

Feature Selection Methods for Classification Models

Lecture 09

Prepared by

Dr. Hager Saleh

Activate Windows

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest, chi2, RFE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier
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 matplotlib.pyplot as plt
import seaborn as sns
```

Reading dataset

```
data = pd.read_csv('kidney_disease_clean_final.csv')
data.head()
```

	age	bp	sg	al	su	pc	pcc	ba	bgr	bu	...
0	48.0	80.0	1.020	1.0	0.0	1	0	0	121.000000	36.0	...
1	7.0	50.0	1.020	4.0	0.0	1	0	0	147.943503	18.0	...
2	62.0	80.0	1.010	2.0	3.0	1	0	0	423.000000	53.0	...
3	48.0	70.0	1.005	4.0	0.0	0	1	0	117.000000	56.0	...
4	51.0	80.0	1.010	2.0	0.0	1	0	0	106.000000	26.0	...

	wc	htn	dm	cad	appet	pe	ane	classification
0	72	1	4	1	0	0	0	0
1	56	0	3	1	0	0	0	0
2	70	0	4	1	1	0	1	0
3	62	1	3	1	1	1	1	0
4	68	0	3	1	0	0	0	0

[5 rows x 23 columns]

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

Split dataset into train and test sets

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

Feature Selection Methods

1. Filter Method (Chi-Square Test)

```
select_feature = SelectKBest(chi2,k=8).fit(X_train, y_train)

data = pd.DataFrame([])
data._append(pd.DataFrame({'Feature': X.columns, 'Score':
select_feature .scores_}))
```

	Feature	Score
0	age	89.667221
1	bp	61.592245
2	sg	0.004780
3	al	187.744741
4	su	94.643703

5	pc	1.552639
6	pcc	9.764294
7	ba	2.569833
8	bgr	2339.941276
9	bu	1876.856436
10	sc	292.499959
11	sod	24.497481
12	pot	5.604887
13	hemo	95.418799
14	pcv	120.593956
15	wc	80.785317
16	htn	59.190424
17	dm	4.341060
18	cad	0.754929
19	appet	42.326531
20	pe	37.346939
21	ane	29.255102

```
X_train_selected = select_feature.transform(X_train)
X_test_selected = select_feature.transform(X_test)
```

Applying RF on selected features

```
def evaluate_model(model_name, y_true, y_pred):

    # Calculate metrics
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

    # Output results
    metrics = {
        'Model Name': model_name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
    }

    return metrics
```

RandomForestClassifier

```
# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)
```

```

# Define the hyperparameters for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
    param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the grid search to the data
grid_search.fit(X_train_selected, y_train)

# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_

# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test_selected)

# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test,
    y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
        f"{key}: \n{value}")

# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

Model Name:
RandomForestClassifier
Accuracy: 0.9625
Precision: 0.9661
Recall: 0.9625
F1 Score: 0.9629

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
    'n_estimators': 300}

```

2. Wrapper Method (Recursive Feature Elimination - RFE)

```

from sklearn.feature_selection import RFE

clf_rf_2 = RandomForestClassifier(random_state=43)

```

```
X_train_selected_rfe = rfe_selector.fit_transform(X_train, y_train)
X_test_selected_rfe = rfe_selector.transform(X_test)
```

```
# Get the selected feature indices
```

```
selected_features = rfe_selector.get_support(indices=True)
```

```
print("Selected Features (RFE):", selected_features)
```

```
C:\Users\compumarts\anaconda3\envs\tensorflow\Lib\site-packages\
sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs
failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

Increase the number of iterations (max_iter) or 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 (RFE): [ 3  4 10 12 13 16 19 20]
```

Applying RF on selected features

```
# Define the Random Forest Classifier
```

```
rf_classifier = RandomForestClassifier(random_state=42)
```

```
# Define the hyperparameters for grid search
```

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
# Initialize GridSearchCV
```

```
grid_search = GridSearchCV(estimator=rf_classifier,
    param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
```

```
# Fit the grid search to the data
```

```
grid_search.fit(X_train_selected_rfe, y_train)
```

```
# Get the best estimator from grid search
```

```
best_rf_classifier = grid_search.best_estimator_
```

```
# Make predictions using the best model
```

```
y_pred = best_rf_classifier.predict(X_test_selected_rfe)
```

```
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test,
y_pred)

# Print the evaluation results
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

Model Name:
RandomForestClassifier
Accuracy: 0.9625
Precision: 0.9631
Recall: 0.9625
F1 Score: 0.9626

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5,
'n_estimators': 300}

img = Image.open('P2.png')
display(img)
```

Final Project Steps

1. Download dataset from Kaggle, such as Diabetes, Chronic Kidney Disease (CKD), Heart Disease, and Breast Cancer.
2. Handle missing values appropriately to prepare the datasets for applying machine learning models.
3. Apply machine learning models with a grid search optimizer to the cleaned datasets and evaluate the models' performance.
4. Use two feature selection methods to identify the most important features from the datasets.
5. Apply machine learning models to the selected features and evaluate the models.
6. Save the files code as your full names and upload the code along with the datasets.