

```
from IPython.display import Image, display
display(Image(filename='Lecture 9.jpg'))
```



Feature selection

Feature selection methods refer to techniques used in machine learning and data analysis to choose the most important input variables (features) for building a model. Instead of using all available features—which may include irrelevant, redundant, or noisy data—feature selection picks only those that significantly contribute to prediction performance.

Why Feature Selection Matters

Feature selection helps to:

Improve model accuracy by removing irrelevant noise

Reduce overfitting

Decrease training time and computational cost

Make models more interpretable

Handle high-dimensional datasets (common in genomics, images, text)

Main Categories of Feature Selection for classification problem

1. Filter Methods

Feature selection is done before training the model, using statistical tests. Model not involved.

Examples

- Correlation
- Chi-Square Test
- ANOVA (F-test)
- Mutual Information

2. Wrapper Methods

Feature selection is done by training a model repeatedly on different subsets of features. Model is involved and performance-based.

Examples

- Forward Selection
- Backward Elimination
- Recursive Feature Elimination (RFE)

3. Embedded Methods

Feature selection happens inside the model training process. The model learns which features matter automatically.

Examples

- L1 Regularization (Lasso)
- Decision Trees
- Random Forest Importance
- XGBoost / LightGBM Importance

Import necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score, classification_report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

```

The dataset linking

The dataset linking is <https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset>

Reading dataset

```

df = pd.read_csv("breast-cancer.xls")
df.head()

```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean
area_mean \					
0	842302	M	17.99	10.38	122.80
1001.0					
1	842517	M	20.57	17.77	132.90
1326.0					
2	84300903	M	19.69	21.25	130.00
1203.0					
3	84348301	M	11.42	20.38	77.58
386.1					
4	84358402	M	20.29	14.34	135.10
1297.0					
smoothness_mean					
compactness_mean					
concavity_mean					
concave					
points_mean \					
0	0.11840		0.27760	0.3001	
0.14710					
1	0.08474		0.07864	0.0869	
0.07017					
2	0.10960		0.15990	0.1974	
0.12790					
3	0.14250		0.28390	0.2414	
0.10520					
4	0.10030		0.13280	0.1980	
0.10430					
...	radius_worst	texture_worst	perimeter_worst	area_worst	\

0	...	25.38	17.33	184.60	2019.0
1	...	24.99	23.41	158.80	1956.0
2	...	23.57	25.53	152.50	1709.0
3	...	14.91	26.50	98.87	567.7
4	...	22.54	16.67	152.20	1575.0

	smoothness_worst	compactness_worst	concavity_worst	concave points_worst \
0	0.1622	0.6656	0.7119	0.2654
1	0.1238	0.1866	0.2416	0.1860
2	0.1444	0.4245	0.4504	0.2430
3	0.2098	0.8663	0.6869	0.2575
4	0.1374	0.2050	0.4000	0.1625

	symmetry_worst	fractal_dimension_worst
0	0.4601	0.11890
1	0.2750	0.08902
2	0.3613	0.08758
3	0.6638	0.17300
4	0.2364	0.07678

[5 rows x 32 columns]

Define features and target variable

```
# Check for missing values
```

```
df.isnull().sum()
```

id	0
diagnosis	0
radius_mean	0
texture_mean	0
perimeter_mean	0
area_mean	0
smoothness_mean	0
compactness_mean	0
concavity_mean	0
concave points_mean	0
symmetry_mean	0
fractal_dimension_mean	0
radius_se	0
texture_se	0
perimeter_se	0
area_se	0
smoothness_se	0

compactness_se	0
concavity_se	0
concave points_se	0
symmetry_se	0
fractal_dimension_se	0
radius_worst	0
texture_worst	0
perimeter_worst	0
area_worst	0
smoothness_worst	0
compactness_worst	0
concavity_worst	0
concave points_worst	0
symmetry_worst	0
fractal_dimension_worst	0

dtype: int64

df.dtypes

id	int64
diagnosis	object
radius_mean	float64
texture_mean	float64
perimeter_mean	float64
area_mean	float64
smoothness_mean	float64
compactness_mean	float64
concavity_mean	float64
concave points_mean	float64
symmetry_mean	float64
fractal_dimension_mean	float64
radius_se	float64
texture_se	float64
perimeter_se	float64
area_se	float64
smoothness_se	float64
compactness_se	float64
concavity_se	float64
concave points_se	float64
symmetry_se	float64
fractal_dimension_se	float64
radius_worst	float64
texture_worst	float64
perimeter_worst	float64
area_worst	float64
smoothness_worst	float64
compactness_worst	float64
concavity_worst	float64
concave points_worst	float64
symmetry_worst	float64

```
fractal_dimension_worst    float64
dtype: object
```

```
df.describe()
```

	id	radius_mean	texture_mean	perimeter_mean
area_mean \				
count	5.690000e+02	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033
std	1.250206e+08	3.524049	4.301036	24.298981
min	8.670000e+03	6.981000	9.710000	43.790000
25%	8.692180e+05	11.700000	16.170000	75.170000
50%	9.060240e+05	13.370000	18.840000	86.240000
75%	8.813129e+06	15.780000	21.800000	104.100000
max	9.113205e+08	28.110000	39.280000	188.500000

	smoothness_mean	compactness_mean	concavity_mean	concave
points_mean \				
count	569.000000	569.000000	569.000000	
mean	0.096360	0.104341	0.088799	
std	0.014064	0.052813	0.079720	
min	0.052630	0.019380	0.000000	
25%	0.086370	0.064920	0.029560	
50%	0.095870	0.092630	0.061540	
75%	0.105300	0.130400	0.130700	
max	0.163400	0.345400	0.426800	

	symmetry_mean	...	radius_worst	texture_worst
perimeter_worst \				
count	569.000000	...	569.000000	569.000000
mean	0.181162	...	16.269190	25.677223
std	0.027414	...	4.833242	6.146258

```

33.602542
min      0.106000 ...      7.930000      12.020000
50.410000
25%      0.161900 ...      13.010000      21.080000
84.110000
50%      0.179200 ...      14.970000      25.410000
97.660000
75%      0.195700 ...      18.790000      29.720000
125.400000
max      0.304000 ...      36.040000      49.540000
251.200000

```

