**Feature Selection**

***Objective***

*Understand and apply feature selection techniques (filter, embedded, or other methods) to optimize model performance and interpretability.*

***Instructions***

1. ***Data Selection:***
   * *Choose a dataset from sources like UCI Machine Learning Repository, Kaggle, or any relevant open dataset of your interest.*
   * *Ensure the dataset has at least 10 features and a target variable.*
2. ***Feature Selection Methods:***
   * *Explore and implement at least two feature selection methods:*
     + ***Filter methods:*** *Use statistical measures like correlation, mutual information, or chi-square test.*
     + ***Wrapper methods****: recursive, forward, or backward selection.*
     + ***Embedded methods:*** *Use algorithms like decision trees, Lasso regression, or XGBoost for feature importance.*
3. ***Implementation Steps:***
   * *Preprocess the data (handle missing values, normalize/standardize if necessary).*
   * *Apply feature selection techniques and compare the results.*
   * *Train a machine learning model (e.g., logistic regression, decision tree, or a classifier of your choice) on the selected features.*
   * *Compare the model's performance before and after feature selection.*
4. ***Submission Requirements:***
   * *A detailed report with:*
     + *Dataset description.*
     + *Explanation of methods used for feature selection.*
     + *Results and performance comparison (accuracy, precision, recall, etc.).*
   * *Copy and paste the source code (Python) to your PDF or link from Google Collabs.*

**Boston Housing Dataset**

1. **Introduction**

Feature selection is one of the most significant processes in Machine Learning which concentrates on determining the best set of features in a given data set. Reducing the feature space allows for improving the performance, interpretability and the cost of the models.

This report investigates different feature selection methods that were implemented on the Boston Housing dataset. The goal is to estimate the median value of an owner-occupied home *(medv)* using a number of features that describe housing and neighbourhood characteristics.

1. **Dataset Description**

The Boston Housing dataset contains 506 observations and 14 variables (13 features and 1 target). The target variable is medv, representing the median home price in $1000s.

**Features**

* crim: Per capita crime rate by town.
* zn: Proportion of residential land zoned for lots > 25,000 sq. ft.
* indus: Proportion of non-retail business acres per town.
* chas: Charles River dummy variable (1 if bounds river; 0 otherwise).
* nox: Nitric oxide concentration (parts per 10 million).
* rm: Average number of rooms per dwelling.
* age: Proportion of owner-occupied units built before 1940.
* dis: Weighted distances to five Boston employment centres.
* rad: Accessibility to radial highways.
* tax: Property-tax rate per $10,000.
* ptratio: Pupil-teacher ratio by town.
* black: 1000(Bk - 0.63)^2, where Bk is the proportion of Black residents by town.
* lstat: % lower status of the population.

**Target**

* medv: Median value of owner-occupied homes in $1000s.

1. **Methodology**

Five feature selection techniques were applied. Each method was evaluated by training a Linear Regression model on the selected features.

Performance metrics for the regression model included:

* RMSE (Root Mean Squared Error): Measures average prediction error.
* R² Score: Indicates how well the model explains variability in the target variable.

|  |
| --- |
| # Train a Linear Regression model  baseline\_model = LinearRegression()  baseline\_model.fit(X\_train, y\_train)  # Make predictions on the test set  y\_pred = baseline\_model.predict(X\_test)  # Evaluate the model  baseline\_rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)  baseline\_r2 = r2\_score(y\_test, y\_pred)  # Model Performance  print(f"RMSE: {baseline\_rmse:.2f}")  print(f"R^2 Score: {baseline\_r2:.2f}")  RMSE: 4.64  R^2 Score: 0.71 |

* 1. **Filter Methods**

A graph of different sizes and colors

Description automatically generated with medium confidence

Both correlation and mutual information identify rm and lstat as the most important features for predicting medv.

Correlation captures linear relationships, while mutual information highlights non-linear dependencies, revealing additional relevance for features like indus and nox.

