PRESENTATION

2024.12.19

기계학습 프로젝트

하이퍼 파라미터 튜닝을 통한 유방암 진단 모델 성능 개선

컴퓨터공학부
202204495 홍창희

목차

프로젝트 개요

유방암은 전 세계적으로 중요한 건강 문제 중 하나로,
조기 진단은 환자의 생존율을 크게 향상시킬 수 있습니다.

본 프로젝트는 머신러닝 기법들을 활용하여 유방암 진단 모델을 설계하여, 각 모델마다 하이퍼파라미터 튜닝을 통해 이뤄지는 성능 개선을 측정하고, 그에 대한 각 모델에 대한 성능을 비교하여 가장 적합한 모델을 선정하는 것을 목표로 진행했습니다.

이를 위해 UCI Breast Cancer Wisconsin 데이터셋을 활용하였고, 로지스틱 회귀, 랜덤 포레스트, 신경망 모델을 통해 진단 모델을 각각 구현했습니다.

데이터셋

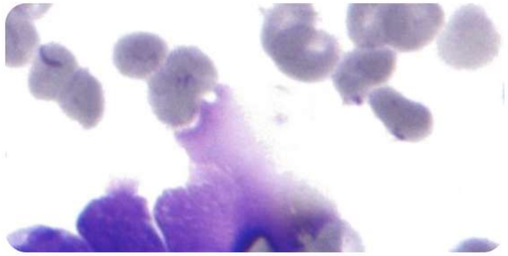
- 데이터셋 설명

UCI MACHINE LEARNING AND 1 COLLABORATOR · UPDATED 8 YEARS AGO

3677 | New Notebook | Download

Breast Cancer Wisconsin (Diagnostic) Data Set

Predict whether the cancer is benign or malignant



유방암 종양의 세포 핵의 10가지 물리적 특성으로 구성된 Kaggle에서 제공하는 데이터셋
레이블로 양성(B)과 음성(M)을 갖는 테이블 형태의 csv파일

데이터셋

- 데이터 전처리
  - 결측치 처리
  - 불필요한 컬럼 제거
  - 라벨 인코딩
  - 특성과 라벨 분리
  - 정규화, 훈련/테스트 데이터 분리

```
# 1. 결측치 처리
# 수치형 데이터와 범주형 데이터 구분
numerical_cols = data.select_dtypes(include=['float64', 'int64']).columns
categorical_cols = data.select_dtypes(include=['object']).columns

# 결측치 대체
for col in numerical_cols:
    data[col].fillna(data[col].median(), inplace=True)
for col in categorical_cols:
    data[col].fillna(data[col].mode()[0], inplace=True)

# 2. ID 제거
data = data.drop(columns=['ID'])

# 3. 라벨 인코딩
data['Diagnosis'] = data['Diagnosis'].map({'M': 1, 'B': 0})

# 4. 특성과 라벨 분리
X = data.iloc[:, 1:]
y = data['Diagnosis']

# 5. 데이터 스케일링 및 학습/테스트 데이터 분리
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
```

머신러닝 모델 설계

- 로지스틱 회귀

```
# ---------------------- 로지스틱 회귀 ----------------------
# 기본 모델
lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)
evaluation_results.append(evaluate_model(y_test, lr_predictions, "Logistic Regression"))

# 하이퍼파라미터 튜닝 적용 모델
lr_param_dist = {
    'C': np.logspace(-4, 4, 20),
    'solver': ['liblinear', 'lbfgs', 'newton-cg', 'saga']
}
lr_random_search = RandomizedSearchCV(
    estimator=LogisticRegression(max_iter=1000, random_state=42),
    param_distributions=lr_param_dist,
    n_iter=10,
    scoring='accuracy',
    cv=5,
    random_state=42
)
lr_random_search.fit(X_train, y_train)
best_lr_model = lr_random_search.best_estimator_
best_lr_predictions = best_lr_model.predict(X_test)
evaluation_results.append(evaluate_model(y_test, best_lr_predictions, "Logistic Regression", tuned=True))
```

머신러닝 모델 설계

- 랜덤포레스트

```
# ---------------------- 랜덤포레스트 ----------------------
# 기본 모델
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
evaluation_results.append(evaluate_model(y_test, rf_predictions, "Random Forest"))

# 하이퍼파라미터 튜닝 적용 모델
rf_param_dist = {
    'n_estimators': random.sample(range(50, 201), min(10, len(range(50, 201)))),
    'max_depth': random.sample(range(5, 21), min(10, len(range(5, 21)))),
    'min_samples_split': random.sample(range(2, 11), min(10, len(range(2, 11))))
}
rf_random_search = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_distributions=rf_param_dist,
    n_iter=10,
    scoring='accuracy',
    cv=5,
    random_state=42
)
rf_random_search.fit(X_train, y_train)
best_rf_model = rf_random_search.best_estimator_
best_rf_predictions = best_rf_model.predict(X_test)
evaluation_results.append(evaluate_model(y_test, best_rf_predictions, "Random Forest", tuned=True))
```