```

      area_worst  smoothness_worst  compactness_worst
concavity_worst \
count  569.000000      569.000000      569.000000
569.000000
mean    880.583128      0.132369      0.254265
0.272188
std     569.356993      0.022832      0.157336
0.208624
min     185.200000      0.071170      0.027290
0.000000
25%     515.300000      0.116600      0.147200
0.114500
50%     686.500000      0.131300      0.211900
0.226700
75%    1084.000000      0.146000      0.339100
0.382900
max     4254.000000      0.222600      1.058000
1.252000

```

```

      concave points_worst  symmetry_worst  fractal_dimension_worst
count      569.000000      569.000000      569.000000
mean        0.114606      0.290076      0.083946
std         0.065732      0.061867      0.018061
min         0.000000      0.156500      0.055040
25%         0.064930      0.250400      0.071460
50%         0.099930      0.282200      0.080040
75%         0.161400      0.317900      0.092080
max         0.291000      0.663800      0.207500

```

[8 rows x 31 columns]

```

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis'])

X = df.drop(['diagnosis', 'id'], axis=1)
y = df['diagnosis']

```

Chi-Square

This measures dependence between categorical target & categorical features.

- Input features must be non-negative
- Fast and suitable for high-dimensional datasets

```
from sklearn.feature_selection import SelectKBest, chi2

# Create selector (choose number of features you want, e.g., k=5)
selector = SelectKBest(score_func=chi2, k=20)

# Fit and transform
selector.fit(X, y)

print("Selected Features:", selector.get_support())
print("Scores:", selector.scores_)

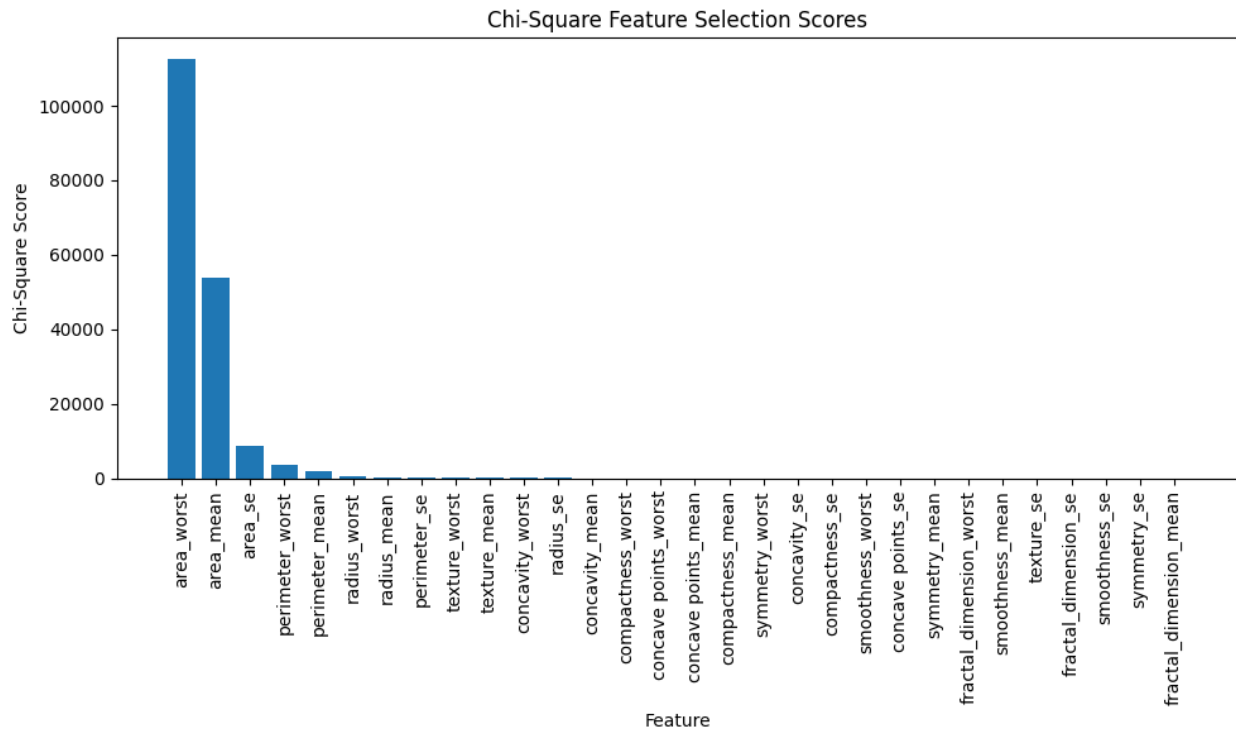
Selected Features: [False False  True  True False False False False
 False False False False
  False  True False False False False False False False  True
  True
  False False False False False False]
Scores: [2.66104917e+02 9.38975081e+01 2.01110286e+03 5.39916559e+04
 1.49899264e-01 5.40307549e+00 1.97123536e+01 1.05440354e+01
 2.57379775e-01 7.43065536e-05 3.46752472e+01 9.79353970e-03
 2.50571896e+02 8.75850471e+03 3.26620664e-03 6.13785332e-01
 1.04471761e+00 3.05231563e-01 8.03633831e-05 6.37136566e-03
 4.91689157e+02 1.74449400e+02 3.66503542e+03 1.12598432e+05
 3.97365694e-01 1.93149220e+01 3.95169151e+01 1.34854195e+01
 1.29886140e+00 2.31522407e-01]

import pandas as pd
import matplotlib.pyplot as plt

# Create DataFrame of features and chi-square scores
data = pd.DataFrame({'Feature': X.columns, 'Score': selector.scores_})

# Sort by score (optional, but makes plot clearer)
data = data.sort_values(by='Score', ascending=False)

# Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("Chi-Square Score")
plt.title("Chi-Square Feature Selection Scores")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Recursive Feature Elimination (RFE) with Logistic Regression

Recursive Feature Elimination (RFE) is a wrapper-based feature selection method that works by iteratively removing the least important features until it reaches the desired number of features.

It uses a machine-learning model (e.g., Logistic Regression, SVM, Random Forest) to rank feature importance.

Main Idea of RFE

- Train a model using ALL features
- Check feature importance (coefficients or `feature_importances_`)
- Remove the least important feature(s)
- Retrain the model on the remaining features
- Repeat until only the desired number of features remain

Ranking_

1 → selected feature

2, 3, 4... → removed earlier (higher = worse)

Example RFE with Logistic Regression

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
import warnings
```

```
model = LogisticRegression(max_iter=500)
```

```
rfe = RFE(model, n_features_to_select=15)
rfe.fit(X, y)
```

```
print("Selected Features:", rfe.get_support())
print("Ranking:", rfe.ranking_)
```

```
C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence
(max_iter=500).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence
(max_iter=500).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence
(max_iter=500).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence
(max_iter=500).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence
(max_iter=500).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 500 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT
```

Increase the number of iterations to improve the convergence
(max_iter=500).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
C:\Users\Compumarts\anaconda3\envs\coding\Lib\site-packages\sklearn\
linear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to
converge after 500 iteration(s) (status=1):
```

STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence
(max_iter=500).

You might also want to scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

```
n_iter_i = _check_optimize_result(
```

```
Selected Features: [ True False False False  True  True  True  True
 True False False  True
```

```
False False False False False False False False  True  True  True
False
```

```
True  True  True  True  True False]
```

```
Ranking: [ 1  3  5 15  1  1  1  1  1 14  9  1 16  4 12  6  8 10  7 13
1  1  1 11
```

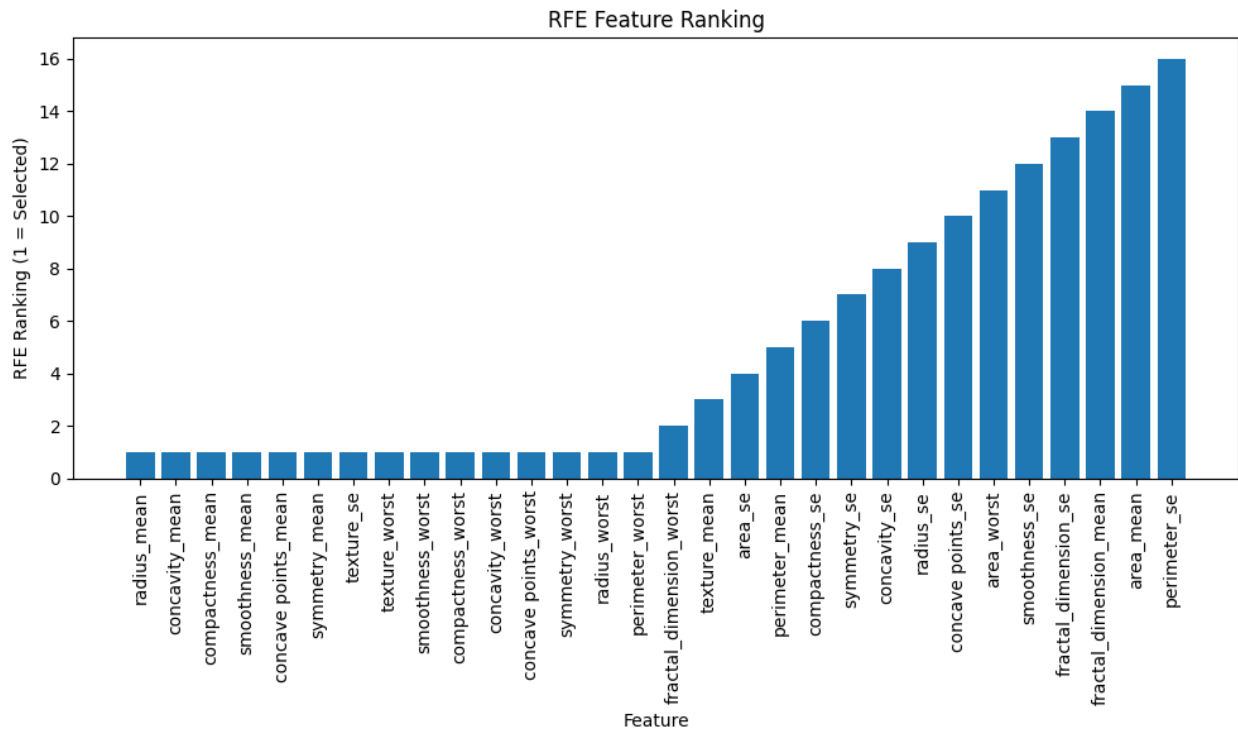
```
1  1  1  1  1  2]
```

```
import pandas as pd
```

```
feature_ranking = pd.DataFrame({
    'Feature': X.columns,
    'Ranking': rfe.ranking_
})
```

```
feature_ranking = feature_ranking.sort_values(by='Ranking',
ascending=True)
```

```
plt.figure(figsize=(10, 6))
plt.bar(feature_ranking['Feature'], feature_ranking['Ranking'])
plt.xlabel("Feature")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("RFE Feature Ranking")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Example RFE with Random Forest