* + 1. **Correlation:** Features are ranked by their correlation with the target variable.

|  |
| --- |
| # Select features  X\_train\_corr = pd.DataFrame(X\_train, columns=X.columns)[top\_corr\_features]  X\_test\_corr = pd.DataFrame(X\_test, columns=X.columns)[top\_corr\_features]  # Training  model\_corr = LinearRegression()  model\_corr.fit(X\_train\_corr, y\_train)  # Predictions  y\_pred\_corr = model\_corr.predict(X\_test\_corr)  # Model Performance  rmse\_corr = mean\_squared\_error(y\_test, y\_pred\_corr, squared=False)  r2\_corr = r2\_score(y\_test, y\_pred\_corr)  print("\nModel Performance with Top Correlation Features:")  print(f"\n - RMSE: {rmse\_corr:.2f}")  print(f"\n - R^2 Score: {r2\_corr:.2f}")  Model Performance with Top Correlation Features:  - RMSE: 5.11  - R^2 Score: 0.65 |

* + 1. **Mutual Information:** Measures non-linear dependencies between features and the target.

|  |
| --- |
| # Select features  X\_train\_mi = pd.DataFrame(X\_train, columns=X.columns)[top\_mi\_features]  X\_test\_mi = pd.DataFrame(X\_test, columns=X.columns)[top\_mi\_features]  # Training  model\_mi = LinearRegression()  model\_mi.fit(X\_train\_mi, y\_train)  # Predictions  y\_pred\_mi = model\_mi.predict(X\_test\_mi)  # Model Performance  rmse\_mi = mean\_squared\_error(y\_test, y\_pred\_mi, squared=False)  r2\_mi = r2\_score(y\_test, y\_pred\_mi)  print("Model Performance with Top Mutual Information Features:")  print(f"\n - RMSE: {rmse\_mi:.2f}")  print(f"\n - R^2 Score: {r2\_mi:.2f}")  Model Performance with Top Mutual Information Features:  - RMSE: 5.11  - R^2 Score: 0.65 |

* 1. **Wrapper Method**

**Recursive Feature Elimination (RFE):** Iteratively removes less important features using a model's coefficients.

|  |
| --- |
| # Linear Regression  estimator = LinearRegression()  selector = RFE(estimator, n\_features\_to\_select=5, step=1)  # Fit RFE on training data  selector = selector.fit(X\_train, y\_train)  # Select features  rfe\_selected\_features = X.columns[selector.support\_].tolist()  print("Features Selected by RFE:", rfe\_selected\_features)  # Training and evaluation  X\_train\_rfe = pd.DataFrame(X\_train, columns=X.columns)[rfe\_selected\_features]  X\_test\_rfe = pd.DataFrame(X\_test, columns=X.columns)[rfe\_selected\_features]  model\_rfe = LinearRegression()  model\_rfe.fit(X\_train\_rfe, y\_train)  y\_pred\_rfe = model\_rfe.predict(X\_test\_rfe)  # Model Performance  rmse\_rfe = mean\_squared\_error(y\_test, y\_pred\_rfe, squared=False)  r2\_rfe = r2\_score(y\_test, y\_pred\_rfe)  print("\nModel Performance with RFE Features:")  print(f"\n - RMSE: {rmse\_rfe:.2f}")  print(f"\n - R^2 Score: {r2\_rfe:.2f}")  Features Selected by RFE:  ['nox', 'rm', 'dis', 'ptratio', 'lstat']  Model Performance with RFE Features:  - RMSE: 4.77  - R^2 Score: 0.70 |

* 1. **Embedded Methods**
     1. **Lasso Regression (L1 Regularization):** Shrinks less important feature coefficients to zero.

