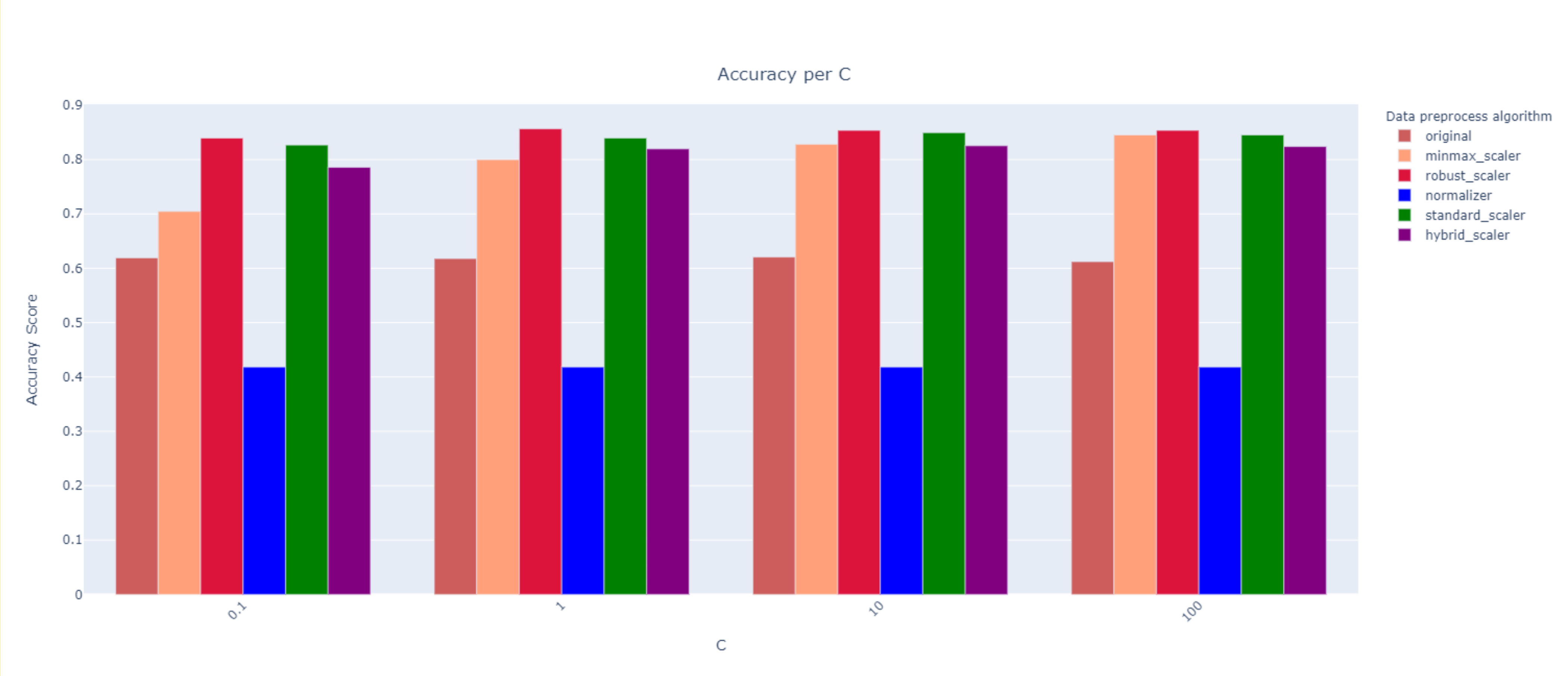
Algorithm used to find the global likelihood function maximum at Gradient Ascent

low C ->
Higher C ->

Logistic Regression – Accuracy per solver



Logistic Regression – Accuracy per C value



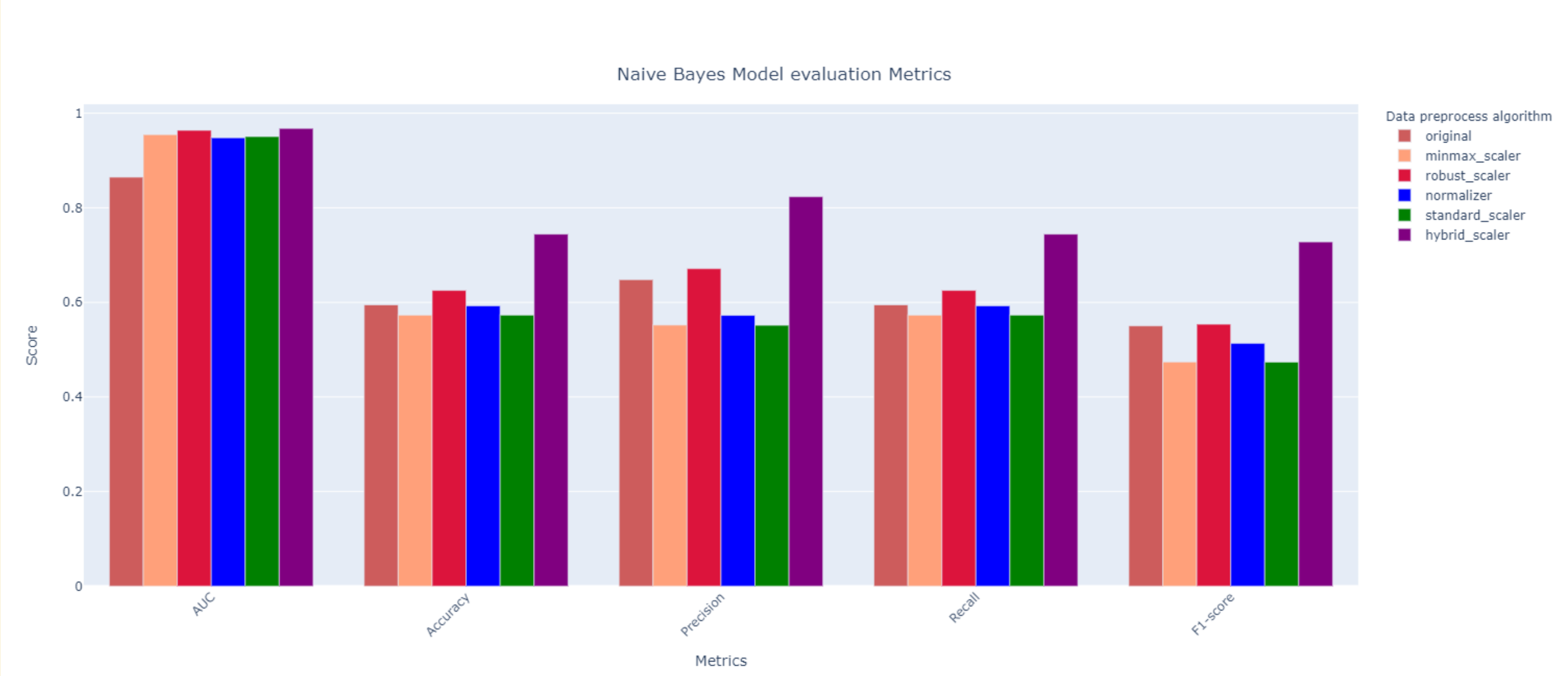
Logistic Regression – Optimal Parameters

- solver: lbfgs
- C: 1.0
- data preprocessing policy: robust_scaler
- Accuracy score: 0.8566

Gaussian Naive Bayes

- based on Bayes' theorem
- assumption of independence between every pair of features -> highly correlated features should be removed
- assumption of Gaussian distribution of joint probability $p(x|y)$
- ***sklearn.naive_bayes*** library was used
- Gaussian Naive Bayes classifier used
- stratified 10 fold cross-validation was performed
- accuracy, precision, recall, F1-measure metrics were used for trained model performance evaluation

