1.Supervised Techniques

1.1 Filter-based Approach

- Information gain
- Chi-square Test
- Fisher's Score
- Missing Value Ratio

1.2 Wrapper-based Approach

- Forward Selection
- Backward Selection
- Exhaustive Feature Selection
- Recursive Feature Elimination

1.3 Embedded Approach

- Regularization
- Random Forest Importance

2.Unsupervised Techniques

2.1 PCA

2.2 ICA

2.3 NMF

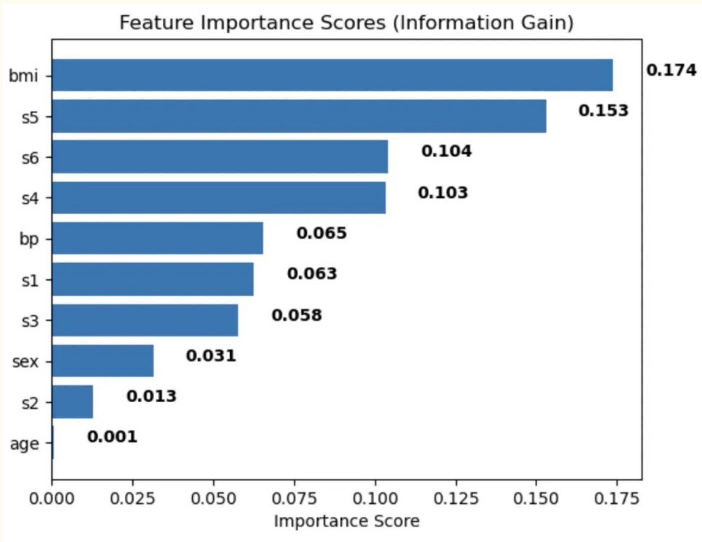
2.4 t-SNE

2.5 Autoencoder

1. Filter Based:

a) Info gain:

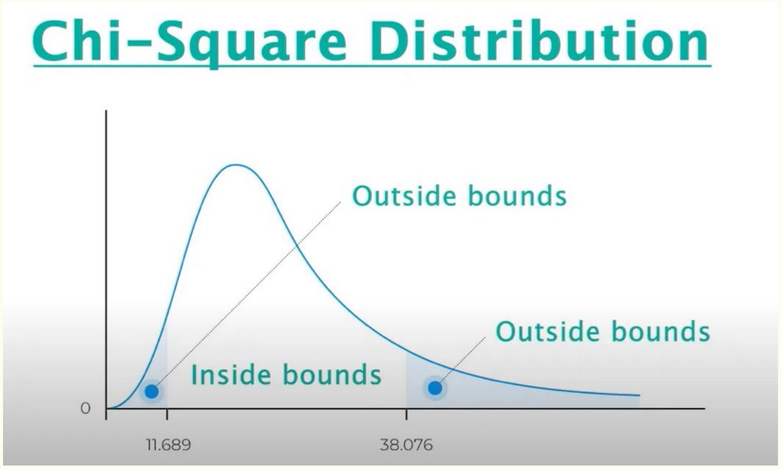
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.feature_selection import mutual_info_regression

# Load the diabetes dataset
data = load_diabetes()

# Split the dataset into features and target
X = data.data
y = data.target

# Apply Information Gain
ig = mutual_info_regression(X, y)

# Create a dictionary of feature importance scores
feature_scores = {}
for i in range(len(data.feature_names)):
    feature_scores[data.feature_names[i]] = ig[i]
# Sort the features by importance score in descending order
sorted_features = sorted(feature_scores.items(), key=lambda x: x[1], reverse=Tru

# Print the feature importance scores and the sorted features
for feature, score in sorted_features:
    print("Feature:", feature, "Score:", score)
# Plot a horizontal bar chart of the feature importance scores
fig, ax = plt.subplots()
y_pos = np.arange(len(sorted_features))
ax.barh(y_pos, [score for feature, score in sorted_features], align="center")
ax.set_yticks(y_pos)
ax.set_yticklabels([feature for feature, score in sorted_features])
ax.invert_yaxis()  # Labels read top-to-bottom
ax.set_xlabel("Importance Score")
ax.set_title("Feature Importance Scores (Information Gain)")