|  |
| --- |
| # Lasso Regression  lasso = Lasso(alpha=0.1, random\_state=42)  lasso.fit(X\_train, y\_train)  # Feature Selection  lasso\_selected\_features = X.columns[lasso.coef\_ != 0].tolist()  print("\nFeatures Selected by Lasso Regression:\n", lasso\_selected\_features)  # Trainning and evaluation  X\_train\_lasso = pd.DataFrame(X\_train, columns=X.columns)[lasso\_selected\_features]  X\_test\_lasso = pd.DataFrame(X\_test, columns=X.columns)[lasso\_selected\_features]  model\_lasso = LinearRegression()  model\_lasso.fit(X\_train\_lasso, y\_train)  y\_pred\_lasso = model\_lasso.predict(X\_test\_lasso)  # Model Performance  rmse\_lasso = mean\_squared\_error(y\_test, y\_pred\_lasso, squared=False)  r2\_lasso = r2\_score(y\_test, y\_pred\_lasso)  print("\nModel Performance with Lasso-Selected Features:")  print(f"\n - RMSE: {rmse\_lasso:.2f}")  print(f"\n - R^2 Score: {r2\_lasso:.2f}")  Features Selected by Lasso Regression:  ['crim', 'zn', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax', 'ptratio', 'black', 'lstat']  Model Performance with Lasso-Selected Features:  - RMSE: 4.63  - R^2 Score: 0.71 |

* + 1. **Random Forest Feature Importance:** Uses a tree-based model to rank features by their contribution to predictions.

|  |
| --- |
| # Random Forest Regressor  rf = RandomForestRegressor(random\_state=42)  rf.fit(X\_train, y\_train)  # Feature importances  rf\_feature\_importances = pd.Series(rf.feature\_importances\_, index=X.columns).sort\_values(ascending=False)  # Select top 5 features  top\_rf\_features = rf\_feature\_importances.nlargest(5).index.tolist()  print("Top Features by Random Forest:", top\_rf\_features)  # Training and evaluation with selected features  X\_train\_rf = pd.DataFrame(X\_train, columns=X.columns)[top\_rf\_features]  X\_test\_rf = pd.DataFrame(X\_test, columns=X.columns)[top\_rf\_features]  model\_rf = LinearRegression()  model\_rf.fit(X\_train\_rf, y\_train)  y\_pred\_rf = model\_rf.predict(X\_test\_rf)  # Model Performance  rmse\_rf = mean\_squared\_error(y\_test, y\_pred\_rf, squared=False)  r2\_rf = r2\_score(y\_test, y\_pred\_rf)  print("Model Performance with RF-Selected Features:")  print(f"RMSE: {rmse\_rf:.2f}")  print(f"R^2 Score: {r2\_rf:.2f}")  Top Features by Random Forest:  ['rm', 'lstat', 'dis', 'crim', 'ptratio']  Model Performance with RF-Selected Features:  - RMSE: 5.05  - R^2 Score: 0.66 |

1. **Results and Evaluation**

**Performance of Feature Selection Methods**

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Selected Features** | **RMSE** | **R^2** |
| Baseline  (All Features) | All | 4.64 | 0.71 |
| Filter  (Correlation) | *lstat, rm, ptratio, indus, tax* | 5.11 | 0.65 |
| Filter  (Mutual Information) | *lstat, rm, indus, nox, ptratio* | 5.11 | 0.65 |
| Wrapper  (RFE) | *nox, rm, dis, ptratio, lstat* | 4.77 | 0.70 |
| Embedded  (Lasso Regression) | Most features except *indus* | 4.63 | 0.71 |
| Embedded  (Random Forest Importance) | *rm, lstat, dis, crim, ptratio* | 5.05 | 0.66 |

* With all features, the model achieved the best RMSE: 4.64 and R²: 0.71.
* With Filter Methods, Correlation and Mutual Information selected overlapping features but underperformed RMSE: 5.11 and R²: 0.65.
* With Wrapper Method the feature set was reduced to 5 features while maintaining strong performance RMSE: 4.77 and R²: 0.70.
* With Embedded Methods, Lasso Regression matched the baseline model’s performance while reducing the feature set, making it **the most effective** **technique** with RMSE: 4.63 and R²: 0.71., while Random Forest identified relevant features but slightly underperformed RMSE: 5.05 and R²: 0.66.

A close-up of a graph

Description automatically generated

1. **Conclusion**

* Lasso Regression was the most effective feature selection method, achieving baseline performance with fewer features.

1. **Appendix**

[**Google Collab Link**](https://colab.research.google.com/drive/1MJVx_xqhlg7OnnQsA5ers7j-UyvL3MsZ?usp=sharing)