Gaussian Naive Bayes

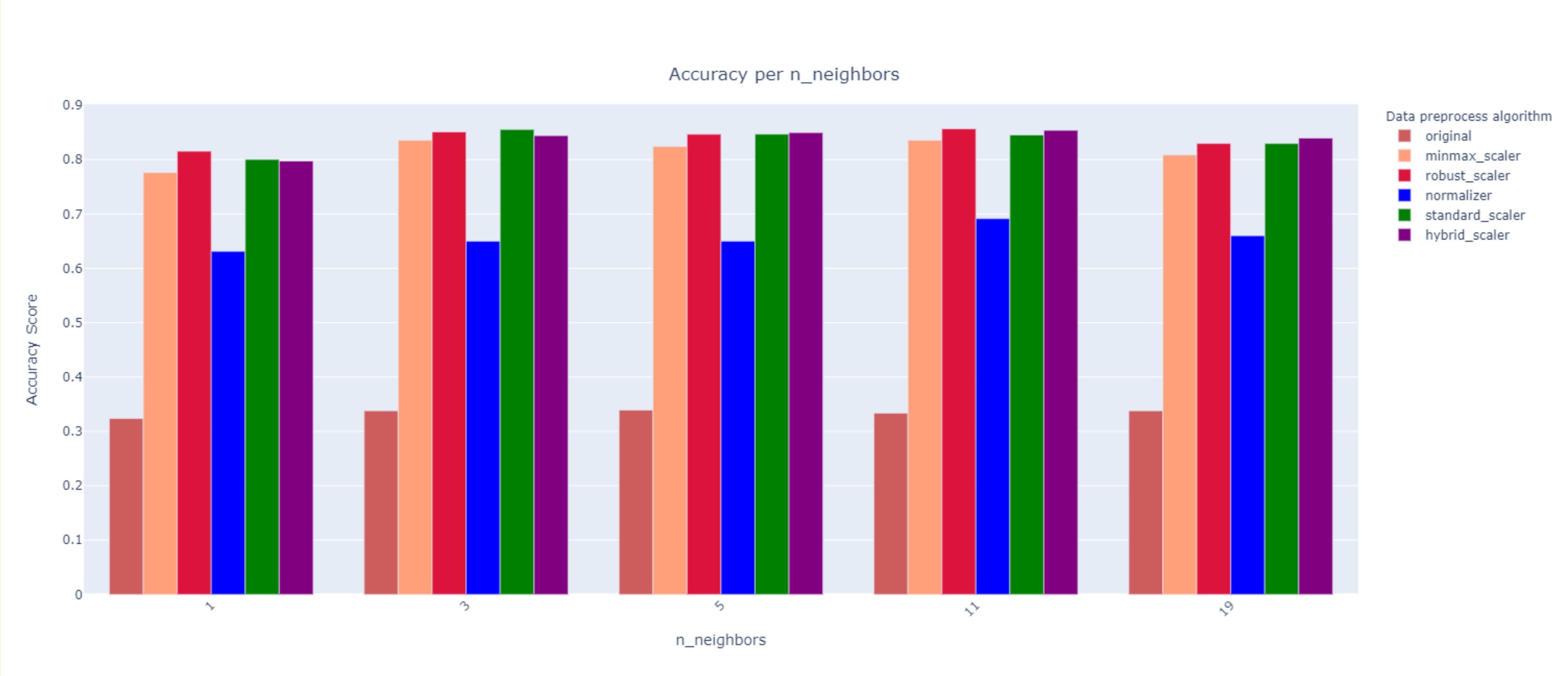


k-Nearest Neighbors

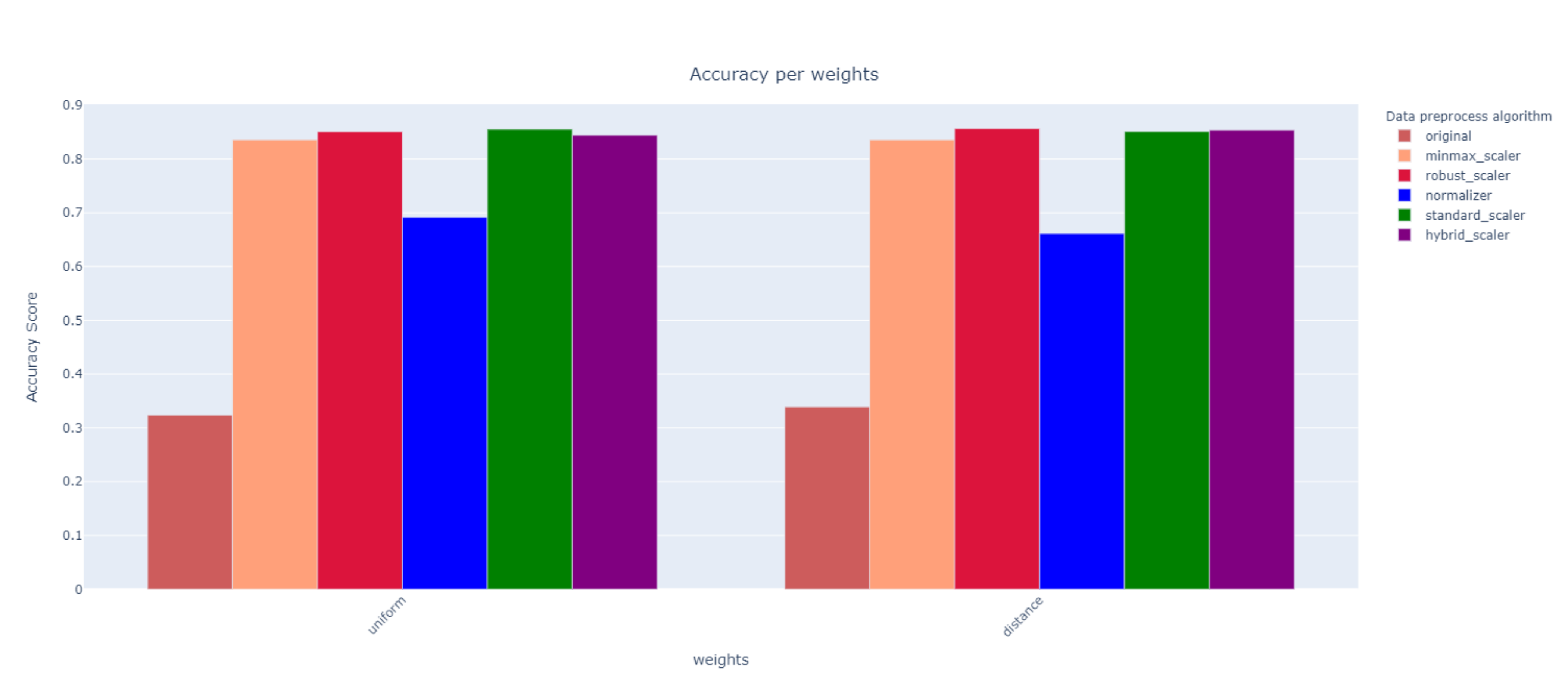
- classification over k-nearest neighbors majority vote
- **sklearn.neighbors** library was used
- stratified 10 fold cross-validation was performed
- accuracy metric was used for trained model performance evaluation

