---

1. Filter Methods

Explanation: Filter methods assess the importance of features based on statistical tests and metrics, independent of any machine learning model.

a) Correlation-based Feature Selection

- **When to Use**: When you want to remove redundant features that are highly correlated with each other. It is mainly used for numerical features.
- **Why to Use**: Helps in reducing multicollinearity and improving model performance.

```
import pandas as pd

# Example data
data = pd.read_csv('data.csv')
correlation_matrix = data.corr()

# Select features with correlation above a threshold (0.8)
highly_correlated_features = correlation_matrix[correlation_matrix.abs() > 0.8]
print(highly_correlated_features)
```

---

b) Chi-square Test (for Categorical Data)

- **When to Use**: For categorical features. Chi-square test is used to find the dependency between the feature and the target variable.
- **Why to Use**: Works well when the data is categorical and helps in selecting features that have strong dependencies with the target variable.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import LabelEncoder

# Example data
X = data.drop(columns=['target'])
y = LabelEncoder().fit_transform(data['target'])

# Select top 5 features based on Chi-Square test
chi2_selector = SelectKBest(chi2, k=5)
X_new = chi2_selector.fit_transform(X, y)
print(X_new)
```

---

2. Wrapper Methods

Explanation: Wrapper methods evaluate subsets of features based on the performance of a given machine learning model.

a) Recursive Feature Elimination (RFE)

- **When to Use**: When you need to select a specific number of features and want to recursively remove the least important features based on model performance.
- **Why to Use**: RFE can help identify the most relevant features in a dataset using model performance as a metric.

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

# Example data
X = data.drop(columns=['target'])
y = data['target']

# Logistic Regression model
model = LogisticRegression()

# RFE to select the top 5 features
selector = RFE(model, n_features_to_select=5)
X_new = selector.fit_transform(X, y)
print(X_new)
```

---

3. Embedded Methods

Explanation: Embedded methods perform feature selection during the model training process, making them computationally efficient.

a) Lasso (L1 Regularization)

- **When to Use**: When you have a large number of features and need automatic feature selection while building a regression model.
- **Why to Use**: Lasso shrinks some feature coefficients to zero, effectively performing feature selection while building the model.

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler

# Example data
X = data.drop(columns=['target'])
y = data['target']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Lasso model for feature selection
lasso = Lasso(alpha=0.01)
lasso.fit(X_scaled, y)

# Get the selected features
selected_features = X.columns[(lasso.coef_ != 0)]
print(selected_features)
```

b) Decision Tree-based Feature Importance

- **When to Use**: When using decision trees or tree-based models like Random Forests and need to identify important features.
- **Why to Use**: Decision Trees assign a feature importance score based on how much they reduce impurity in the decision-making process.

```
from sklearn.ensemble import RandomForestClassifier

# Example data
X = data.drop(columns=['target'])
y = data['target']

# Random Forest model
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X, y)

# Get feature importance
importance = rf.feature_importances_
selected_features = X.columns[importance > 0.1]  # Select features with importance > 0.1
print(selected_features)
```

---

4. Mutual Information

Explanation: Mutual information measures the dependency between the feature and the target variable. The higher the mutual information, the stronger the relationship.

a) Mutual Information for Classification

- **When to Use**: For both categorical and continuous data, especially if you want to measure non-linear relationships between features and the target.
- **Why to Use**: It captures both linear and non-linear dependencies between features and target variables.

```
from sklearn.feature_selection import mutual_info_classif

# Example data
X = data.drop(columns=['target'])
y = data['target']

# Calculate mutual information
mi = mutual_info_classif(X, y)

# Select features with high mutual information
selected_features = X.columns[mi > 0.1]  # Example threshold
print(selected_features)
```

---

5. Sequential Feature Selection

Explanation: This method adds or removes features one at a time, based on model performance, to build a feature subset that improves performance.

a) Sequential Feature Selector

- **When to Use**: When you want to select the best subset of features based on cross-validation performance.
- **Why to Use**: Sequential feature selection allows you to select the most relevant features step by step, ensuring optimal model performance.

```
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.linear_model import LogisticRegression

# Example data
X = data.drop(columns=['target'])
y = data['target']

# Sequential feature selection with Logistic Regression
sfs = SequentialFeatureSelector(LogisticRegression(), n_features_to_select=5)
sfs.fit(X, y)

# Get selected features
selected_features = X.columns[sfs.get_support()]
print(selected_features)
```

---

6. Tree-based Methods (e.g., XGBoost, LightGBM)

Explanation: Advanced tree-based methods like **XGBoost** or **LightGBM** naturally provide feature importance as part of the model.

a) Feature Importance using XGBoost

- **When to Use**: When you are using XGBoost for classification or regression tasks, as it inherently provides feature importance.
- **Why to Use**: Tree-based methods, especially XGBoost, provide robust feature importance ranking, helping you identify the most important features.

```
import xgboost as xgb

# Example data
X = data.drop(columns=['target'])
y = data['target']

# XGBoost model
model = xgb.XGBClassifier()
model.fit(X, y)

# Get feature importance
importance = model.feature_importances_
selected_features = X.columns[importance > 0.1]  # Select features with importance > 0.1
print(selected_features)
```

Summary: When and Why to Use Each Method

- **Filter Methods**: Use when you want quick, model-independent feature selection, especially for high-dimensional data.

Feature Selection

- **Wrapper Methods**: Use when accuracy is a priority, and computational cost is manageable. Good for smaller datasets.
- **Embedded Methods**: Use when you want feature selection as part of model training, with minimal additional computation (e.g., Lasso, Decision Trees).
- **Mutual Information**: Use when you want to capture non-linear dependencies between features and the target variable.
- **Sequential Feature Selection**: Use when you need a method to incrementally add or remove features to improve model performance.

Each method has its strengths and should be chosen based on your data type, model, and performance requirements.