pr-10

October 15, 2023

1 Regression Algortihms: Decision Tree, Random Forest, Logistic Regression, Linear Regression, KNN, SVM, Naive Bayes, Gradient Boosting, XGBoost, LightGBM, CatBoost

[4]:

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = load_diabetes()

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.
 ↪target, test_size=0.2, random_state=0)

# Create a decision tree regressor object
regressor = DecisionTreeRegressor(random_state=0)

# Fit the regressor with training data
regressor.fit(X_train, y_train)

# Make predictions using the test set
y_pred = regressor.predict(X_test)

# Calculate and print metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2 ): {r2}")
```

```
Mean Squared Error (MSE): 6891.797752808989
R-squared (R2 ): -0.34397344448845835
```

[5]:
```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = load_diabetes()

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.
 ↪target, test_size=0.2, random_state=0)

# Create a random forest regressor object
regressor = RandomForestRegressor(random_state=0, n_estimators=100)

# Fit the regressor with the training data
regressor.fit(X_train, y_train)

# Make predictions using the test set
y_pred = regressor.predict(X_test)

# Calculate and print metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2 ): {r2}")
```

```
Mean Squared Error (MSE): 3750.300122471911
R-squared (R2 ): 0.26865181564422547
```

[7]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the breast cancer dataset
data = load_breast_cancer()

# The target variable is more suitable for logistic regression since it's␣
 ↪categorical
X, y = data.data, data.target

# Split the dataset into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

```python
# Create a logistic regression classifier
classifier = LogisticRegression(max_iter=10000, random_state=0)

# Fit the classifier with the training data
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)

# Calculate and print metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2 ): {r2}")
```

```
Mean Squared Error (MSE): 0.05263157894736842
R-squared (R2 ): 0.7827881867259447
```

[8]:

```python
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = load_diabetes()

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.
 ↪target, test_size=0.2, random_state=0)

# Create a linear regression object
regressor = LinearRegression()

# Train the model using the training sets
regressor.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regressor.predict(X_test)

# The coefficients
print('Coefficients: \n', regressor.coef_)
# The mean squared error
print('Mean squared error: %.2f' % mean_squared_error(y_test, y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f' % r2_score(y_test, y_pred))
```

```
Coefficients:
 [ -35.55025079 -243.16508959  562.76234744  305.46348218 -662.70290089
  324.20738537   24.74879489  170.3249615   731.63743545   43.0309307 ]
Mean squared error: 3424.26
Coefficient of determination: 0.33
```

[9]:
```python
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = load_diabetes()

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.
 ↪target, test_size=0.2, random_state=0)

# Create KNN regressor object
regressor = KNeighborsRegressor(n_neighbors=5)  # here, 5 is the number of␣
 ↪neighbors

# Train the model using the training sets
regressor.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regressor.predict(X_test)

# Calculate and print metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2 ): {r2}")
```

```
Mean Squared Error (MSE): 4243.422022471909
R-squared (R2 ): 0.1724878302420758
```

[10]:
```python
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

# Load the diabetes dataset
diabetes = load_diabetes()
```

```
# It's a good practice to scale the data for SVM models
scaler = StandardScaler()
X_scaled = scaler.fit_transform(diabetes.data)

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X_scaled, diabetes.target, test_size=0.2, random_state=0)

# Create SVR object
regressor = SVR(kernel='rbf')  # kernel can also be 'linear', 'poly', etc.

# Train the model using the training sets
regressor.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regressor.predict(X_test)

# Calculate and print metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2 ): {r2}")
```

```
Mean Squared Error (MSE): 4470.939682846807
R-squared (R2 ): 0.12811948040601506
```

[11]:
```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = load_diabetes()

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.target, test_size=0.2, random_state=0)

# Create Gradient Boosting Regressor object
regressor = GradientBoostingRegressor(random_state=0)

# Train the model using the training sets
regressor.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regressor.predict(X_test)
```

```python
# Calculate and print metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2 ): {r2}")
```

```
Mean Squared Error (MSE): 4071.9510461055215
R-squared (R2 ): 0.20592648398710256
```

[12]:
```python
import xgboost as xgb
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = load_diabetes()

# Split the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(diabetes.data, diabetes.
 ↪target, test_size=0.2, random_state=0)

# Convert the dataset into an optimized data structure called Dmatrix that␣
 ↪XGBoost supports
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Specify the parameters
params = {
    'max_depth': 3,  # the maximum depth of each tree
    'eta': 0.3,  # the training step for each iteration
    'objective': 'reg:squarederror',  # error evaluation for regression training
    'eval_metric': 'rmse'  # evaluation metric for validation data
}

# Specify validation set to watch performance
watchlist = [(dtest, 'eval'), (dtrain, 'train')]

num_round = 50  # the number of training iterations

# Train the model
bst = xgb.train(params, dtrain, num_round, watchlist)

# Make predictions
y_pred = bst.predict(dtest)
```

```python
# Calculate and print metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2 ): {r2}")
```

```
[0]	eval-rmse:124.75405	train-rmse:125.72972
[1]	eval-rmse:96.98714	train-rmse:95.93678
[2]	eval-rmse:79.56656	train-rmse:76.40035
[3]	eval-rmse:70.01961	train-rmse:63.83979
[4]	eval-rmse:64.08181	train-rmse:56.05293
[5]	eval-rmse:62.62043	train-rmse:50.80412
[6]	eval-rmse:61.49554	train-rmse:47.72474
[7]	eval-rmse:61.78865	train-rmse:45.38086
[8]	eval-rmse:62.51831	train-rmse:43.74025
[9]	eval-rmse:62.43861	train-rmse:42.50393
[10]	eval-rmse:62.37349	train-rmse:41.22457
[11]	eval-rmse:62.28799	train-rmse:40.64192
[12]	eval-rmse:62.40342	train-rmse:39.84432
[13]	eval-rmse:62.46278	train-rmse:39.03206
[14]	eval-rmse:62.62018	train-rmse:38.32677
[15]	eval-rmse:63.19474	train-rmse:37.73889
[16]	eval-rmse:63.00371	train-rmse:37.24875
[17]	eval-rmse:63.29954	train-rmse:36.68275
[18]	eval-rmse:63.72195	train-rmse:35.91110
[19]	eval-rmse:63.80423	train-rmse:35.67652
[20]	eval-rmse:63.94451	train-rmse:35.47408
[21]	eval-rmse:64.22036	train-rmse:35.24786
[22]	eval-rmse:64.65215	train-rmse:34.59054
[23]	eval-rmse:64.54772	train-rmse:34.29866
[24]	eval-rmse:64.49398	train-rmse:33.81683
[25]	eval-rmse:64.41455	train-rmse:33.47276
[26]	eval-rmse:64.38479	train-rmse:33.30762
[27]	eval-rmse:64.36752	train-rmse:33.18210
[28]	eval-rmse:64.51208	train-rmse:32.95136
[29]	eval-rmse:64.50413	train-rmse:32.80177
[30]	eval-rmse:64.56561	train-rmse:32.69032
[31]	eval-rmse:64.11282	train-rmse:32.19289
[32]	eval-rmse:64.10381	train-rmse:31.87899
[33]	eval-rmse:64.66535	train-rmse:31.35188
[34]	eval-rmse:64.05249	train-rmse:30.73868
[35]	eval-rmse:64.49111	train-rmse:30.10054
[36]	eval-rmse:64.62129	train-rmse:29.73208
[37]	eval-rmse:65.24570	train-rmse:29.29309
[38]	eval-rmse:65.42075	train-rmse:28.95054
[39]	eval-rmse:65.24877	train-rmse:28.79885
```

```
[40]    eval-rmse:65.34644      train-rmse:28.67444
[41]    eval-rmse:65.51337      train-rmse:28.54005
[42]    eval-rmse:65.56194      train-rmse:28.17597
[43]    eval-rmse:65.48602      train-rmse:27.70553
[44]    eval-rmse:65.65388      train-rmse:27.58791
[45]    eval-rmse:65.54153      train-rmse:27.27064
[46]    eval-rmse:65.65069      train-rmse:26.66575
[47]    eval-rmse:65.69711      train-rmse:26.25560
[48]    eval-rmse:65.73345      train-rmse:26.14692
[49]    eval-rmse:65.79154      train-rmse:25.95848
Mean Squared Error (MSE): 4328.527010424716
R-squared (R2 ): 0.15589145758220457
```

```
/opt/homebrew/lib/python3.11/site-packages/xgboost/core.py:617: FutureWarning:
Pass `evals` as keyword args.
  warnings.warn(msg, FutureWarning)
```