- Hyper-parameters:
  - **k_neighbors**: defines the number of nearest neighbors, used to classify the unknown point - used: k=1,3,5,11,19
  - **weights**:
    - uniform -> each neighbor 'vote' within the boundary carries the same weight
    - distance -> closer points will be more heavily weighted toward the decision
  - **metric**: the algorithm used to calculate the distance of neighboring points is chosen from the unknown point
- used Euclidean and Manhattan distances

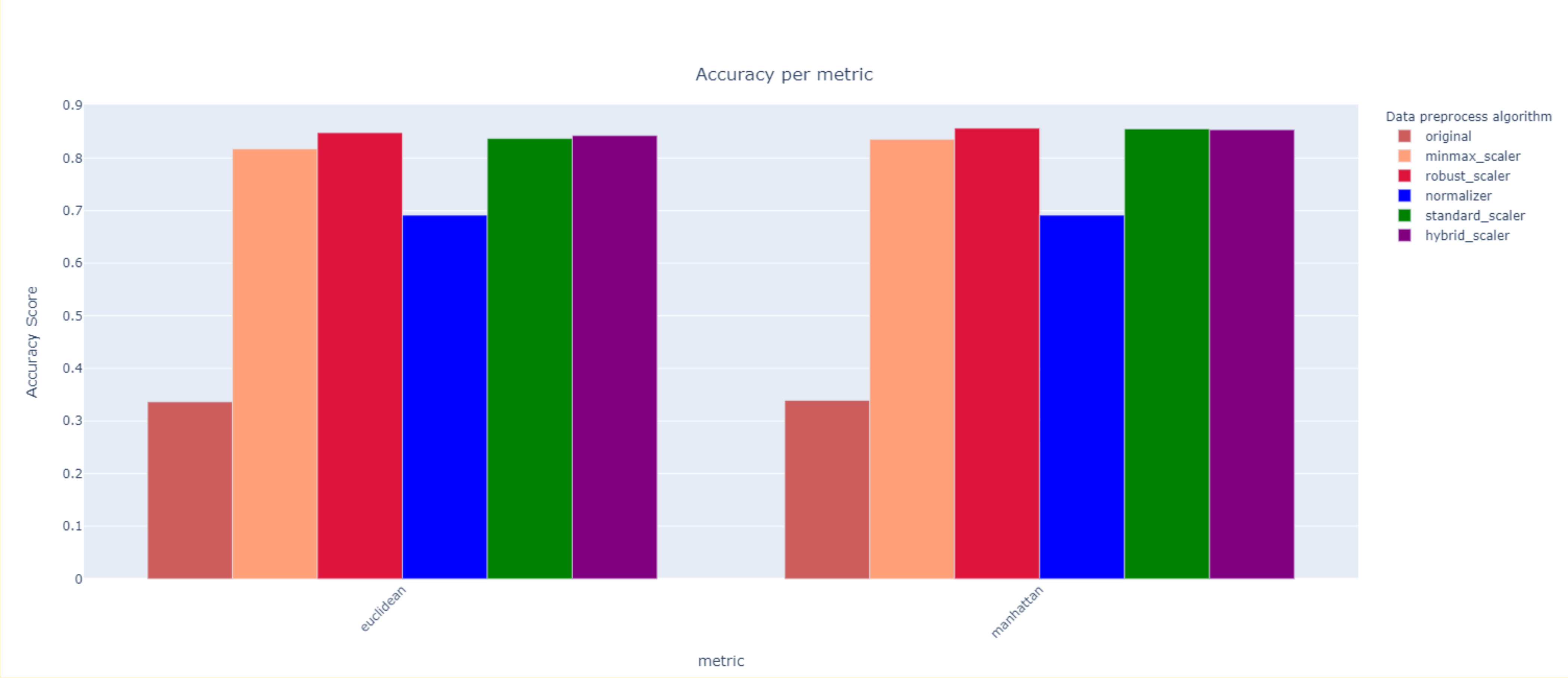
k-Nearest Neighbors – Accuracy per n_neighbors



k-Nearest Neighbors – Accuracy per weights



k-Nearest Neighbors – Accuracy per metric



k-Nearest Neighbors – Optimal Parameters

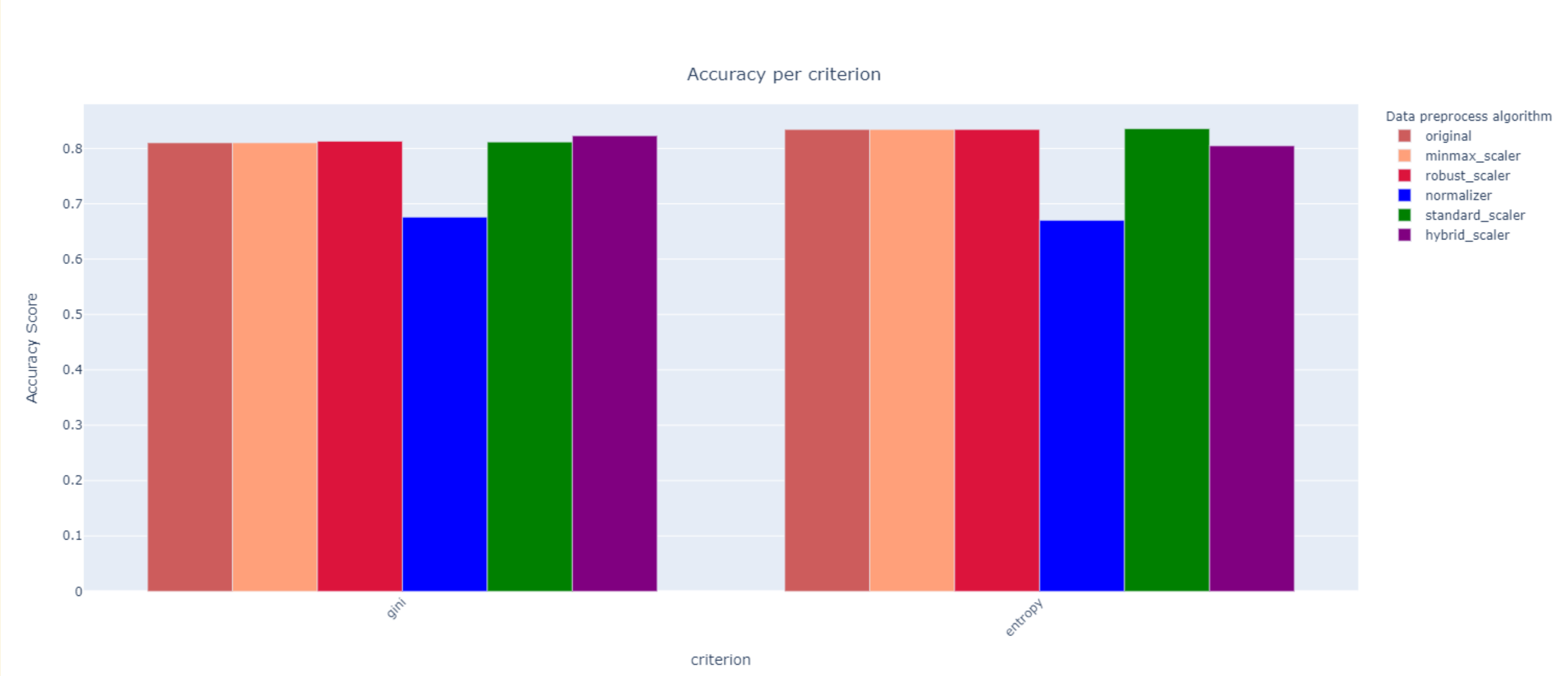
- n_neighbors: 11
- weights: distance
- metric: manhattan
- data preprocessing policy: robust_scaler
- Accuracy score: 0.8568

