

lecture-8

November 22, 2024

[14]:



0.0.1 Import necessary libraries

```
[1]: import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neighbors import KNeighborsRegressor
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

0.0.2 Reading dataset

```
[2]: data = pd.read_csv("WineQT.csv")
data.head()
```

```
[2]:   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides \
0           7.4             0.70         0.00           1.9       0.076
1           7.8             0.88         0.00           2.6       0.098
2           7.8             0.76         0.04           2.3       0.092
3          11.2             0.28         0.56           1.9       0.075
4           7.4             0.70         0.00           1.9       0.076

   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
```

0.0.3 Define features and target variable

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

0.0.4 Scaling features (X)

```
[5]: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

0.0.5 Splitting data

```
[6]: # Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
↪random_state=42)
```

0.0.6 Evaluation models

```
[25]: import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def evaluate_model(model_name, y_true, y_pred):
    """
    Evaluate a regression model using MAE, MSE, RMSE, and R2 score.
    Additionally, plot y_true vs. y_pred with a reference line and display
    ↪ model name and R2 score on the plot.

    Parameters:
        model_name (str): Name of the regression model.
        y_true (array-like): True target values.
        y_pred (array-like): Predicted target values.

    Returns:
        dict: Dictionary containing MAE, MSE, RMSE, and R2 score.
    """
    # Calculate metrics
    mae = mean_absolute_error(y_true, y_pred)
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)

    # Print the evaluation results
    results = {
        "Mean Absolute Error (MAE)": mae,
        "Mean Squared Error (MSE)": mse,
        "Root Mean Squared Error (RMSE)": rmse,
        "R2 Score": r2
    }

    return results
```

0.0.7 RandomForestRegressor

```
[30]: rf_regressor = RandomForestRegressor()

# Define the parameter grid
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_regressor,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation
    scoring='neg_root_mean_squared_error', # Metric to optimize
    verbose=2,
    n_jobs=-1 # Use all available cores
)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE:", grid_search.best_score_)

# Train a new model using the best parameters
best_rf = grid_search.best_estimator_

# Evaluate on the test set
y_pred = best_rf.predict(X_test)
evaluation_results = evaluate_model('RandomForestRegressor', y_test, y_pred)

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

```

Fitting 5 folds for each of 108 candidates, totalling 540 fits
 Best Parameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
 Best CV RMSE: -53.4729106291001

Evaluation Results on Test Set:
 Mean Absolute Error (MAE): 39.8899
 Mean Squared Error (MSE): 2555.9899
 Root Mean Squared Error (RMSE): 50.5568
 R² Score: 0.8485

0.0.8 Decision Tree Regressor

```
[26]: DT = DecisionTreeRegressor(random_state=42)

# Define the parameter grid
param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=DT,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation
    scoring='neg_root_mean_squared_error', # Metric to optimize
    verbose=2,
    n_jobs=-1 # Use all available cores
)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE:", grid_search.best_score_)

# Train a new model using the best parameters
best_DT = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_DT.predict(X_test)
evaluation_results = evaluate_model('Decision Tree Regressor', y_test, y_pred)

# Print the evaluation results
print("\nEvaluation Results on Test Set:")
for key, value in evaluation_results.items():
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: {value}")
```

Fitting 5 folds for each of 36 candidates, totalling 180 fits

Best Parameters: {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split': 10}

Best CV RMSE: -77.20547729998168

Evaluation Results on Test Set:

Mean Absolute Error (MAE): 58.9915

Mean Squared Error (MSE): 5702.1073
Root Mean Squared Error (RMSE): 75.5123
R² Score: 0.6620

0.0.9 Support Vector Regressor (SVR)

```
[27]: SVM = SVR()

# Define the parameter grid
param_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'C': [0.1, 1, 10, 100],
    'epsilon': [0.1, 0.2, 0.5, 1],
    'gamma': ['scale', 'auto']
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=SVM,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation
    scoring='neg_root_mean_squared_error', # Metric to optimize
    verbose=2,
    n_jobs=-1 # Use all available cores
)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE Score:", grid_search.best_score_)

# Train a new model using the best parameters
best_SVM = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_SVM.predict(X_test)
evaluation_results = evaluate_model('Support Vector Regressor', y_test, y_pred)

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

Fitting 5 folds for each of 128 candidates, totalling 640 fits
Best Parameters: {'C': 100, 'epsilon': 0.1, 'gamma': 'scale', 'kernel':

```
'linear']}
```

Best CV RMSE Score: -0.09793542847354482

Evaluation Results on Test Set:

Mean Absolute Error (MAE): 0.0780

Mean Squared Error (MSE): 0.0096

Root Mean Squared Error (RMSE): 0.0979

R² Score: 1.0000

0.0.10 Gradient Boosting Regressor

```
[28]: GBR = GradientBoostingRegressor(random_state=42)

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=GBR,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation
    scoring='neg_root_mean_squared_error', # Metric to optimize
    verbose=2,
    n_jobs=-1 # Use all available cores
)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE Score:", grid_search.best_score_)

# Train a new model using the best parameters
best_GBR = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_GBR.predict(X_test)
evaluation_results = evaluate_model('Gradient Boosting Regressor', y_test, y_pred)
```

```

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

```

Fitting 5 folds for each of 243 candidates, totalling 1215 fits
 Best Parameters: {'learning_rate': 0.2, 'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 300}
 Best CV RMSE Score: -28.934374864941383

Evaluation Results on Test Set:
 Mean Absolute Error (MAE): 22.7533
 Mean Squared Error (MSE): 779.8447
 Root Mean Squared Error (RMSE): 27.9257
 R² Score: 0.9538

0.0.11 KNeighborsRegressor

```

[29]: # Initialize KNeighborsRegressor
knn = KNeighborsRegressor()

# Define the parameter grid
param_grid = {
    'n_neighbors': [3, 5, 7, 10],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski'],
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=knn,
    param_grid=param_grid,
    cv=5, # 5-fold cross-validation
    scoring='neg_root_mean_squared_error', # Metric to optimize
    verbose=2,
    n_jobs=-1 # Use all available cores
)

# Fit the grid search
grid_search.fit(X_train, y_train)

# Get the best parameters and the corresponding score
print("Best Parameters:", grid_search.best_params_)
print("Best CV RMSE Score:", grid_search.best_score_)

# Train a new model using the best parameters

```

```

best_knn = grid_search.best_estimator_

# Evaluate the model on the test set
y_pred = best_knn.predict(X_test)
evaluation_results = evaluate_model('KNeighborsRegressor', y_test, y_pred)

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

```

Fitting 5 folds for each of 24 candidates, totalling 120 fits
 Best Parameters: {'metric': 'euclidean', 'n_neighbors': 7, 'weights': 'distance'}
 Best CV RMSE Score: -64.37327322012888

Evaluation Results on Test Set:
 Mean Absolute Error (MAE): 46.7648
 Mean Squared Error (MSE): 3452.0503
 Root Mean Squared Error (RMSE): 58.7542
 R² Score: 0.7954

[]: