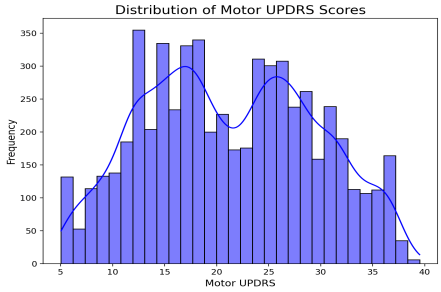
[A2] Comparison of Supervised Learning Algorithms

2021313075 백경인

1. 데이터셋

https://archive.ics.uci.edu/dataset/189/parkinsons+telemonitoring 에서 받아온 데이터로 원격 증상 진행 모니터링을 위한 원격 모니터링 장치의 6 개월 시험에 모집된 초기 파킨슨병 환자 42 명의 다양한 생체의학적 음성 측정으로 구성 되어있다.Data point 개수: 5875 개/ Feature 개수: 16 이며 Label 의 분포는 아래와 같다.



2. 과업의 특성에 따른 적절한 평가지표의 선정:

아무래도 target 이 실수형이라서 MSE 와 MAE 를 고민했으나, 보편적으로 MSE 를 사용하며 이상치가 크게 발견되지 않아 이상치에 민감한 MSE 를 사용하더라도 큰 문제가 없다고 판단했다.

3. 3 개 이상의 학습 알고리즘 비교

LinearRegression, RandomForestRegressor, GradientBoostingRegressor 중에서 RandomForestRegressor, GradientBoostingRegressor, LinearRegression 순으로 성능이 좋았다.

4. 최적의 데이터 전처리 방법:

애초에 비여 있는 값이 없는 데이터를 받아와서 empty value 를 처리할 필요는 없었다. 또한 regression 모델의 target 값이 실수라서 one-hot encoding 도 필요 없었다. 그렇기에 scaling 에만 신경을 썼다. 이를 위해 min-max 와 standard scaling 을 했었는데 minmax 가 더 좋은 성능을 보였다.

5. 최적의 학습 알고리즘:

Random Forest 가 최고의 성능을 보였다. 아무래도 과적합 위험 감소에 용이하고 중요한 feature 를 잘 찾아내는 성향을 갖고 있어서 16 개의 feature 중에서 가장 영향력이 있는 feature 에 집중시킬 수 있게 모델을 짤 수 있었던 것 같다. 또한 여러 feature 들이 target 값과 비선형적인 관계를 갖고 있어 이런면에서 linear model 인 linear regression 보다 우위에 있었던 것 같다. target 이 정규분포와는 살짝 거리가 있는 느낌이라서 Linear Regression 의 MLE 설정시 guassian distribution 을 사용하는 것을 고려했을 때에는 적절하지 않다고 판단했다.

6. 하이퍼파라미터 설정 제시:

Random Forest 같은 경우에는 n_estimators, max_depth, min_samples_split, min_samples_leaf 를 hyperparmeter 로 설정했으며, Gradient boosting 은 추가적으로 learning rate 또한 설정했다. 성능이 가장 좋았던 설정은 Random forest 의 {'n_estimators': 300, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': None} 였다.

7. 추가 성능 개선을 위한 방안 논의:

Cost function 선택에는 자유가 있기에 상황별로 cost function 또한을 학습하는 모델이 있다면 더 scable 한 모델이 성능적인 측면에서 향상을 만들 수 있을 것 같다.

```python
import pandas as pd
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.preprocessing import MinMaxScaler, QuantileTransformer
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import cross_val_score
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Load dataset (Parkinson's UPDRS dataset)
data = pd.read_csv("assignment/HW2/data/parkinsons_updrs.csv")
```

Python

Displaying dataset info

```python
print("Dataset shape:", data.shape)
```

Python

Select features and target

```python
X = data.iloc[:, 6:]  # Columns after the 6th as features
y = data.iloc[:, 4]   # motor_UPDRS as target
```

Python

Set the plot size for better visibility

```python
plt.figure(figsize=(8, 6))
```

Python

Use seaborn's distplot to show both histogram and KDE (Kernel Density Estimation)

```python
sns.histplot(y, kde=True, bins=30, color='blue')
```

Python

Add labels and title

```python
plt.title("Distribution of Motor UPDRS Scores", fontsize=16)
plt.xlabel("Motor UPDRS", fontsize=12)
plt.ylabel("Frequency", fontsize=12)
```

Python

Display the plot

```python
plt.show()
```

Python

Check dataset dimensions

```python
print(f"Number of data points: {X.shape[0]}")
print(f"Number of features: {X.shape[1]}")
```

Python

Splitting data into training and test sets (80% train, 20% test)

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(f"Training set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
```

Python

MinMax Scaling

```python
minmax_scaler = MinMaxScaler()
X_train_scaled = minmax_scaler.fit_transform(X_train)
X_test_scaled = minmax_scaler.transform(X_test)
```

Python

Define a function to train, predict, and evaluate the models

```python
def train_and_evaluate_model(model, model_name):
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    # Calculate metrics
    mse = mean_squared_error(y_test, y_pred)

    print(f"\n{model_name} Performance:")
    print(f"Mean Squared Error (MSE): {mse}")
```

Python

Define model hyperparameter tuning and cross-validation

Random Forest hyperparameters

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

Python

Gradient Boosting hyperparameters

```python
gb_params = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.001, 0.01, 0.1],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

Python

Hyperparameter tuning using RandomizedSearchCV for Random Forest

```python
rf_model = RandomForestRegressor(random_state=42)
rf_random = RandomizedSearchCV(estimator=rf_model, param_distributions=rf_params,
                               n_iter=10, cv=5, random_state=42, n_jobs=-1)
train_and_evaluate_model(rf_random, "Random Forest (Tuned)")
```

Python

Hyperparameter tuning using RandomizedSearchCV for Gradient Boosting

```python
gb_model = GradientBoostingRegressor(random_state=42)
gb_random = RandomizedSearchCV(estimator=gb_model, param_distributions=gb_params,
                               n_iter=10, cv=5, random_state=42, n_jobs=-1)
train_and_evaluate_model(gb_random, "Gradient Boosting (Tuned)")
```

Python

Linear Regression

```python
lr_model = LinearRegression()
train_and_evaluate_model(lr_model, "Linear Regression")
```

Python

Train, predict, and evaluate the models

```python
def train_and_evaluate_model(model, model_name):
    # Fit the model with training data (for RandomizedSearchCV, it also performs hyperparameter tuning)
    model.fit(X_train_scaled, y_train)

    # Best model from RandomizedSearchCV
    if hasattr(model, 'best_estimator_'):
        print(f"Best hyperparameters for {model_name}: {model.best_params_}")
        model = model.best_estimator_  # Update model to best estimator after tuning

    # Predict on the test set
    y_pred = model.predict(X_test_scaled)

    # Calculate metrics
    mse = mean_squared_error(y_test, y_pred)

    # Print results
    print(f"\n{model_name} Performance:")
    print(f"Mean Squared Error (MSE): {mse}")
```

Python

Hyperparameter tuning using RandomizedSearchCV for Random Forest

```python
rf_model = RandomForestRegressor(random_state=42)
rf_random = RandomizedSearchCV(estimator=rf_model, param_distributions=rf_params,
                               n_iter=10, cv=5, random_state=42, n_jobs=-1)
```

Python

Tune and evaluate Random Forest

```python
train_and_evaluate_model(rf_random, "Random Forest (Tuned)")
```

Python

Hyperparameter tuning using RandomizedSearchCV for Gradient Boosting

```python
gb_model = GradientBoostingRegressor(random_state=42)
gb_random = RandomizedSearchCV(estimator=gb_model, param_distributions=gb_params,
                               n_iter=10, cv=5, random_state=42, n_jobs=-1)
```

Python

Tune and evaluate Gradient Boosting

```python
train_and_evaluate_model(gb_random, "Gradient Boosting (Tuned)")
```

Python

Evaluate Linear Regression

```python
lr_model = LinearRegression()
train_and_evaluate_model(lr_model, "Linear Regression")
```

Python