```
rf = RandomForestClassifier(n_estimators=200, random_state=42)
```

```
# Apply RFE (select top 5 features)
```

```
rfe = RFE(estimator=rf, n_features_to_select=15)
```

```
rfe.fit(X, y)
```

```
print("Selected Features Mask:", rfe.get_support())
```

```
print("Ranking of Features:", rfe.ranking_)
```

```
Selected Features Mask: [ True  True  True  True False False  True
 True False False False False
 False  True False False False False False  True  True  True
 True
 True  True  True  True False False]
```

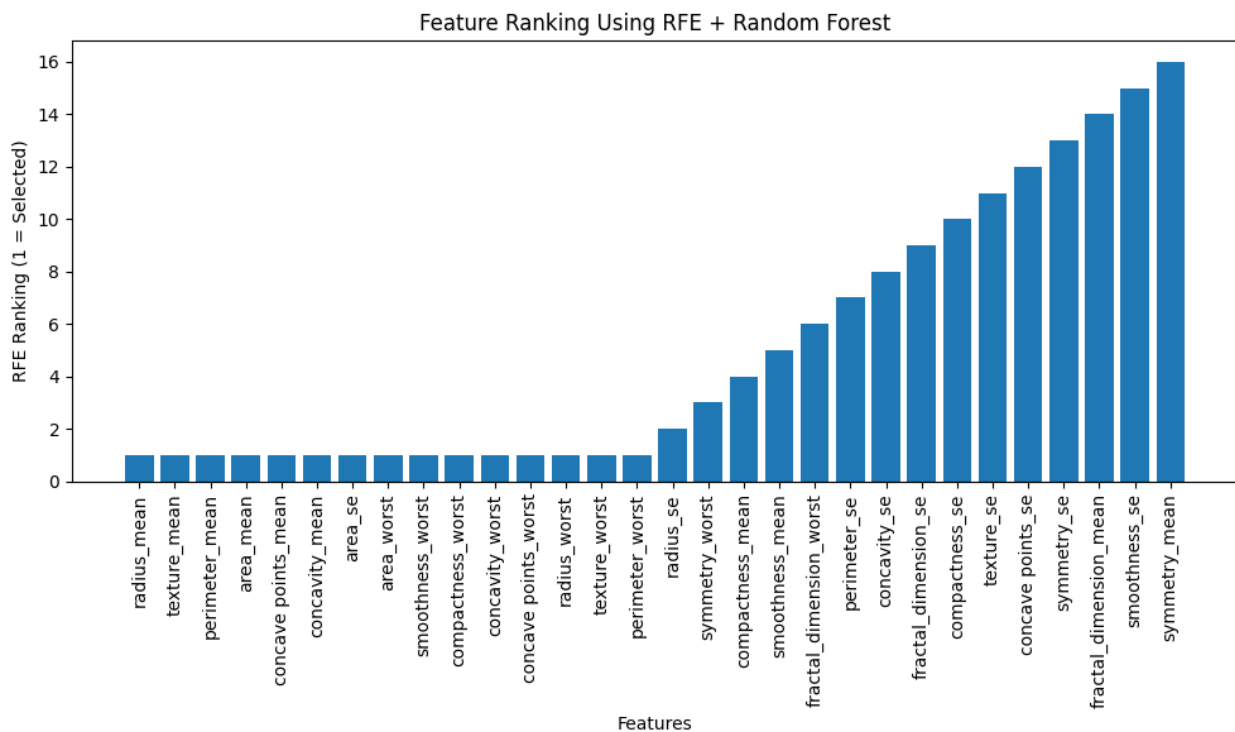
```
Ranking of Features: [ 1  1  1  1  5  4  1  1 16 14  2 11  7  1 15 10
 8 12 13  9  1  1  1  1
 1  1  1  1  3  6]
```

```
import pandas as pd
```

```
ranking_df = pd.DataFrame({
    'Feature': X.columns,
    'RFE_Ranking': rfe.ranking_
}).sort_values('RFE_Ranking')
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.bar(ranking_df['Feature'], ranking_df['RFE_Ranking'])
plt.xlabel("Features")
plt.ylabel("RFE Ranking (1 = Selected)")
plt.title("Feature Ranking Using RFE + Random Forest")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



Embedded Methods (Feature Selection Happens During Training)

These methods use models that have built-in feature importance.

Example: Feature Selection Using Random Forest Importance

```
df = pd.read_csv("breast-cancer.xls")

le = LabelEncoder()
df['diagnosis'] = le.fit_transform(df['diagnosis'])
X = df.drop(['diagnosis', 'id'], axis=1)
y = df['diagnosis']
```

```

# Train Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X, y)

# Extract importances
importances = rf.feature_importances_

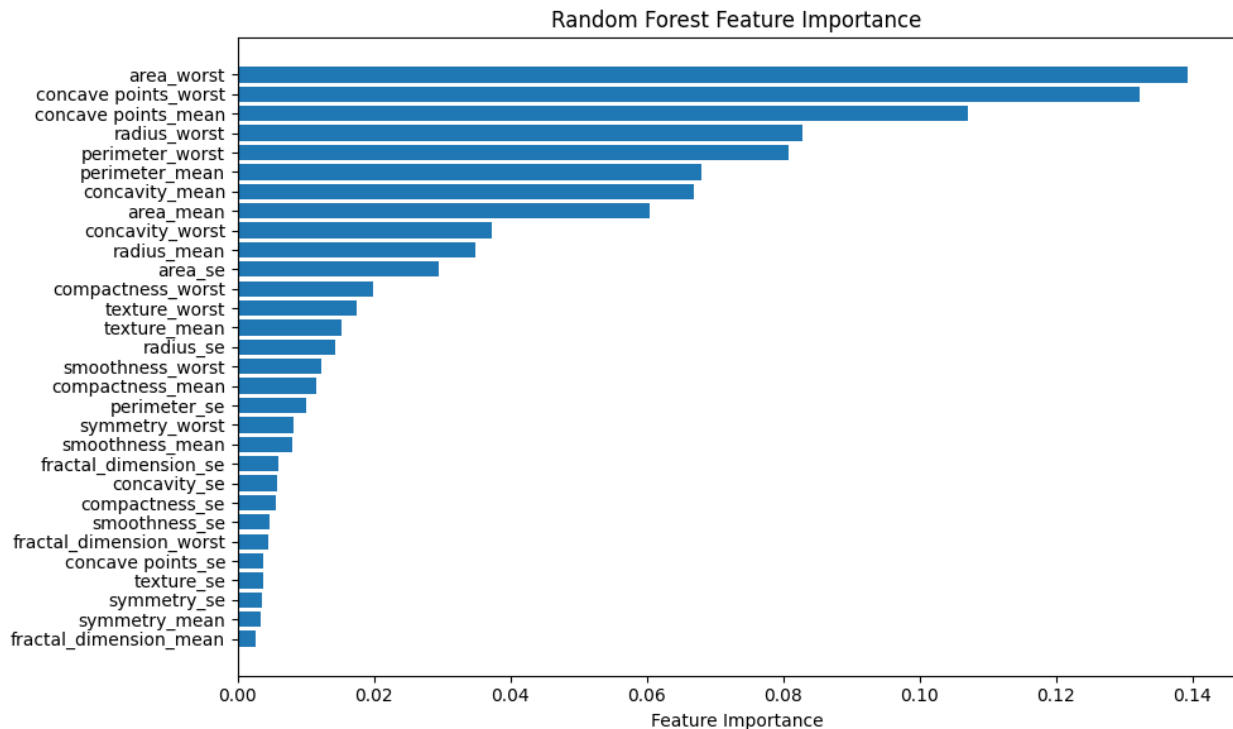
# Use feature names from X (correct and safe)
feature_names = X.columns if hasattr(X, "columns") else [f"Feature
{i}" for i in range(X.shape[1])]

# Create DataFrame
df_importance = pd.DataFrame({
    'Feature': feature_names,
    'Importance': importances
})

# Sort by importance
df_importance = df_importance.sort_values(by='Importance',
ascending=False)

# Plot
plt.figure(figsize=(10, 6))
plt.barh(df_importance['Feature'][::-1], df_importance['Importance']
[:: -1])
plt.xlabel('Feature Importance')
plt.title('Random Forest Feature Importance')
plt.tight_layout()
plt.show()

```



Comparing models using features selected by the chi-square

```
import pandas as pd
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, precision_score,
recall_score

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# 2. CHI-SQUARE FEATURE SELECTION
# -----
k = 15
selector = SelectKBest(chi2, k=k)
X_train_k = selector.fit_transform(X_train, y_train)
X_test_k = selector.transform(X_test)

selected_features = X.columns[selector.get_support()]
print("Selected Features:", list(selected_features))

# -----
# 3. MODELS
# -----
```

```

models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}

metrics = [] # will store results

# -----
# 4. TRAIN + EVALUATE
# -----
for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)

    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred,
average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted')
    })

# -----
# 5. RESULTS AS DATAFRAME
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance:")
print(results_df)

Selected Features: ['radius_mean', 'texture_mean', 'perimeter_mean',
'area_mean', 'concavity_mean', 'radius_se', 'perimeter_se', 'area_se',
'radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst',
'compactness_worst', 'concavity_worst', 'concave points_worst']

Model Performance:

```

	Model	Accuracy	Precision	Recall
0	Random Forest	0.964912	0.965205	0.964912
1	Decision Tree	0.921053	0.921213	0.921053
2	SVM	0.947368	0.951470	0.947368

Comparing models using features selected by REF

```

# -----
# 1. TRAIN/TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

```

```

# -----
# 2. RFE FEATURE SELECTION
# -----
k = 15 # number of features to select

rf_selector = RandomForestClassifier(n_estimators=200,
random_state=42)
rfe = RFE(estimator=rf_selector, n_features_to_select=k)

# Fit RFE on training data
rfe.fit(X_train, y_train)

# Transform datasets
X_train_k = rfe.transform(X_train)
X_test_k = rfe.transform(X_test)

# Get selected features
selected_features = X.columns[rfe.get_support()]
print("Selected Features by RFE:", list(selected_features))

# -----
# 3. MODELS TO COMPARE
# -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}

metrics = [] # to store accuracy, precision, recall

# -----
# 4. TRAIN + EVALUATE
# -----
for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)

    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred,
average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted')
    })

# -----
# 5. RESULTS DATAFRAME

```

```
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RFE-Selected Features:")
results_df

Selected Features by RFE: ['radius_mean', 'texture_mean',
'perimeter_mean', 'area_mean', 'concavity_mean', 'concave
points_mean', 'area_se', 'radius_worst', 'texture_worst',
'perimeter_worst', 'area_worst', 'smoothness_worst',
'concavity_worst', 'concave points_worst', 'symmetry_worst']
```

Model Performance Using RFE-Selected Features:

	Model	Accuracy	Precision	Recall
0	Random Forest	0.964912	0.965205	0.964912
1	Decision Tree	0.929825	0.929825	0.929825
2	SVM	0.947368	0.951470	0.947368

Comparing models using features selected by the Random Forest Feature Importance

```
# -----
# 1. TRAIN/TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# 2. SELECT FEATURES USING RANDOM FOREST IMPORTANCE
# -----
k = 18 # number of top important features to keep

rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)

# Get importance scores
importances = rf.feature_importances_

# Sort indices by importance (descending)
sorted_idx = np.argsort(importances)[::-1]

# Select top-k feature indices
top_k_idx = sorted_idx[:k]

# Get feature names
selected_features = X.columns[top_k_idx]
print("Selected Features by RF Importance:", list(selected_features))
```

```

# Reduce datasets to selected features
X_train_k = X_train[selected_features]
X_test_k = X_test[selected_features]

# -----
# 3. MODELS TO COMPARE
# -----
models = {
    "Random Forest": RandomForestClassifier(n_estimators=200,
random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "SVM": SVC(kernel='rbf')
}

metrics = []

# -----
# 4. TRAIN + EVALUATE
# -----
for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)

    metrics.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred,
average='weighted'),
        "Recall": recall_score(y_test, y_pred, average='weighted')
    })

# -----
# 5. RESULTS DATAFRAME
# -----
results_df = pd.DataFrame(metrics)
print("\nModel Performance Using RF-Selected Features:")
results_df

Selected Features by RF Importance: ['area_worst', 'concave
points_worst', 'perimeter_worst', 'concave points_mean',
'radius_worst', 'concavity_mean', 'radius_mean', 'area_mean',
'concavity_worst', 'perimeter_mean', 'area_se', 'texture_worst',
'compactness_worst', 'radius_se', 'compactness_mean', 'texture_mean',
'smoothness_worst', 'concavity_se']

Model Performance Using RF-Selected Features:

```

	Model	Accuracy	Precision	Recall
0	Random Forest	0.956140	0.956088	0.956140

1	Decision Tree	0.912281	0.912281	0.912281
2	SVM	0.947368	0.951470	0.947368

Feature selection methods for regression models

1. Filter Methods (Statistical Tests Before Modeling)

Filter methods evaluate each feature independently of the regression model. They use statistical metrics to decide which features to keep.

Examples

- Correlation coefficient (Pearson, Spearman)
- Mutual information regression
- Variance threshold
- ANOVA F-test (f_regression)

```
data = pd.read_csv("WineQT.csv")
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar
0	7.4	0.70	0.00	1.9
1	7.8	0.88	0.00	2.6
2	7.8	0.76	0.04	2.3
3	11.2	0.28	0.56	1.9
4	7.4	0.70	0.00	1.9

	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	11.0	34.0	0.9978	3.51	0.56
1	25.0	67.0	0.9968	3.20	0.68
2	15.0	54.0	0.9970	3.26	0.65
3	17.0	60.0	0.9980	3.16	0.58
4	11.0	34.0	0.9978	3.51	0.56

alcohol	quality	Id
---------	---------	----

0	9.4	5	0
1	9.8	5	1
2	9.8	5	2
3	9.8	6	3
4	9.4	5	4