Decision Trees

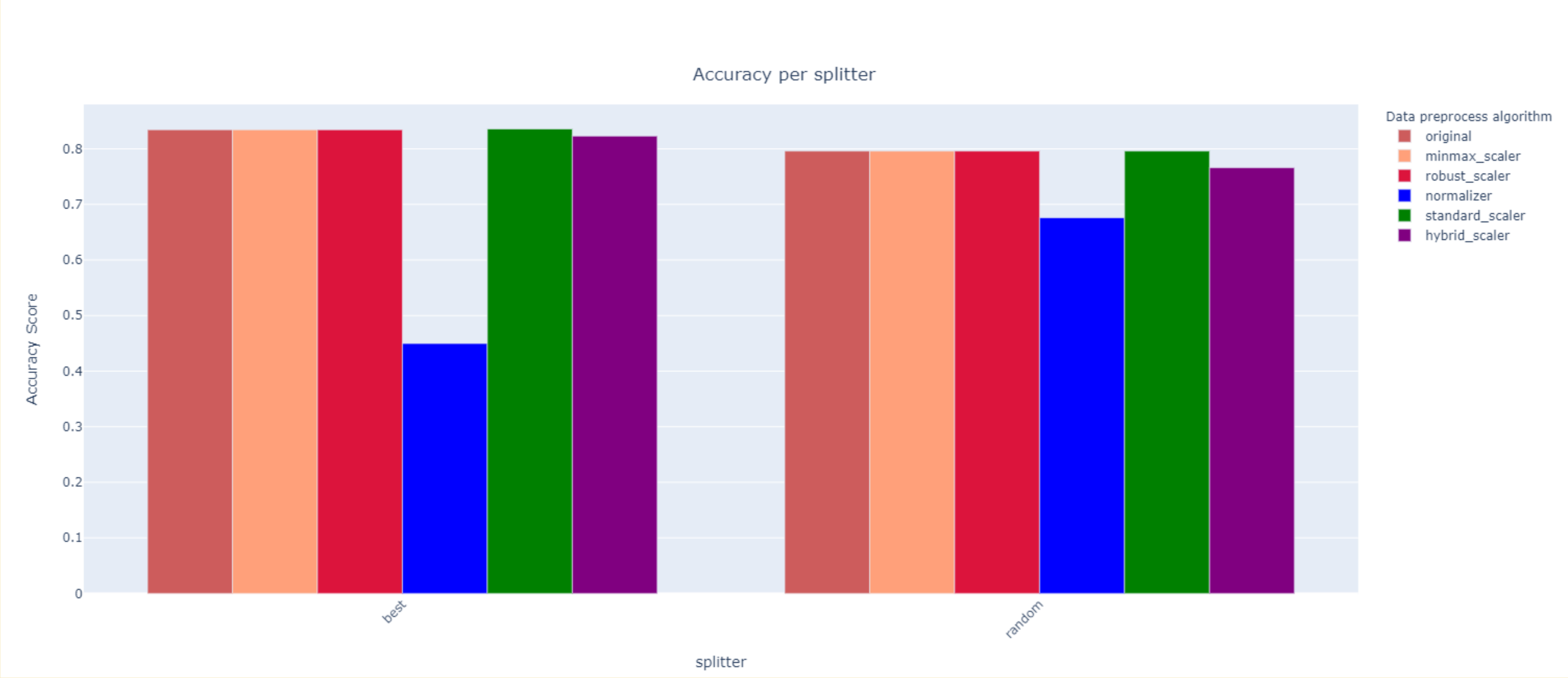
- Classification model built in the form of a tree structure
- ***sklearn.tree*** library was used
- stratified 10 fold cross-validation was performed
- accuracy metric was used for trained model performance evaluation

- Hyper-parameters:
  - ***criterion***: defines the function to measure the information gain - used: gini, entropy
  - ***splitter***: the strategy used to choose the split at each node
    - best -> best split among all is chosen
    - random -> best split among a random subset is chosen
  - ***max_depth***: the maximum depth of the tree, if None -> nodes are expanded until all leaves are pure - used None, 1, 5, 10, 15, 20, 25