# Add importance scores as labels on the horizontal bar chart
for i, v in enumerate([score for feature, score in sorted_features]):
    ax.text(v + 0.01, i, str(round(v, 3)), color="black", fontweight="bold")
plt.show()
```



b) χ^2 - test:

→ works for categorical data.

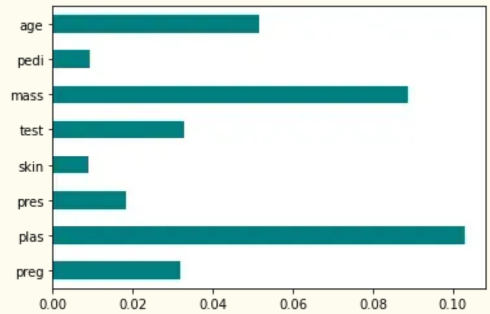
$$X^2 = \Sigma \frac{(O_i - E_i)^2}{E_i}$$



c) fischer's score:

→ way to find out which things are most important in the group.

```
from skfeature.function.similarity_based import fisher_score
import matplotlib.pyplot as plt
%matplotlib inline

# Calculating scores
ranks = fisher_score.fisher_score(X, Y)

# Plotting the ranks
feat_importances = pd.Series(ranks, dataframe.columns[0:len(dataframe.columns)-1
feat_importances.plot(kind='barh', color = 'teal')
plt.show()
```



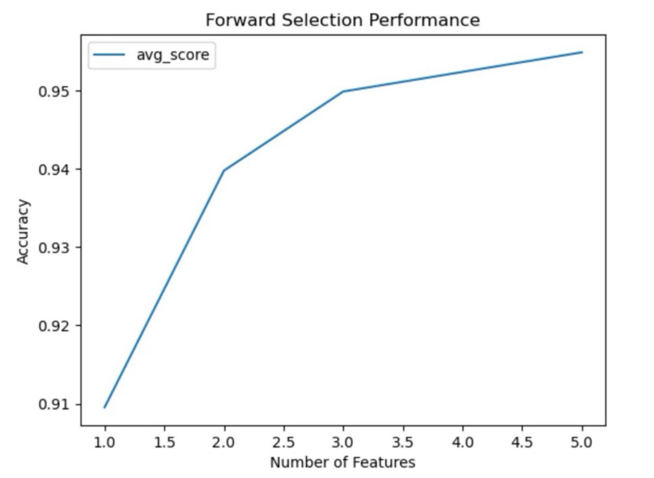
2. Wrapper Based:

a) Forward Selection:

→ starts with an empty set and keep on adding features iterative-ly to improve model's performance.

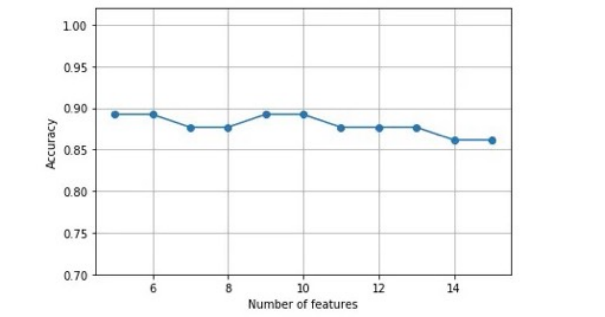
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from mlxtend.feature_selection import SequentialFeatureSelector as SFS

# Load the breast cancer dataset
data = load_breast_cancer()

# Split the dataset into features and target
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Define the logistic regression model
model = LogisticRegression()

# Define the forward selection object
sfs = SFS(model, k_features=5, forward=True, floating=False, scoring="accuracy",

# Perform forward selection on the training set
sfs.fit(X_train, y_train)

# Print the selected features
print("Selected Features:", sfs.k_feature_names_)

# Evaluate the performance of the selected features on the testing set
accuracy = sfs.k_score_
print("Accuracy:", accuracy)

# Plot the performance of the model with different feature subsets
sfs_df = pd.DataFrame.from_dict(sfs.get_metric_dict()).T
sfs_df["avg_score"] = sfs_df["avg_score"].astype(float)
fig, ax = plt.subplots()
sfs_df.plot(kind="line", y="avg_score", ax=ax)
ax.set_xlabel("Number of Features")
ax.set_ylabel("Accuracy")
ax.set_title("Forward Selection Performance")
plt.show()
```

```
Selected Features: ('0', '1', '4', '21', '22')
Accuracy: 0.9548417721518987
```



b. Backward Elimination:

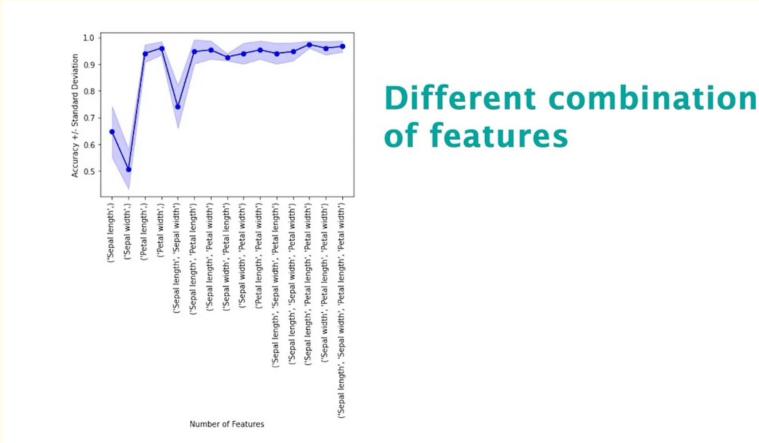
→ starts with full set and keep on removing features until optimal subset is reached



c) Exhaustive feature selection:

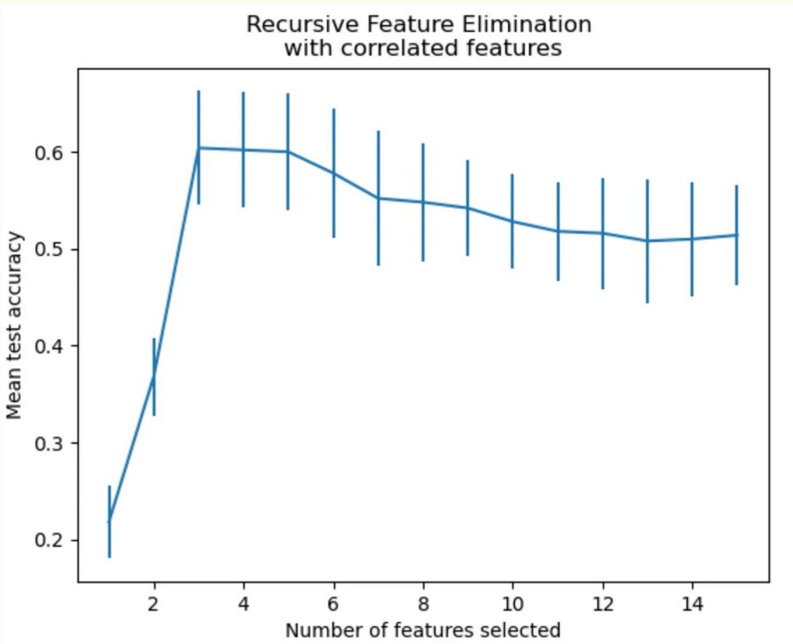
→ tries all possible combination of features.

```python
from mlxtend.feature_selection import ExhaustiveFeatureSelector
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.metrics import roc_auc_score

feature_selector = ExhaustiveFeatureSelector(RandomForestClassifier(n_jobs=-1),
           min_features=2,
           max_features=4,
           scoring='roc_auc',
           print_progress=True,
           cv=2)

features = feature_selector.fit(np.array(train_features.fillna(0)), train_labels

filtered_features= train_features.columns[list(features.k_feature_idx_)]
filtered_features

clf = RandomForestClassifier(n_estimators=100, random_state=41, max_depth=3)
clf.fit(train_features[filtered_features].fillna(0), train_labels)

train_pred = clf.predict_proba(train_features[filtered_features].fillna(0))
print('Accuracy on training set: {}'.format(roc_auc_score(train_labels, train_pr

test_pred = clf.predict_proba(test_features[filtered_features].fillna(0))
print('Accuracy on test set: {}'.format(roc_auc_score(test_labels, test_pred [:,
```



d) Recursive feature elimination:

→ Initially starts with a subset of features then add or remove features based on their importance.



Embedded Approach:

1. Random Forest:

→ Gini impurity calculation is done while buiding the trees.

→ that can be used as feature selection.

```
from sklearn.ensemble import RandomForestClassifier

# create the random forest with your hyperparameters.
model = RandomForestClassifier(n_estimators=340)

# fit the model to start training.
model.fit(X, Y)

# get the importance of the resulting features.
importances = model.feature_importances_

# create a data frame for visualization.
final_df = pd.DataFrame({"Features": pd.DataFrame(X).columns, "Importances":importances})
final_df.set_index('Importances')

# sort in ascending order to better visualization.
final_df = final_df.sort_values('Importances')

# plot the feature importances in bars.
final_df.plot.bar(color = 'teal')
```

<AxesSubplot:>