assignment 7 final

March 27, 2024

1 EET 4501 Applied Machine learning

1.1 Assignment 7 – KNN and Linear Regression for Robotic Arm Control

1.1.1 In this assignment, you will use two different algorithms, linear regression and KNN, to build a inverse kinematic machine learning-based model. The model will obtain the joint angles (q1, ..., q6) of a robot arm based on a given end-effector position (x,y,z).

Implementation Steps:

```
1) Load the provided dataset.
2) Get familiar with the data (identify the columns of the joint angles and the space-coordinat
3) Split the data in training and testing.
4) Use linear regression algorithm to train a model for inverse kinematics (to predict the joi
5) Predict over the testing dataset.
6) Measure the performance of the linear regression algorithm for inverse kinematics using the
7) Repeat steps 4) - 6) using KNN algorithm for regression.
8) Compare both performances and provide a conclusion about which algorithm perform better for
```

Here is a suggested template for this assigment, feel free to modify it to complete the tasks

[1]:
```
##Import libraries
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

##Import ML algorithm functions (from sklearn....    )
from sklearn.linear_model import Ridge
from sklearn.neighbors import KNeighborsRegressor

##Import the functions to meassure the performance (from sklearn.... )
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

[2]:
```
## loading the dataset using the numpy library
data = np.loadtxt("robot_inverse_kinematics_dataset.csv", skiprows=1,␣
 ↪delimiter=',')
print(data.shape)
print(type(data))
```

```
(15000, 9)
<class 'numpy.ndarray'>
```

[3]:
```
## Check the values of data
print(data)
```

```
[[-1.51    -0.763    1.85    … -0.0947   0.15     0.301  ]
 [-2.84     0.52     1.58    …  0.142   -0.1      0.225  ]
 [-1.23     0.695    1.22    … -0.0833   0.223    0.206  ]
 …
 [ 2.62     1.41     1.56    …  0.131   -0.16     0.362  ]
 [-1.89     1.85     1.51    …  0.0829  -0.016    0.441  ]
 [ 2.68    -1.79     1.79    … -0.157   -0.00369  0.254  ]]
```

[4]:
```
# Split the input and output variables
X = data[:, :6]  # input features (joint angles)
y = data[:, 6:]  # output variables (end-effector positions)
```

[5]:
```
# Split data into training and validation sets (suggested ration 8:2, suggested␣
 ↪random_state=42)

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

[6]:
```
# Normalizing the feature dataset
scaler = MinMaxScaler()

## Fit the scaler
scaler.fit(X_train)

## scale the end-effector position on the different splits

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[7]:
```
# Print shapes of training and validation sets

train_test_shapes = {
    'Training set shapes': {
        'X_train': X_train_scaled.shape,
        'y_train': y_train.shape
    },
    'Testing set shapes': {
        'X_test': X_test_scaled.shape,
        'y_test': y_test.shape
    }
}
```

[8]:
```
## Define the linear regression (lr) model, feel free to explore different␣
 ↪values for the hyperparameter alpha
lr_model = Ridge(alpha=1.0)  # alpha is a hyperparameter that controls the␣
 ↪regularization strength
# Fit the model to the training data
lr_model.fit(X_train_scaled, y_train)
```

[8]:
```
Ridge()
```

[9]:
```
## Predict using the lr model from the testing dataset
y_pred_lr = lr_model.predict(X_test_scaled)
```

[10]:
```
## Inspect the values prining some samples of joint-angles (y_test) and the␣
 ↪predictions generated by the model (predictions - lr)
samples_lr = np.column_stack((y_test[:5], y_pred_lr[:5]))
```

[11]:
```
# Calculate the performance metrics - lr
mse_lr = mean_squared_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mse_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
```

[12]:
```
## Define the KNN regressor model, feel free to explore different values for␣
 ↪the hyperparameter k
knn_model = KNeighborsRegressor(n_neighbors=5)

## Fit the model to the data (training)
knn_model.fit(X_train_scaled, y_train)
```

[12]:
```
KNeighborsRegressor()
```

[13]:
```
## Predict using the KNN regressor model from the testing dataset
y_pred_knn = knn_model.predict(X_test_scaled)
```

[14]:
```
## Inspect the values prining some samples of joint-angles (y_test) and the␣
 ↪predictions generated by the model (predictions -knn)
samples_knn = np.column_stack((y_test[:5], y_pred_knn[:5]))
```

[15]:
```
# Calculate the performance metrics - knn
mse_knn = mean_squared_error(y_test, y_pred_knn)
rmse_knn = np.sqrt(mse_knn)
mae_knn = mean_absolute_error(y_test, y_pred_knn)
r2_knn = r2_score(y_test, y_pred_knn)
(train_test_shapes, samples_lr, mse_lr, rmse_lr, mae_lr, r2_lr, samples_knn,␣
 ↪mse_knn, rmse_knn, mae_knn, r2_knn)
```

```
[15]: ({'Training set shapes': {'X_train': (12000, 6), 'y_train': (12000, 3)},
        'Testing set shapes': {'X_test': (3000, 6), 'y_test': (3000, 3)}},
       array([[-0.0714    ,  0.000885  ,  0.122     ,  0.00137629, -0.02624326,
                0.17981277],
              [ 0.0968    , -0.0333    ,  0.0924    , -0.00057815,  0.00313054,
                0.14939925],
              [-0.117     , -0.17      ,  0.134     , -0.00518846, -0.02274725,
                0.23366154],
              [-0.0665    ,  0.143     ,  0.358     , -0.01260666,  0.03533981,
                0.35257999],
              [ 0.0441    , -0.0179    ,  0.356     , -0.01421681,  0.03486993,
                0.31908794]]),
       0.009425905461054875,
       0.09708710244442809,
       0.08079842214193875,
       0.16201800175330106,
       array([[-0.0714  ,  0.000885,  0.122   , -0.067278,  0.00352 ,  0.1262  ],
              [ 0.0968  , -0.0333  ,  0.0924  ,  0.09622 , -0.03129 ,  0.1186  ],
              [-0.117   , -0.17    ,  0.134   , -0.06585 , -0.14404 ,  0.157516],
              [-0.0665  ,  0.143   ,  0.358   , -0.08112 ,  0.12022 ,  0.3652  ],
              [ 0.0441  , -0.0179  ,  0.356   ,  0.09272 , -0.00304 ,  0.37    ]]),
       0.0013342453897832226,
       0.03652732387929922,
       0.02853730300000001,
       0.8812164054614756)
```

2 Provide a conclussion of your findings

Training and Testing Set Shapes: Training set: 12,000 samples with 6 features for input (X_train) and 3 features for output (y_train) Testing set: 3,000 samples with 6 features for input (X_test) and 3 features for output (y_test) Linear Regression Model (Using Ridge Regression): Mean Squared Error (MSE): 0.00943 Root Mean Squared Error (RMSE): 0.09709 Mean Absolute Error (MAE): 0.08080 R^2 Score: 0.16202 Sample Predictions: For the first five test samples, the predictions differ from the actual values, which reflects the modest R^2 score. K-Nearest Neighbors (KNN) Model: Mean Squared Error (MSE): 0.00133 Root Mean Squared Error (RMSE): 0.03653 Mean Absolute Error (MAE): 0.02854 R^2 Score: 0.88122 Sample Predictions: The predictions for the first five test samples are much closer to the actual values, indicated by the higher R^2 score and lower error metrics. Conclusion: Based on the performance metrics, the KNN model significantly outperforms the Linear Regression model for this task. The much lower MSE, RMSE, and MAE, alongside a substantially higher R^2 score, suggest that the KNN model has a better fit for the dataset provided. It is able to more accurately predict the end-effector positions of the robotic arm based on the given joint angles.

In practical terms, for the task of inverse kinematics in robotics, the KNN model would likely provide more precise control and positioning of the robotic arm, which could be crucial in applications requiring high accuracy, such as in assembly lines or surgical robots. It's also important to note that the hyperparameters for both models (the alpha for Ridge and n_neighbors for KNN) were

set to default values; tuning these could potentially improve the performance of the models further.

Here are some useful sources:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html