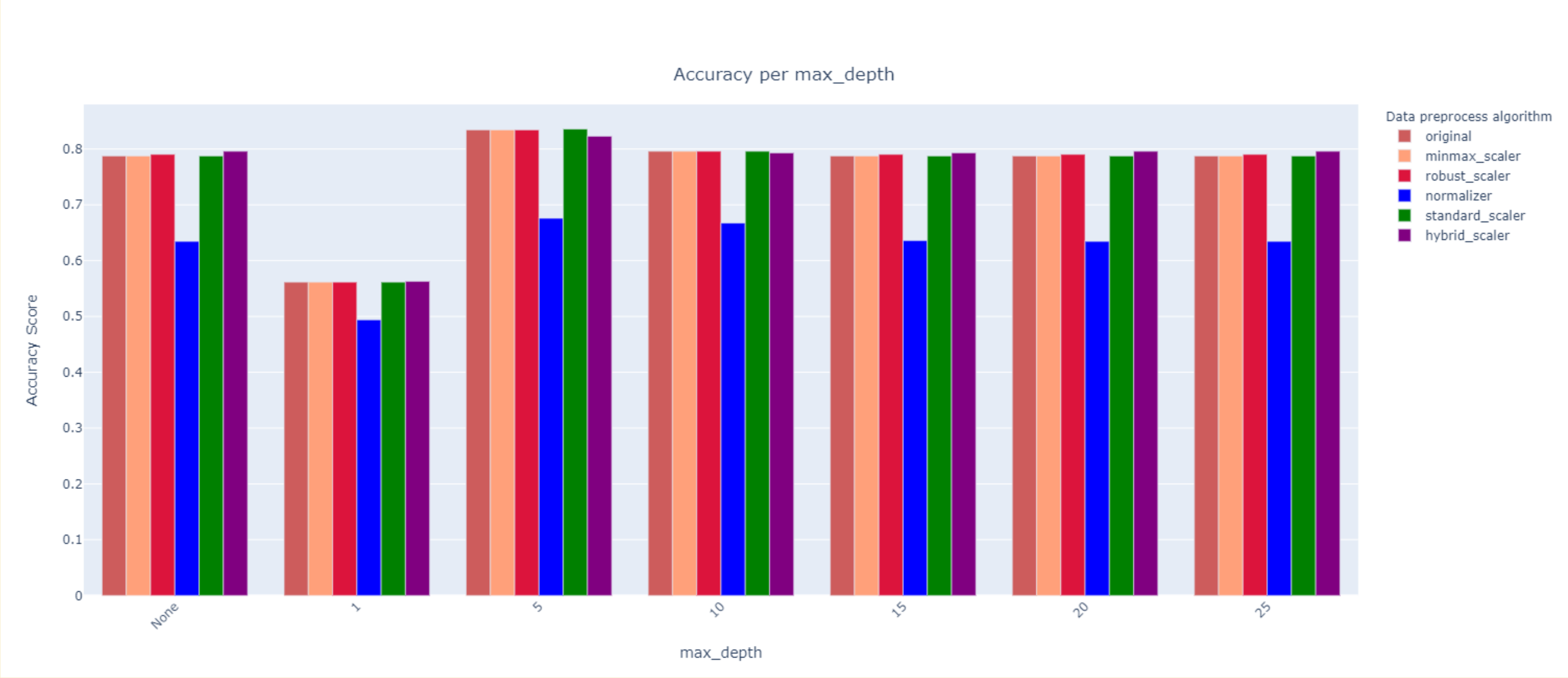
Decision Trees – Accuracy per criterion



Decision Trees – Accuracy per splitter



Decision Trees – Accuracy per max_depth



Decision Trees – Optimal Parameters

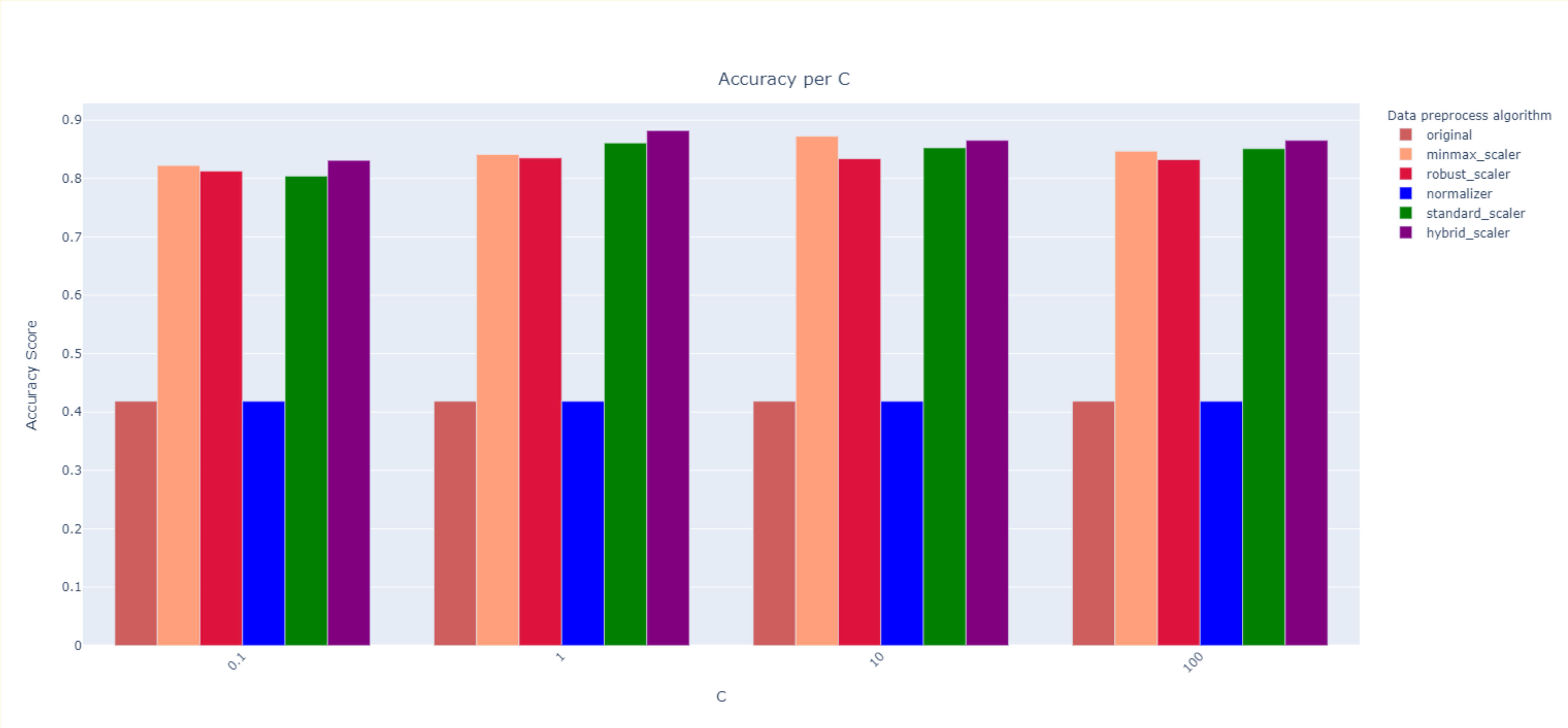
- criterion: entropy
- max_depth: 5
- splitter: best
- data preprocessing policy: standard_scaler
- Accuracy score: 0.8358

Support Vector Machine (SVM)

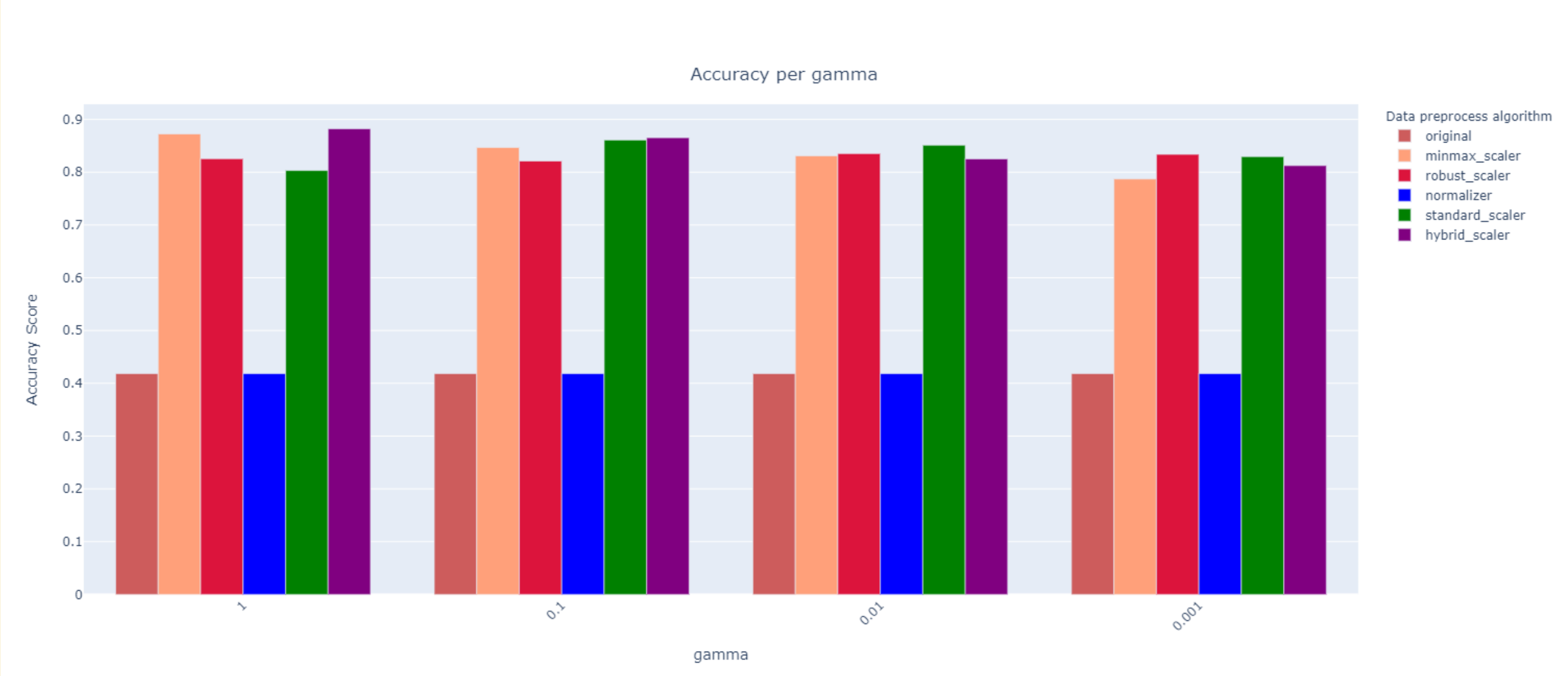
- Training data represented as points in space
- Classification achieved by defining a decision surface that separates the classes
- Goal: maximize the margin between the data points of the two classes
- ***sklearn.svm.SVC*** library was used
- stratified 10 fold cross-validation was performed
- accuracy metric was used for trained model performance evaluation

- Hyper-parameters:
  - ***kernels***: the function used to transform low dimensional input space into a higher-dimensional space
    - poly -> applying the polynomial combination of all the existing features (e.g. x^2)
    - rbf -> generates new features by measuring the distance between all other dots to a specific dot/dots centers
    - sigmoid -> sigmoid function is used to transform features
  - **C**: the penalty parameter, which represents misclassification or error term - used C = 0.1, 1, 10, 100
  - ***gamma***: defines how far influences the calculation of plausible line of separation - used g = 1, 0.1, 0.01, 0.001

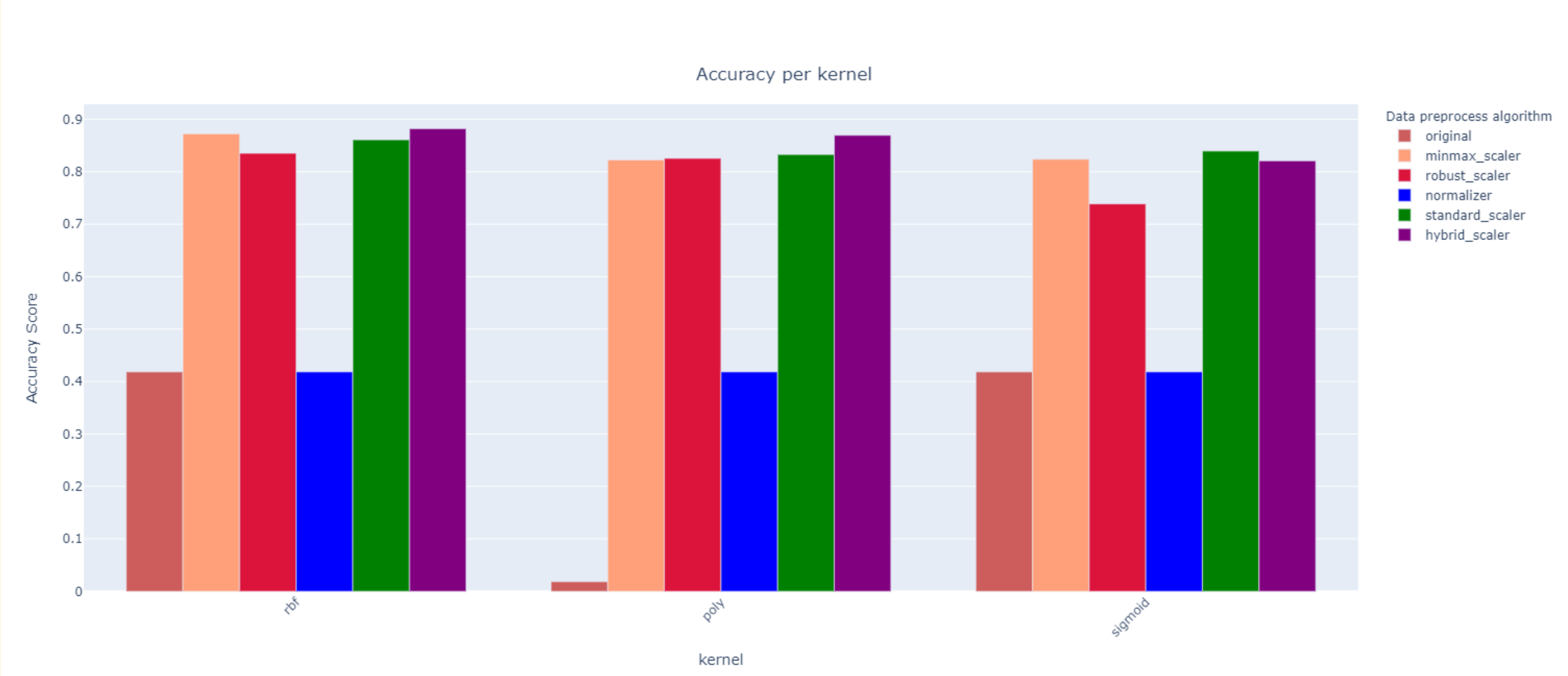
SVM – Accuracy per C



SVM – Accuracy per gamma



SVM – Accuracy per kernel



SVM – Optimal Parameters

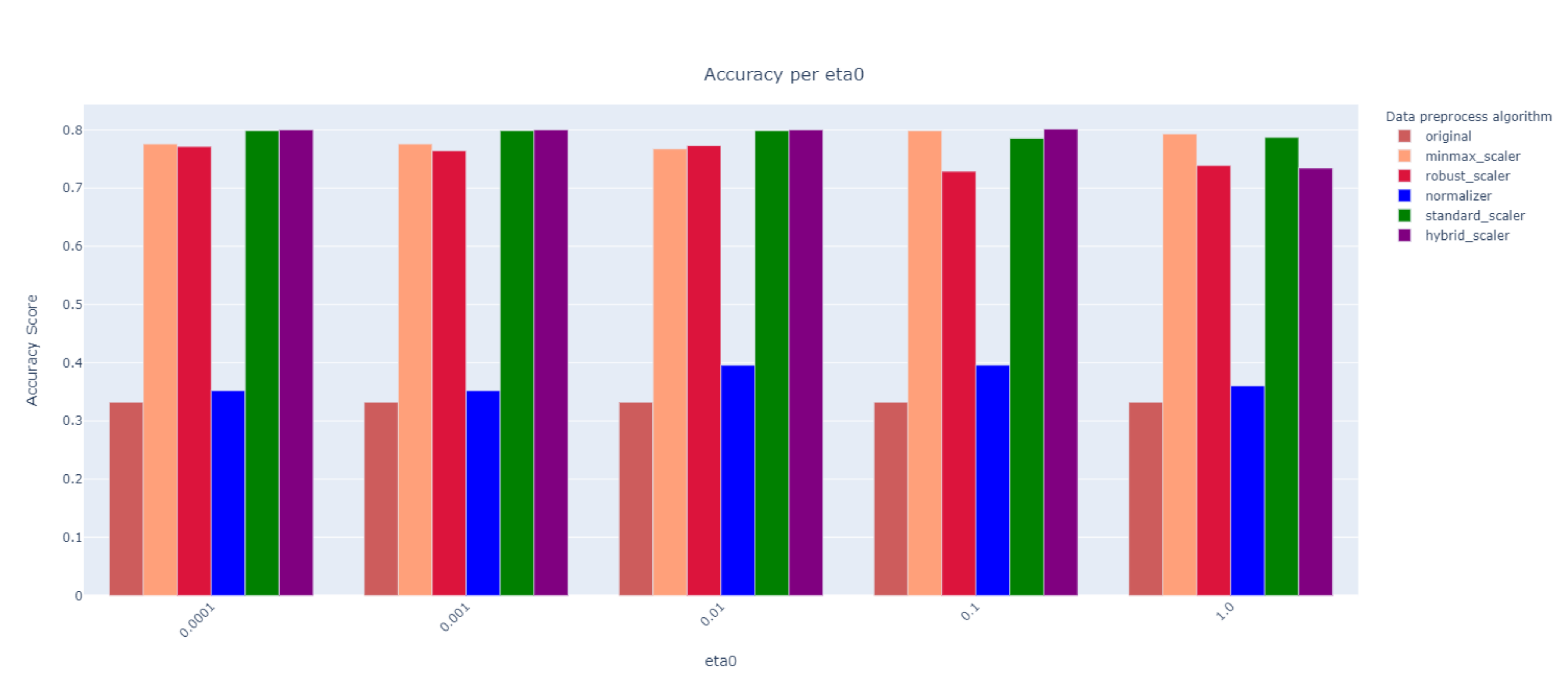
- C: 1
- gamma: 1
- kernel: rbf
- data preprocessing policy: hybrid_scaler
- Accuracy score: 0.8822