머신러닝 모델 설계

- 신경망 모델

```
# ---------------------- 신경망 모델 ----------------------
# 기본 모델
base_nn_model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.5),
    Dense(32, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
base_nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
base_nn_model.fit(X_train, y_train, validation_split=0.2, epochs=10, batch_size=32, verbose=0)
base_nn_predictions = (base_nn_model.predict(X_test) > 0.5).astype("int32")
evaluation_results.append(evaluate_model(y_test, base_nn_predictions, "Neural Network"))

# 하이퍼파라미터 튜닝 적용 모델
nn_param_distributions = {
    'patience': range(1, 21),
    'dropout_rate': [0.2, 0.3, 0.4, 0.5, 0.6]
}
results = []
for _ in range(10):
    params = {
        'patience': random.choice(nn_param_distributions['patience']),
        'dropout_rate': random.choice(nn_param_distributions['dropout_rate'])
    }
    early_stopping = EarlyStopping(monitor='val_loss', patience=params['patience'], restore_best_weights=True)
    nn_model = Sequential([
        Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
        Dropout(params['dropout_rate']),
        Dense(32, activation='relu'),
        Dropout(params['dropout_rate']),
        Dense(1, activation='sigmoid')
    ])
    nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    nn_model.fit(X_train, y_train, validation_split=0.2, epochs=100, batch_size=32, callbacks=[early_stopping], verbose=0)
    nn_predictions = (nn_model.predict(X_test) > 0.5).astype("int32")
    test_accuracy = accuracy_score(y_test, nn_predictions)
    results.append({'params': params, 'accuracy': test_accuracy, 'predictions': nn_predictions})

best_nn_result = max(results, key=lambda x: x['accuracy'])
evaluation_results.append(evaluate_model(y_test, best_nn_result['predictions'], "Neural Network", tuned=True))
```

성능 평가

- 평가 지표

  - Accuracy (정확도): 모델이 전체 데이터를 얼마나 잘 예측했는지를 나타냄
  - Precision (정밀도): 모델이 양성으로 예측한 샘플 중 실제로 양성인 샘플의 비율을 나타냄
  - Recall (재현율): 실제 양성인 샘플 중에서 모델이 올바르게 양성으로 예측한 비율을 나타냄
  - F1-Score: 정밀도와 재현율의 균형을 얼마나 잘 맞췄는지 나타냄 (둘의 조화 평균)

성능 평가

- 모델 성능 결과

[Model Performance Summary]

	Model	Accuracy	Precision	Recall	F1-Score
0	Logistic Regression (Base)	0.965	1.000	0.905	0.950
1	Logistic Regression (Tuned)	0.965	1.000	0.905	0.950
2	Random Forest (Base)	0.974	0.976	0.952	0.964
3	Random Forest (Tuned)	0.982	1.000	0.952	0.976
4	Neural Network (Base)	0.965	0.975	0.929	0.951
5	Neural Network (Tuned)	0.982	1.000	0.952	0.976

하이퍼파라미터 튜닝 적용 전 후로, 로지스틱 회귀 모델은 차이가 없음
랜덤포레스트와 신경망 모델은 성능이 개선됨
가장 높은 성능을 보이는 모델은 랜덤포레스트와 신경망 모델

결론

- 하이퍼파라미터 튜닝의 영향

  - 로지스틱회귀는 단순하기에 튜닝이 크게 영향 X,
    랜덤포레스트와 신경망 모델은 큰 성능 개선 O

- 모델 선정

  - 랜덤포레스트와 신경망 모델은 동일한 점수를 보이지만,
    차후에 복잡한 데이터셋을 학습할 시 신경망 모델이 은닉
    층을 추가해 더 높은 성능을 기대할 수 있기에, 최종적인
    적합한 모델로 신경망 모델 선정

- 결과에 대한 고찰

  - 랜덤 서치의 무작위성으로 인해 매번 결과가 달라져 성능 평과 결과의 일관성과 신뢰도가 떨어짐
    추후 시간적 여유가 확보되면 결과 도출 횟수를 늘려 통계적 분석을 통해 더 유의미한 성능 비교를 이루고자 함