```
X = data.drop(['quality', 'Id'], axis=1)
y = data['quality']
```

Import necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest,
mutual_info_regression
from xgboost import XGBRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Select top k features (choose any k)
selector = SelectKBest(score_func=mutual_info_regression, k='all')

# Fit selector
selector.fit(X, y)

print("Selected Features:", selector.get_support())
print("Scores:", selector.scores_)

# Create DataFrame of features and F-test scores
data = pd.DataFrame({
    'Feature': X.columns,
    'Score': selector.scores_
})

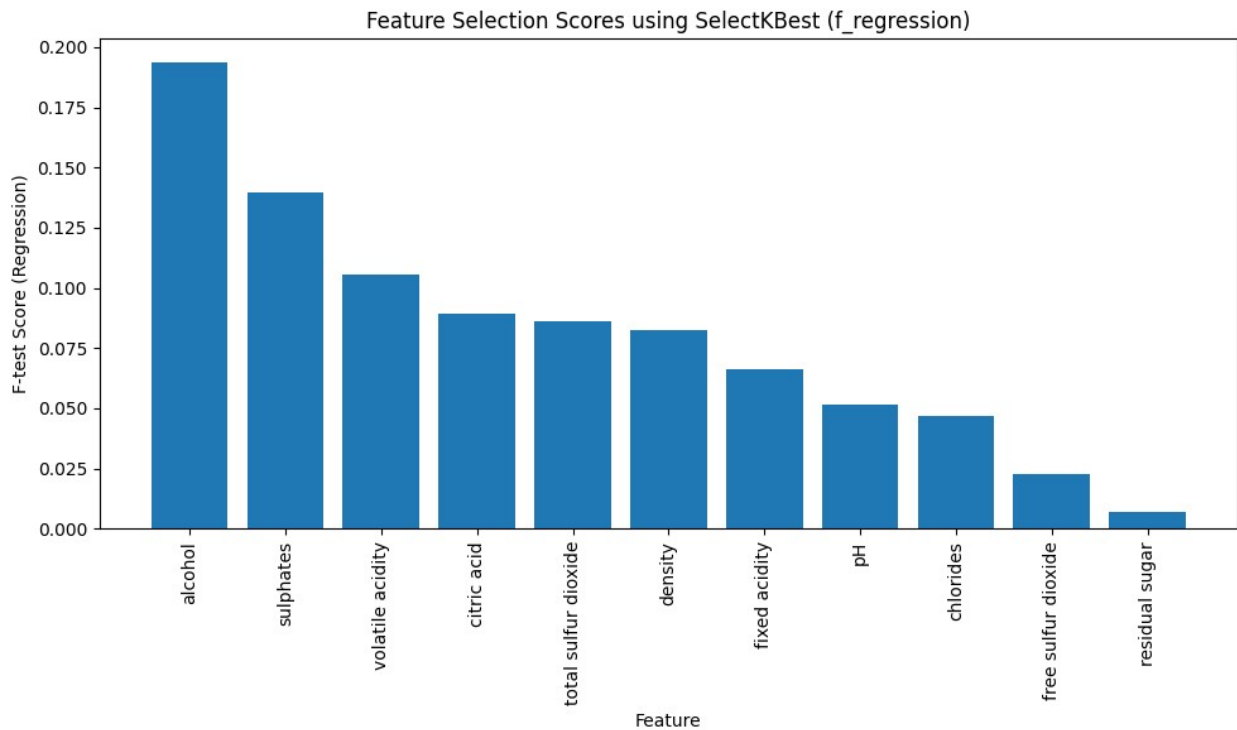
# Sort by score for clearer plot
data = data.sort_values(by='Score', ascending=False)

# Plot bar chart
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Score'])
plt.xlabel("Feature")
plt.ylabel("F-test Score (Regression)")
plt.title("Feature Selection Scores using SelectKBest (f_regression)")
plt.xticks(rotation=90)
```

```
plt.tight_layout()
plt.show()
```

```
Selected Features: [ True  True  True  True  True  True  True  True
 True  True  True]
```

```
Scores: [0.06648509 0.10580416 0.08929389 0.00688396 0.04664157
0.02252055
0.08621084 0.08259931 0.0515941 0.13962082 0.19376264]
```



2. Wrapper Methods (Search for Best Subset)

Wrapper methods evaluate subsets of features using a regression model. They are computationally expensive but often provide better results.

Examples

- Forward Feature Selection
- Backward Elimination
- Recursive Feature Elimination (RFE)

```
from sklearn.feature_selection import RFE

# Define regression model for RFE
model = RandomForestRegressor()
```

```

# RFE: choose number of features (or use all for ranking)
rfe = RFE(estimator=model, n_features_to_select=5)  # choose k
features
rfe.fit(X, y)

# Get ranking (1 = selected, higher = less important)
ranking = rfe.ranking_
support = rfe.support_

print("Selected Features:", X.columns[support])
print("Feature Ranking:", ranking)

# Create DataFrame of features and RFE ranks
data = pd.DataFrame({
    'Feature': X.columns,
    'Rank': ranking
})

# Sort by rank (1 is best)
data = data.sort_values(by='Rank', ascending=True)

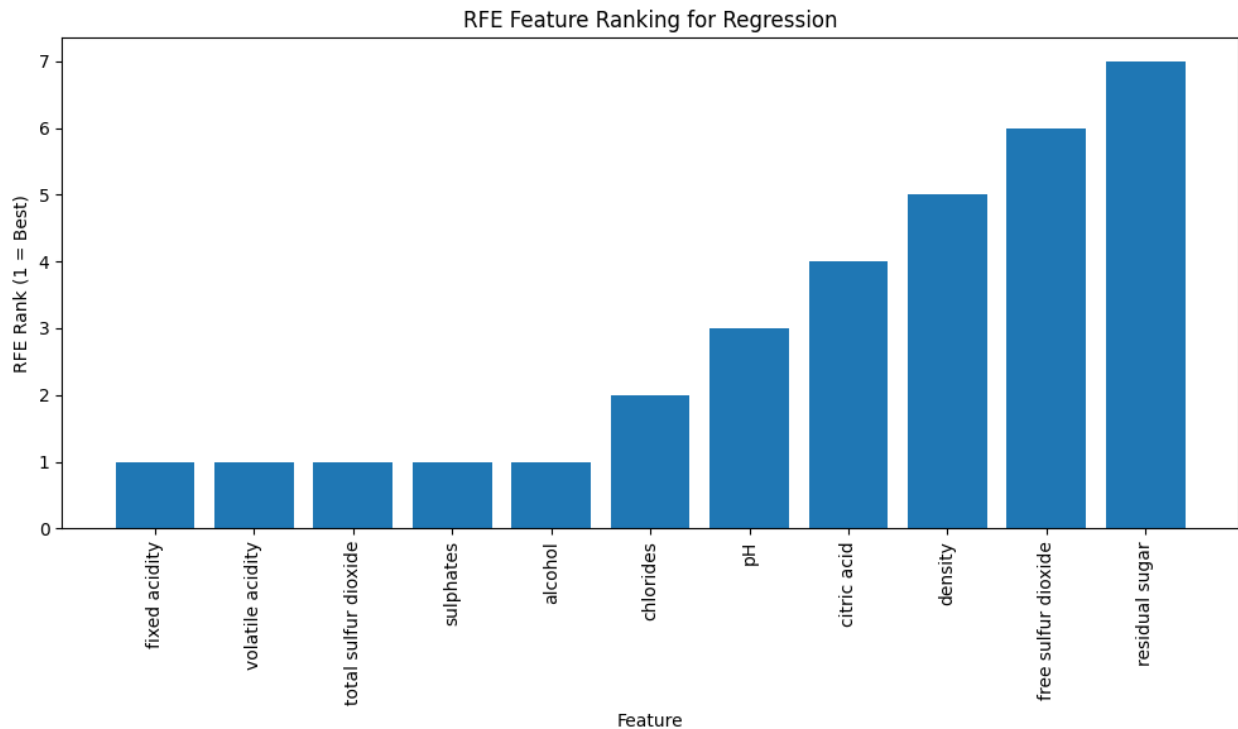
# Plot bar chart of feature ranks
plt.figure(figsize=(10, 6))
plt.bar(data['Feature'], data['Rank'])
plt.xlabel("Feature")
plt.ylabel("RFE Rank (1 = Best)")
plt.title("RFE Feature Ranking for Regression")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()

```

```

Selected Features: Index(['fixed acidity', 'volatile acidity', 'total
sulphur dioxide',
                        'sulphates', 'alcohol'],
                        dtype='object')
Feature Ranking: [1 1 4 7 2 6 1 5 3 1 1]

```



3. Embedded Methods (Feature Selection During Model Training)

Embedded methods perform feature selection within the model itself. They are efficient and commonly used in regression.

Examples

- Lasso Regression (L1 penalty) → shrinks some coefficients to zero
- Elastic Net Regression (L1 + L2)
- Random Forest Regressor (feature importance) or XGBoost regressor

```
from xgboost import XGBRegressor

# Define XGBoost Regressor (embedded method)
model = XGBRegressor(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)

# Fit XGBoost
model.fit(X, y)
```

```

# Get feature importance scores
importances = model.feature_importances_

# Create DataFrame
data = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
})

# Sort by importance
data = data.sort_values(by='Importance', ascending=False)

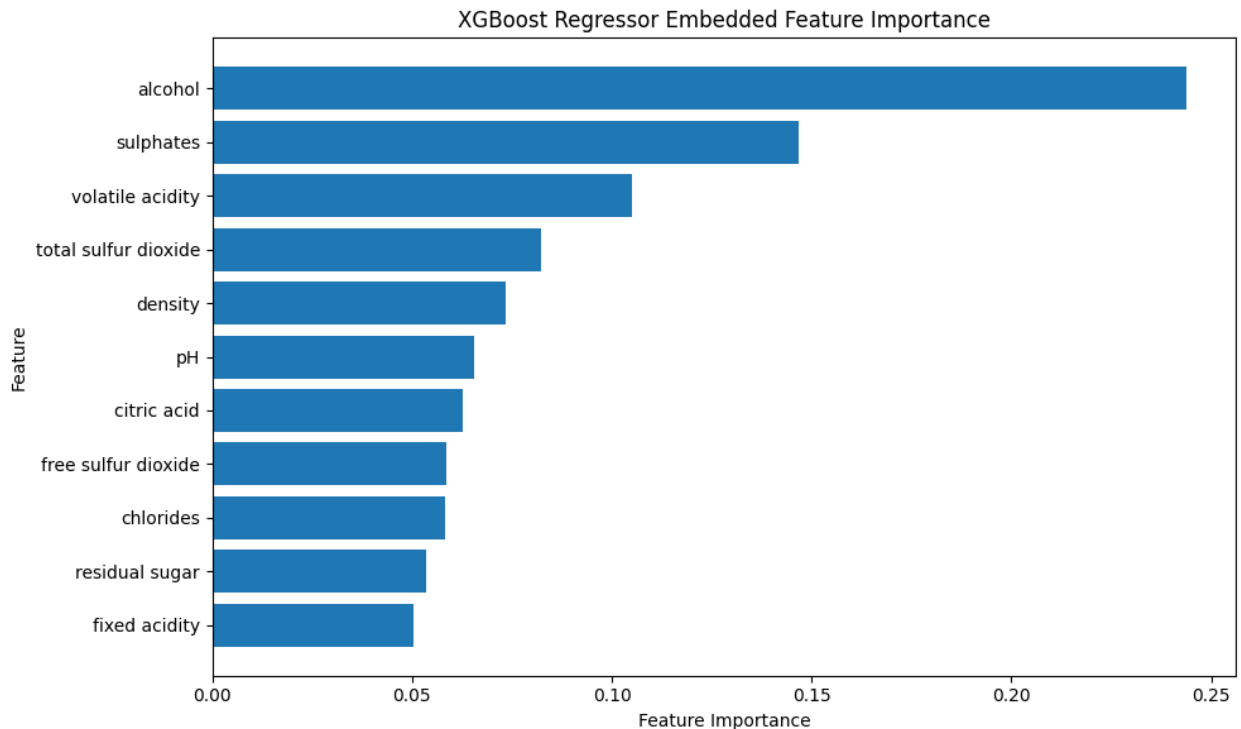
print("Top Features Selected by XGBoost Embedded Method:")
print(data)

# Plot feature importance
plt.figure(figsize=(10, 6))
plt.barh(data['Feature'][::-1], data['Importance'][::-1])
plt.xlabel("Feature Importance")
plt.ylabel("Feature")
plt.title("XGBoost Regressor Embedded Feature Importance")
plt.tight_layout()
plt.show()

```

Top Features Selected by XGBoost Embedded Method:

	Feature	Importance
10	alcohol	0.243906
9	sulphates	0.146671
1	volatile acidity	0.104981
6	total sulfur dioxide	0.082145
7	density	0.073281
8	pH	0.065634
2	citric acid	0.062707
5	free sulfur dioxide	0.058579
4	chlorides	0.058345
3	residual sugar	0.053470
0	fixed acidity	0.050279