(Single-layer) Perceptron

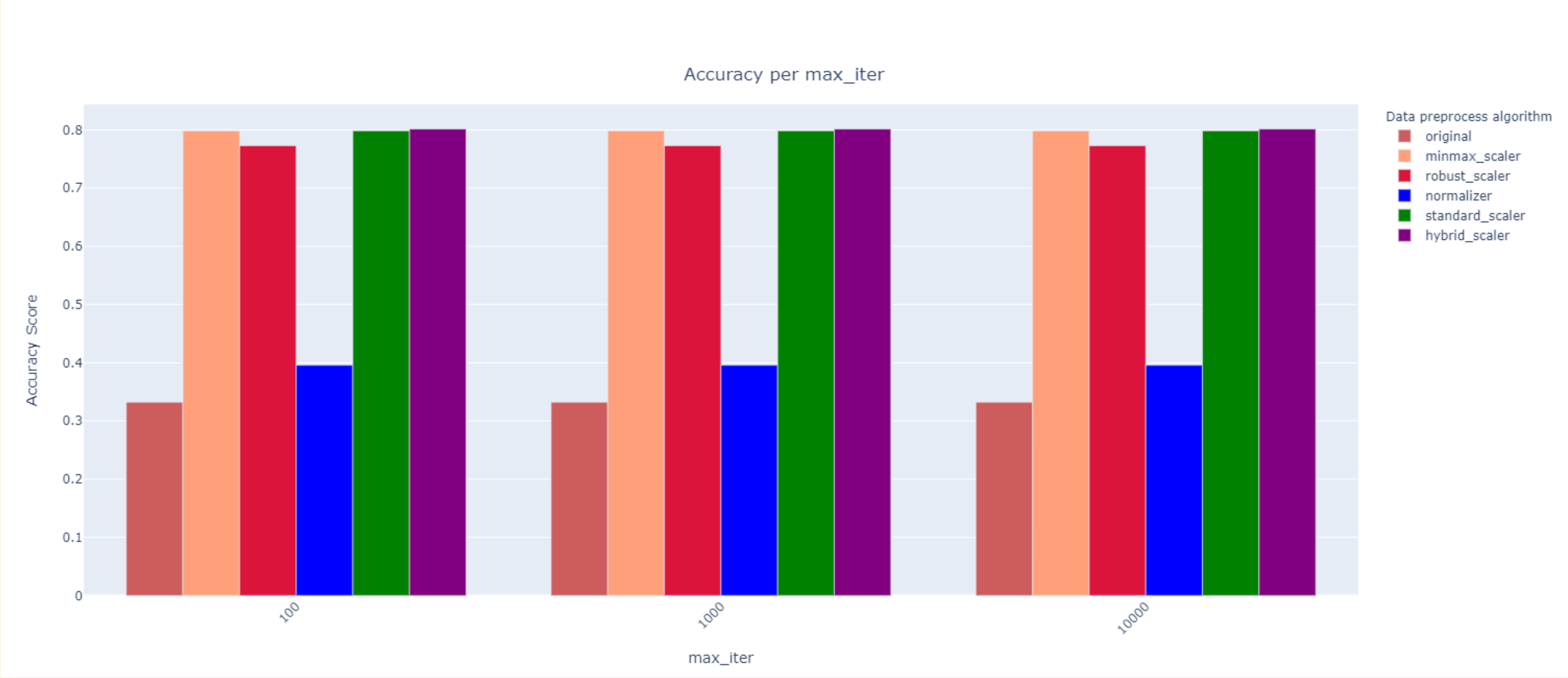
- single layer Perceptron is the basic unit of a neural network
- binary classification machine learning algorithm
- multi-class classification can be implemented by one-vs-all policy
- transforms the original feature space by multiplying feature vectors with the respective weights
- aims to define a decision hyperplane where the classes are fully separated (using an activation function -> sigmoid)
- is actually an SVM classifier without the soft-margin concept
- **sklearn.linear_model** library was used
- stratified 10 fold cross-validation was performed
- accuracy metric was used for trained model performance evaluation

- Hyper-parameters:
  - ***learning rate (eta0)***:
    - high learning rate -> fast training ~ lower performance,
    - low learning rate -> slow training ~ better performance,
    - used 0.0001, 0.001, 0.01, 0.1, 1
  - ***max iterations***: the number of epochs used to train the model - used 100, 1000, 10000

Perceptron – Accuracy per learning rate



Perceptron – Accuracy per max iteration



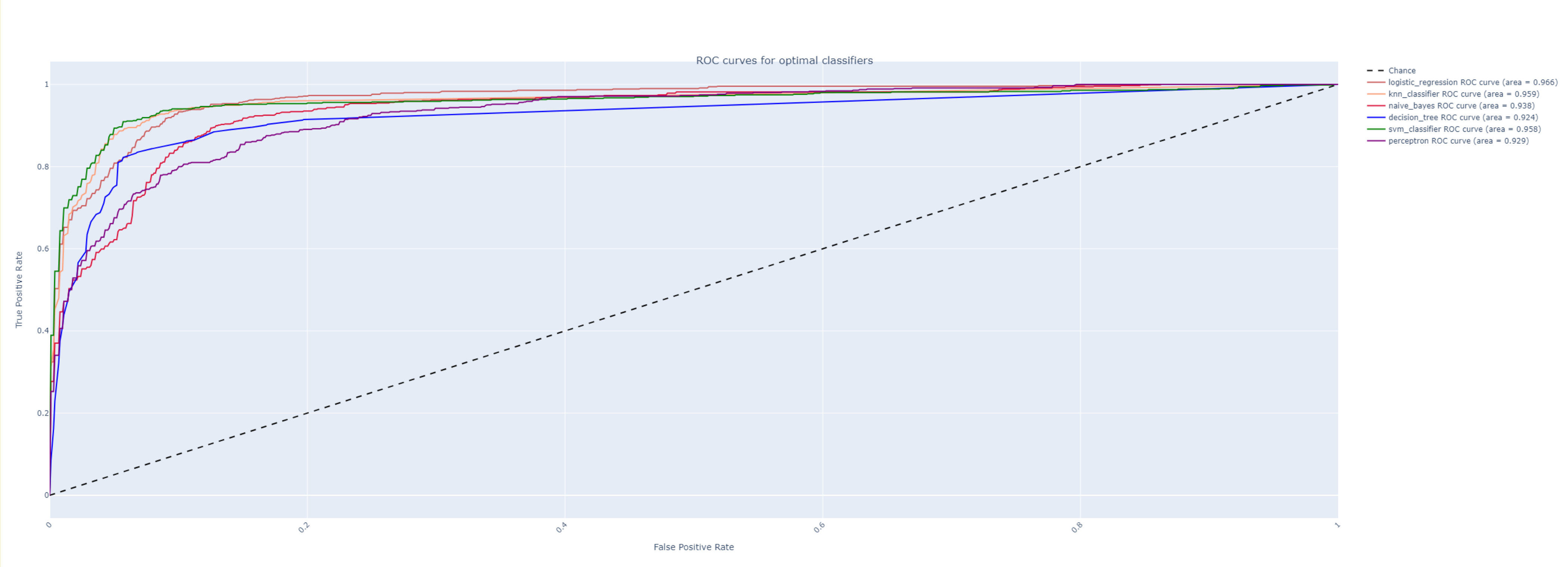
Perceptron – Optimal Parameters

- eta0: 0.1
- max_iter: 100
- data preprocessing policy: hybrid_scaler
- Accuracy score: 0.8015

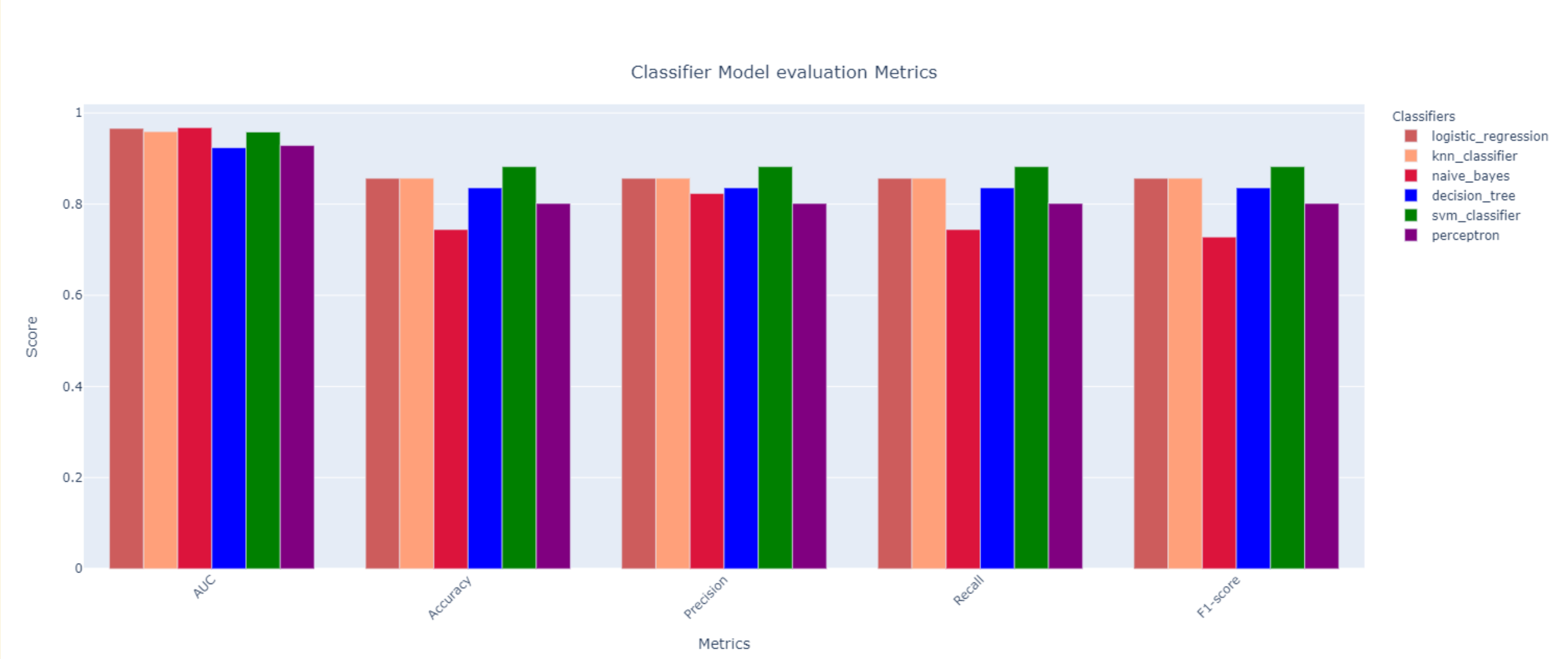
Classifiers Comparison

- Optimal parameters and best data preprocess policy chosen per classifier
- Implemented ROC curves per classifier
- Accuracy, Precision, Recall, F1-score and AUC scores compared
- Model Fit time also compared

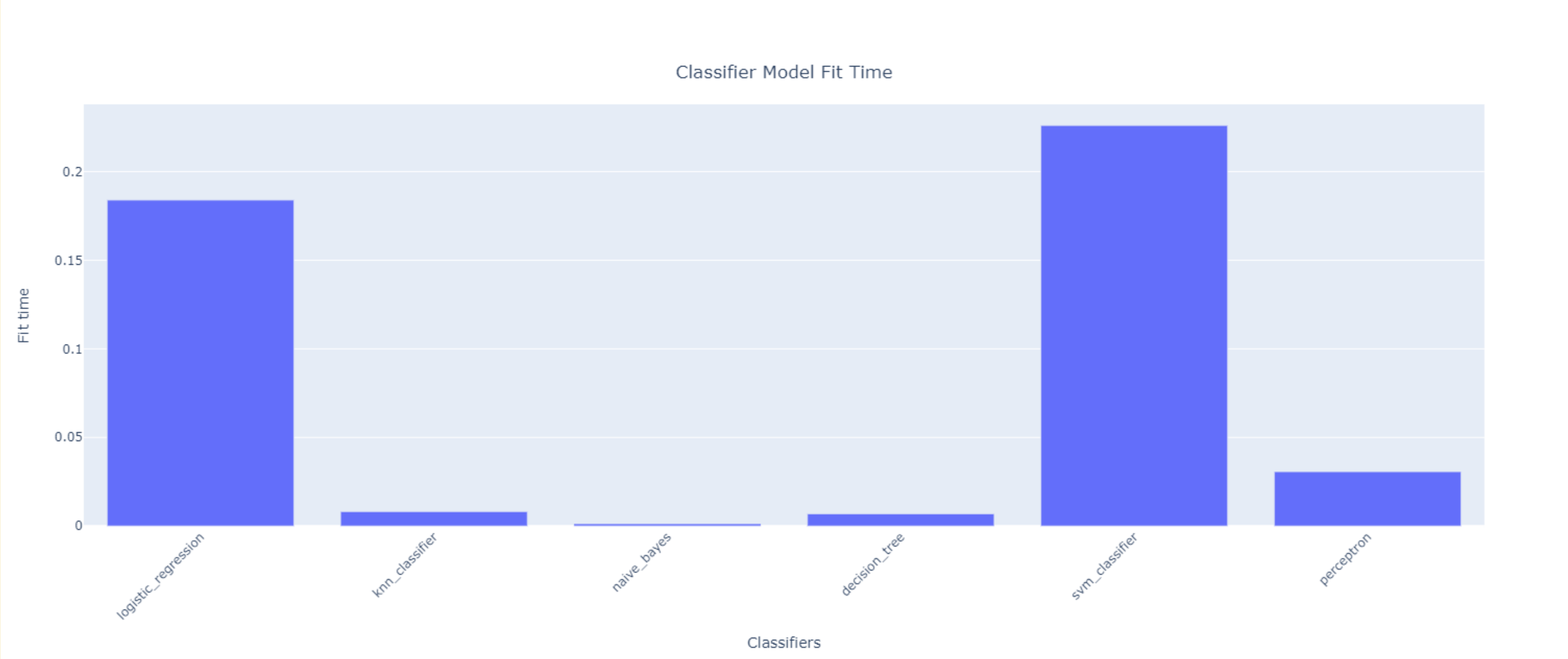
Optimal Classifiers ROC curves



Optimal Classifiers Evaluation Metrics



Optimal Classifiers Model fit time



Conclusion

- Overall best classifier (based on accuracy metrics): SVM
- Best hyper-parameters:
  - C: 1
  - gamma: 1
  - kernel: rbf
- Fastest model training: Gaussian Naïve Bayes
- Best tradeoff (accuracy~training time): kNN classifier