Compare DTR, RFR, SVR Using SelectKBest + mutual_info_regression

```
# -----
# 1. TRAIN-TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# 2. FEATURE SELECTION USING SelectKBest + mutual_info_regression
# -----

k = 5 # choose top k features

selector = SelectKBest(score_func=mutual_info_regression, k=k)
X_train_k = selector.fit_transform(X_train, y_train)
X_test_k = selector.transform(X_test)

selected_features = X.columns[selector.get_support()]
print("\nSelected Features by Mutual Information Regression:")
print(list(selected_features))

# -----
# 3. REGRESSION MODELS TO COMPARE
# -----
models = {
```

```

    "Decision Tree Regressor": DecisionTreeRegressor(random_state=42),
    "Random Forest Regressor": RandomForestRegressor(n_estimators=200,
random_state=42),
    "SVR (RBF Kernel)": SVR(kernel='rbf')
}

results = []

# -----
# 4. TRAIN + EVALUATE MODELS
# -----
for name, model in models.items():
    model.fit(X_train_k, y_train)
    y_pred = model.predict(X_test_k)

    results.append({
        "Model": name,
        "MAE": mean_absolute_error(y_test, y_pred),
        "MSE": mean_squared_error(y_test, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_test, y_pred)),
        "R2 Score": r2_score(y_test, y_pred)
    })

# -----
# 5. RESULTS DATAFRAME
# -----
results_df = pd.DataFrame(results)
print("\nModel Performance Comparison Using MI-selected Features:")
print(results_df)

```

Selected Features by Mutual Information Regression:
['volatile acidity', 'citric acid', 'total sulfur dioxide',
'sulphates', 'alcohol']

Model Performance Comparison Using MI-selected Features:

	Model	MAE	MSE	RMSE	R ² Score
0	Decision Tree Regressor	0.454148	0.541485	0.735856	0.026934
1	Random Forest Regressor	0.436594	0.332861	0.576941	0.401838
2	SVR (RBF Kernel)	0.496435	0.438637	0.662297	0.211754

Compare DTR, RFR, SVR Using RFE-Selected Features regrssion

```

# -----
# 1. TRAIN-TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

```

```

# -----
# 2. FEATURE SELECTION USING RFE
# -----
selector_model = RandomForestRegressor()
rfe = RFE(estimator=selector_model, n_features_to_select=5)
rfe.fit(X_train, y_train)

# Select only RFE-chosen features
selected_features = X.columns[rfe.get_support()]
print("Selected Features by RFE:", list(selected_features))

X_train_rfe = rfe.transform(X_train)
X_test_rfe = rfe.transform(X_test)

# -----
# 3. REGRESSION MODELS
# -----
models = {
    "Decision Tree Regressor": DecisionTreeRegressor(random_state=42),
    "Random Forest Regressor": RandomForestRegressor(n_estimators=200,
random_state=42),
    "SVR (RBF Kernel)": SVR(kernel='rbf')
}

results = []

# -----
# 4. TRAIN + EVALUATE MODELS
# -----
for name, model in models.items():
    model.fit(X_train_rfe, y_train)
    y_pred = model.predict(X_test_rfe)

    results.append({
        "Model": name,
        "MAE": mean_absolute_error(y_test, y_pred),
        "MSE": mean_squared_error(y_test, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_test, y_pred)),
        "R2 Score": r2_score(y_test, y_pred)
    })

# -----
# 5. RESULTS DATAFRAME
# -----
results_df = pd.DataFrame(results)
print("\nModel Performance Comparison:")
print(results_df)

```

Selected Features by RFE: ['volatile acidity', 'total sulfur dioxide', 'pH', 'sulphates', 'alcohol']

Model Performance Comparison:

	Model	MAE	MSE	RMSE	R ² Score
0	Decision Tree Regressor	0.519651	0.676856	0.822713	-0.216332
1	Random Forest Regressor	0.425917	0.316719	0.562778	0.430846
2	SVR (RBF Kernel)	0.496729	0.439815	0.663185	0.209638

Compare DTR, RFR, SVR Using embedded Features regrssion

```
# -----
# 1. TRAIN-TEST SPLIT
# -----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# -----
# 2. EMBEDDED FEATURE SELECTION USING XGBOOST
# -----
xgb_selector = XGBRegressor(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=5,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42
)

xgb_selector.fit(X_train, y_train)

# Extract feature importances
importances = xgb_selector.feature_importances_

# Keep top-k most important features
k = 5
indices = np.argsort(importances)[::-1][:k]
selected_features = X.columns[indices]

print("\nTop 5 Selected Features by XGBoost Embedded Method:")
print(list(selected_features))

# Reduce dataset to selected features
X_train_emb = X_train[selected_features]
X_test_emb = X_test[selected_features]

# -----
# 3. MODELS TO COMPARE
# -----
```

```

models = {
    "Decision Tree Regressor": DecisionTreeRegressor(random_state=42),
    "Random Forest Regressor": RandomForestRegressor(n_estimators=200,
random_state=42),
    "SVR (RBF Kernel)": SVR(kernel='rbf')
}

results = []

# -----
# 4. TRAIN + EVALUATE MODELS
# -----
for name, model in models.items():
    model.fit(X_train_emb, y_train)
    y_pred = model.predict(X_test_emb)

    results.append({
        "Model": name,
        "MAE": mean_absolute_error(y_test, y_pred),
        "MSE": mean_squared_error(y_test, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_test, y_pred)),
        "R2 Score": r2_score(y_test, y_pred)
    })

# -----
# 5. RESULTS DATAFRAME
# -----
results_df = pd.DataFrame(results)
print("\nModel Performance Comparison using XGBoost Selected
Features:")
print(results_df)

```

Top 5 Selected Features by XGBoost Embedded Method:
['alcohol', 'sulphates', 'volatile acidity', 'total sulfur dioxide',
'density']

Model Performance Comparison using XGBoost Selected Features:

	Model	MAE	MSE	RMSE	R ² Score
0	Decision Tree Regressor	0.502183	0.676856	0.822713	-0.216332
1	Random Forest Regressor	0.435983	0.330795	0.575148	0.405550
2	SVR (RBF Kernel)	0.497041	0.439889	0.663241	0.209506