دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

مبانی سیستم های هوشمند

گزارش پایانترم

مهدی خلیل‌زاده

۹۹۳۲۲۱۳

استاد : آقای دکتر مهدی علیاری

۳۰ دی ۱۴۰۳

فهرست مطالب

عنوان شماره صفحه

چکیده

در این گزارش، تحلیل، راه‌حل و کدهای مربوط به سوالان آزمون پایانی درس آورده شده‌اند. همچنین صفحه گوگل کولب سوالات زیر در زیر آمده است:

لینک کولب

بخش ۱: سوالات هاهنگ شده

پرسش یک – الف)

در یک مسئله طبقه‌بندی شده، اگر $y > 0$ باشد؛ کلاس 1 و در غیر این صورت کلاس 0 می‌باشد، مرز تصمیم‌گیری را رسم نمایید.

$$y = -e^{-\|x-t_1\|^2} - e^{-\|x-t_2\|^2} + 1$$

مرز تصمیم گیری با حل معادله $y = 0$ به دست می‌آید. پس با جایگذاری داریم:

$$y = 0 \rightarrow 0 = -e^{-\|x-t_1\|^2} - e^{-\|x-t_2\|^2} + 1 \rightarrow 1 = e^{-\|x-t_1\|^2} + e^{-\|x-t_2\|^2} \quad [1]$$

در این معادله مرز تصمیم گیری یک خط مستقیم نخواهد بود و یک منحنی است که شکل آن به پارامترهای t1 و t2 بستگی دارد. تابع $\|x - t_1\|^2$ در واقع همان فاصله اقلیدسی x از t1 است و $e^{-\|x-t_1\|^2}$ این فاصله را به احتمال تعلق به t1 تبدیل می‌کند. پس معادله [1] نشان‌دهنده‌ی مجموعه‌ای از نقاط x است که در آن‌ها مجموع دو تابع گاوسی (که هر کدام حول نقاط t1 و t2 متمرکز شده‌اند) برابر با ۱ است.

حالا اگر x به t1 یا t2 نزدیک باشد، مقدار $e^{-\|x-t_1\|^2}$ یا $e^{-\|x-t_2\|^2}$ به ۱ نزدیک می‌شود. بنابراین، مجموع این دو مقدار ممکن است از ۱ بیشتر شود و y مثبت شود (کلاس ۱). و اگر x از هر دو نقطه t1 و t2 دور باشد، هر دو مقدار $e^{-\|x-t_1\|^2}$ و $e^{-\|x-t_2\|^2}$ کوچک خواهند بود و مجموع آن‌ها کمتر از ۱ خواهد شد، بنابراین y منفی می‌شود (کلاس ۰).

همچنین اگر t1 و t2 به هم نزدیک باشند، مرز تصمیم‌گیری ممکن است به شکل یک دایره یا بیضی حول این دو نقطه باشد. اگر t1 و t2 از هم دور باشند، مرز تصمیم‌گیری ممکن است به شکل دو دایره جداگانه حول هر یک از نقاط باشد. اگر هم خیلی به هم نزدیک باشند ممکن است همپوشانی داشته باشیم و مرز خوب مشخص نشود. به همین علت برای نقاط t1 و t2 و نقاط (1, 0) و (-1, 0) را فرض شدند.

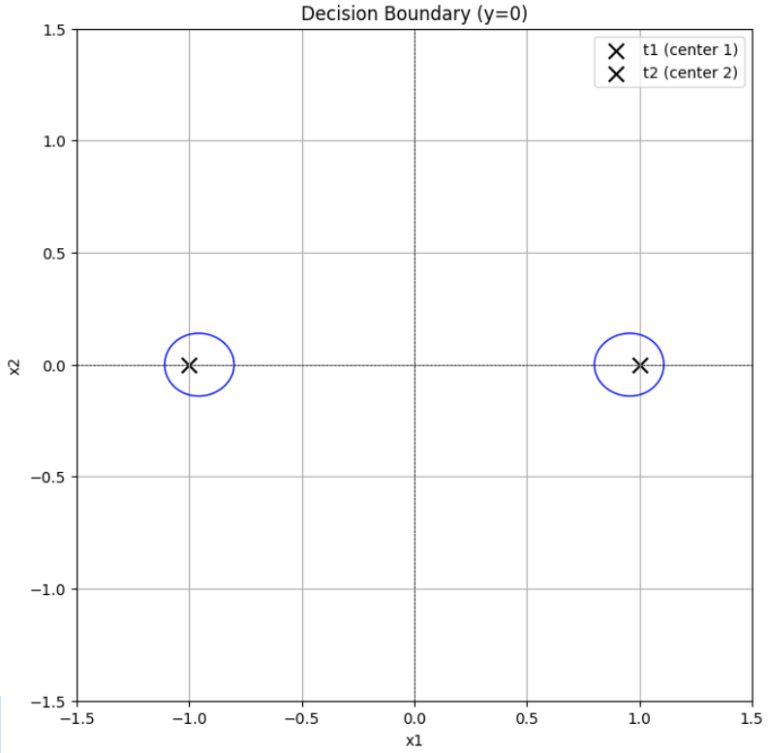
حالا با کد پایتون ابتدا نقاط t1 و t2 مشخص شدند:

```
# Define the centers t1 and t2
t1 = np.array([1, 0])
t2 = np.array([-1, 0])
```

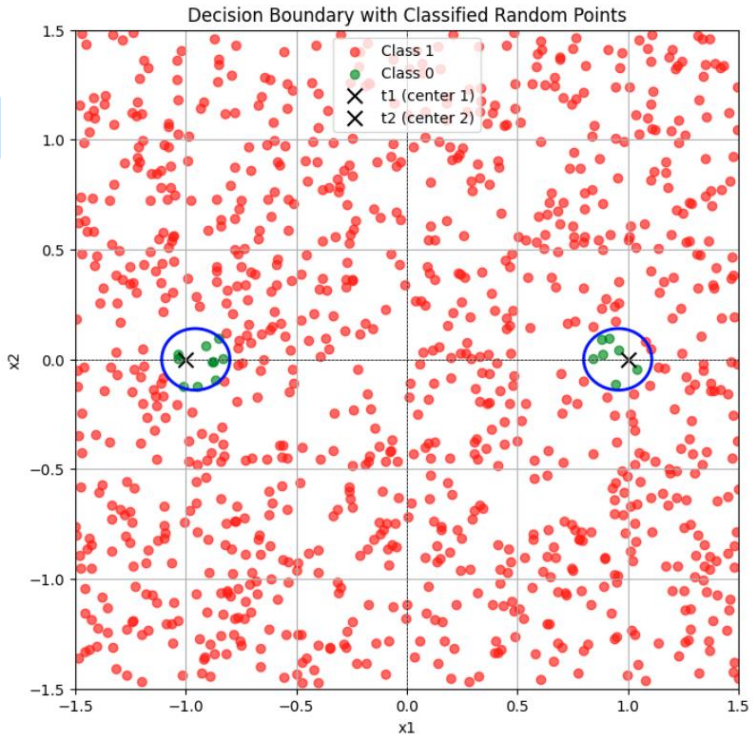
سپس نقاط تعریف شدند و با آن‌ها یک meshgrid تعریف شد و در ادامه تابع مرز تصمیم‌گیری تعریف شد:

```
y = -np.exp(-np.linalg.norm(X - t1, axis=2)**2) - np.exp(-np.linalg.norm(X - t2, axis=2)**2) + 1
```

شکل مرز تصمیم‌گیری به صورت زیر درآمد:



همچنین در ادامه تعدادی نقاط تصادفی تولید شدند و با استفاده از مرز تصمیم گیری به دست آمده طبقه‌بندی شدند:



پرسش یک - ب)

چرا در شبکه RBF، تعیین میزان گستردگی توابع پایه شعاعی ($\boldsymbol{Spreads}$ یا $\boldsymbol{\sigma}$) از طریق رابطه زیر بهترین جواب را می‌دهد؟

$$\boldsymbol{\sigma = \frac{Maximum\ distance\ between\ any\ 2\ centers}{\sqrt{number\ of\ centers}} = \frac{d_{max}}{\sqrt{m_l}}}$$

رابطه $\sigma = \frac{d_{max}}{\sqrt{m_l}}$ به دلیل سادگی و تعادل در تطبیق گستردگی توابع پایه شعاعی (RBF) با تعداد مراکز (m_l) و توزیع آن‌ها در فضای ویژگی، یک انتخاب مناسب برای تعیین میزان گستردگی است. این رابطه با استفاده از d_{max}، یعنی بیشترین فاصله بین هر دو مرکز، تضمین می‌کند که σ با مقیاس داده‌ها و توزیع مراکز هماهنگ باشد. علاوه بر این، تقسیم d_{max} بر $\sqrt{m_l}$ باعث می‌شود که σ به طور خودکار با تعداد مراکز تنظیم شود. هر چه تعداد مراکز بیشتر باشد، توابع شعاعی محدودتر می‌شوند تا از هم‌پوشانی بیش از حد جلوگیری شود، و بالعکس.

این رابطه به جلوگیری از مشکلاتی مانند Overfitting و Underfitting کمک می‌کند. اگر σ بیش از حد کوچک باشد، توابع شعاعی بسیار محلی خواهند شد و مدل تنها داده‌های آموزشی را یاد می‌گیرد و قابلیت تعمیم به داده‌های جدید را از دست می‌دهد .(Overfitting) از طرف دیگر، اگر σ بیش از حد بزرگ باشد، توابع شعاعی بیش از حد گسترده خواهند شد و مدل جزئیات داده‌ها را نادیده می‌گیرد .(Underfitting) این رابطه با تنظیم σ به گونه‌ای که متناسب با توزیع مراکز و تعداد آن‌ها باشد، این مشکلات را کاهش می‌دهد.

در عمل، این رابطه تضمین می‌کند که توابع RBF فضای ویژگی را به صورت بهینه پوشش دهند. به طور خاص، d_{max} مقیاس کلی فاصله‌های مراکز را نشان می‌دهد و $\sqrt{m_l}$ یک عامل نرمال‌سازی است که به مدل کمک می‌کند تا به طور متناسب با تعداد مراکز عمل کند. این ویژگی‌ها باعث می‌شود که این رابطه یک انتخاب استاندارد و مؤثر برای تنظیم گستردگی توابع شعاعی باشد.

بخش۲: سوالات هماهنگ نشده

پرسش دوم

ابتدا ۱۰۰۰ نمونه داده تصادفی برای دو متغیر x و y در بازه $[-1,1]$ تولید شد. سپس برای هر ورودی ۳ تابع تعلق گوسی تعریف شد و مراکز آن‌ها به‌صورت یکنواخت در این بازه قرار گرفتند$(-1,0,1)$. واریانس پیشنهادی برای هر تابع تعلق با استفاده از رابطه $\frac{d_{max}}{n_{mfs}}$ محاسبه شد، که در آن d_{max} فاصله بیشینه بین مراکز و n_{mfs} تعداد توابع تعلق است. خروجی کد به صورت زیر شد و واریانس پیشنهادی ۰.۶۶۷ محاسبه شد:

```
Membership function parameters:
Centers for x: [-1.  0.  1.]
Proposed variance for x: 0.667
Centers for y: [-1.  0.  1.]
Proposed variance for y: 0.667
```

الف) اول برای خروجی z به‌صورت ترکیبی از توابع سینوسی و کسینوسی تعریف شده است که به مقادیر ورودی x و y وابسته است:

```
z = np.sin(np.pi * x) + np.cos(np.pi * y)
```

حالا داده ها به train و test تقسیم شدند و بعد mlp تعریف شد:

```
mlp = MLPRegressor(hidden_layer_sizes=(10,), activation='relu', solver='adam', max_iter=500, random_state=13)
mlp.fit(X_train, y_train)
```

MSE برای test و train و تعداد پارامترهای آن:

```
Train MSE: 0.0422590482616815 66
Test MSE: 0.04473942517590944
Number of Parameters in MLP: 41
```

در سیستم ANFIS با دو ورودی x و y و ۳ تابع عضویت گوسی برای هر ورودی، تعداد کل پارامترها برابر با ۳۹ است. این تعداد شامل ۱۲ پارامتر برای توابع عضویت (هر تابع گوسی دارای مرکز و واریانس است) و ۲۷ پارامتر خطی برای قوانین فازی است. تعداد قوانین فازی، که برابر با ترکیب توابع عضویت ورودی‌هاست، ۹ قانون است و هر قانون دارای سه پارامتر خطی (p,q,r) است.

پرسش سوم

ابتدا فایل دیتا در گوگل درایو آپلود و توسط کد زیر با دستور gdown در محیط کولب لود و در یک دیتا فریم ذخیره شد:

```
file_id = "1HW_LtzES6R8Jdbn-dkNP7gMKFgtzhgb7"
url = f"https://drive.google.com/uc?id={file_id}"

# File Direction
File_dir = "/content/evaporator.dat"
gdown.download(url, File_dir, quiet=False)
```

ذخیره در دیتافریم با توجه به جدا کننده عددها و تنظیم header=0 چون ردیف اول header نیست:

```
# move to dataframe
df = pd.read_csv(File_dir, sep='\t', header=None, engine='python')

# Convert data to numeric if necessary
df = df.apply(pd.to_numeric, errors='coerce')
df
```

خروجی دیتافریم:

	0	1	2	3	4	5	6
0	1.011246	-0.935022	1.019753	-0.315268	0.182425	-0.062869	NaN
1	1.011246	-0.935022	-0.913381	-0.743985	0.827542	-0.062869	NaN
2	-0.991556	-0.962638	-1.030541	-0.101554	1.096341	0.261300	NaN
3	1.039063	0.984236	1.019753	-0.512577	0.881302	0.131632	NaN
4	0.997338	0.984236	1.027075	-0.281169	0.343704	-0.062869	NaN
...	...	...	...	...	...	...	...
6300	0.993524	-0.982784	1.032686	1.469205	-0.270319	-1.763189	NaN
6301	-0.968710	1.038635	1.025654	1.572471	-0.317648	-1.763189	NaN
6302	0.965492	-0.861499	-1.048885	1.529801	0.486946	-1.717518	NaN
6303	-1.024774	1.011683	-1.069982	1.156953	1.386198	-1.443495	NaN
6304	0.979508	0.998207	1.074880	0.814164	1.149553	-1.352154	NaN

6305 rows × 7 columns

الف) مقادیر NAN در ستون ۶ وجود داشت پس این ستون حذف شد:

```
df_cleaned = df.drop(columns=[6])
```

در مرحله بعد با MinmaxScaler نرمال سازی شدند:

```
# Normalize the data
scaler = MinMaxScaler()
df_normalized = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
```

و در نهایت با استفاده از کتابخانه sklearn به نسبت ۷۰و ۳۰ به دسته های train و test تقسیم شدند:

```
# Split the data into train (70%) and test (30%) sets
train_data, test_data = train_test_split(df_normalized, test_size=0.3, random_state=13)
```

ب) ابتدا سیستم توسط RBF شناسایی شد. برای اینکار اول ستون آخر دیتاست به عنوان تارگت در دسته های train و test جدا شد:

```
# Define input (X) and output (y)
X_train = train_data.iloc[:, :-1].values  # All columns except the last
y_train = train_data.iloc[:, -1].values  # The last column
X_test = test_data.iloc[:, :-1].values
y_test = test_data.iloc[:, -1].values
```

حالا با استفاده الگوریتم K_Means مرکزهای RBF ها را تعیین کرده. با این کار داده های train به ۱۰ خوشه تقسیم می شوند و مرکز این خوشه ها به عنوان مرکز RBF ها انتخاب می شوند.

```
# Determine RBF centers using KMeans
n_centers = 10  # Number of RBF centers
kmeans = KMeans(n_clusters=n_centers, random_state=13)
kmeans.fit(X_train)
centers = kmeans.cluster_centers_
```

سپس تابع فعال سازی RBF مطابق فرمولش تعریف شد:

```
def rbf_activation(X, centers, sigma):
    # Compute the distance between each input and each center
    distances = np.linalg.norm(X[:, np.newaxis] - centers, axis=2)
    # Apply the Gaussian function
    return np.exp(-distances**2 / (2 * sigma**2))
```

سپس پهنای تابع گوسی یا سیگما محاسبه شد:

```
sigma = np.max(np.linalg.norm(centers[:, np.newaxis] - centers, axis=2)) / np.sqrt(n_centers)
```

و داده ها تنظیم به تابع فعالسازی وارد شدند:

```
phi_train = rbf_activation(X_train, centers, sigma)
phi_test = rbf_activation(X_test, centers, sigma)
```

حالا با یک رگرسیون پیش بینی انجام شد:

```
# Predict and evaluate the model
y_pred_train = regressor.predict(phi_train)
y_pred_test = regressor.predict(phi_test)
```

MSE مربوط به train و test به صورت زیر به دست آمدند:

```
Train MSE: 0.0212
Test MSE: 0.0211
```

حالا سیستم توسط ANFIS شناسایی شد. برای اینکار به صورت from scratch این مدل پیاده شد. ابتدا تابع عصویت گاوسی تعریف شد:

```
# Define Gaussian Membership Function
def gaussian_mf(x, c, sigma):
    return np.exp(-((x - c) ** 2) / (2 * sigma ** 2))
```

بعد تابع پیاده سازی توابع عضویت برای تعداد ورودی و تعداد تابغ عضویت. داخل این تابع مرکاز به صورت مساوی پخش می‌شوند. و در خروجی پهنا و مراکز خروجی داده می‌شود.

```
# Initialize Membership Functions
def initialize_mfs(n_inputs, n_mfs):
    centers = []
    sigmas = []
    for _ in range(n_inputs):
        centers.append(np.linspace(0, 1, n_mfs))
        sigmas.append(np.full(n_mfs, (1 / (n_mfs - 1))))
    return np.array(centers), np.array(sigmas)
```

بعد تابع forward pass تعریف شد که بردار ورودی، مراکز و پهنا ها و ضرایب را دریافت می کند و فعال سازی قوانین را تنظیم میکند (به علت کم بود وقت مراقب گفتند نیاز به توضیح کامل نیست) سپس تابعbackward pass نیز تعریف شد. در نهایت با استفاده از پیاده سازی انجام شده یک مدل anfis طراحی شد. این مدل برای هر ورودی سه تابع عضویت از نوع گاوسی دارد و برای ۱۰ epoch آموزش داده شد.

خروجی MSE برای هر epoch و MSE برای دیتای test یه صورت زیر شدند:

```
Epoch 1/10, MSE: 0.0317
Epoch 2/10, MSE: 0.0271
Epoch 3/10, MSE: 0.0252
Epoch 4/10, MSE: 0.0242
Epoch 5/10, MSE: 0.0235
Epoch 6/10, MSE: 0.0230
Epoch 7/10, MSE: 0.0227
Epoch 8/10, MSE: 0.0224
Epoch 9/10, MSE: 0.0222
Epoch 10/10, MSE: 0.0220
Test MSE: 0.0220
```

نکته خیلی قابل توجه وقت بسیار زیادی بود که مدل Anfis برای آموزش نیاز داشت حتی برای ۱۰ تکرار که حتی با این حال در انتها خطای بیشتری داشت. علت این اتفاق افزایش تصاعدی rule ها با افزایش ورودی ها است. که اگر هر ورودی تعداد تابع عضویت یکسان داشته باشید تعداد به صورت تعداد تابع عضویت به توان ورودی می‌شود. ولی مدل RBF به سرعت و دقت بالا مدل را شناسایی کرد. پس باید دقت کرد که چه موقع از مدل ANFIS استفاده می‌شود. مزیت این مدل در دیتا هایی هست که مقدار قطعی ندارند و تعداد ورودی ها کم است ولی با افزایش ورودی ها باید از مدل های دیگر استفاده کرد. پس ANFIS برای سیستم‌هایی با روابط پیچیده و غیرخطی و داده‌های دارای عدم قطعیت مناسب‌تر است، زیرا از منطق فازی برای مدل‌سازی بهره می‌برد و قوانین فازی قابل‌تفسیر دارد. این روش در کاربردهایی مثل تصمیم‌گیری و سیستم‌های کنترلی که شفافیت مدل اهمیت دارد بهتر است. در مقابل، RBF به دلیل ساختار ساده‌تر و استفاده از توابع شعاعی، سرعت بسیار بالاتری در آموزش و پیش‌بینی دارد و برای داده‌هایی با روابط غیرخطی ساده‌تر یا حجم بزرگ‌تر مناسب‌تر است. در مسائل دسته‌بندی و رگرسیون با نیاز به پردازش سریع، RBF انتخاب بهتری است، اما ANFIS در مدیریت پیچیدگی و ابهام داده‌ها عملکرد بهتری دارد.

پرسش چهار

پرسش پنج -۱

برای مدل سازی تابع سینک با استفاده از ANFIS از نرم افزار متلب استفاده شد.

ابتدا نقاط x و y تعریف شدند:

```
% Define the data range
x = -10:0.5:10;
y = -10:0.5:10;
[X, Y] = meshgrid(x, y);
```

بعد مقادیر واقعی تابع sinc محاسبه شدند:

```
% Compute the values of the sinc function
R = sqrt(X.^2 + Y.^2);
Z = sinc(pi * R);
```

سپس دیتا به صورت ستونی در آورده شد:

```
% Convert data to column format for ANFIS
L1 = reshape(X, [row*col, 1]);
L2 = reshape(Y, [row*col, 1]);
L3 = reshape(Z, [row*col, 1]);
data = [L1, L2, L3];
```

حال دیتا به دسته train و test به نسبت ۷۰ به ۳۰ تقسیم شد:

```
% Split the data into training and testing sets
train_ratio = 0.7;
n_train = round(size(data, 1) * train_ratio);
idx = randperm(size(data, 1)); % Random permutation of indices
train_data = data(idx(1:n_train), :);
test_data = data(idx(n_train+1:end), :);
```

حالا به طراحی مدل پرداخته شد. ابتدا به صورت زیر تعداد membership ها برای ورودی ها ۳ در نظر گرفته شد. دقت شد که افزایش این توابع موجب بهبود عملکرد سیستم می شود اما زمان آموزش زیاد می شود چون rule ها هم به صورت تصاعدی زیاد می‌شوند! برای ممبرشیپ فاکشن ها از تابع عضویت گاوسی استفاده شد. و دیتای train به عنوان ورودی داده شد.

```
% Generate the initial fuzzy system
numMFs = 3; % Number of membership functions for each input
input_fis = genfis1(train_data, numMFs, 'gaussmf', 'linear');
```

سپس با دستور anfis مدل آموزش داده شد و تعداد epoch هم ۱۰۰ تا انتخاب شد.

```
% Train the ANFIS model
epoch_n = 100; % Number of training epochs
anfis_model = anfis(train_data, input_fis, epoch_n);
```

حال با دستور evalfis مدل آموزش داده شده با دیتای test بررسی شد و مقدار RMSE محاسبه و چاپ شد.

```
predicted_output = evalfis(test_input, anfis_model);
```

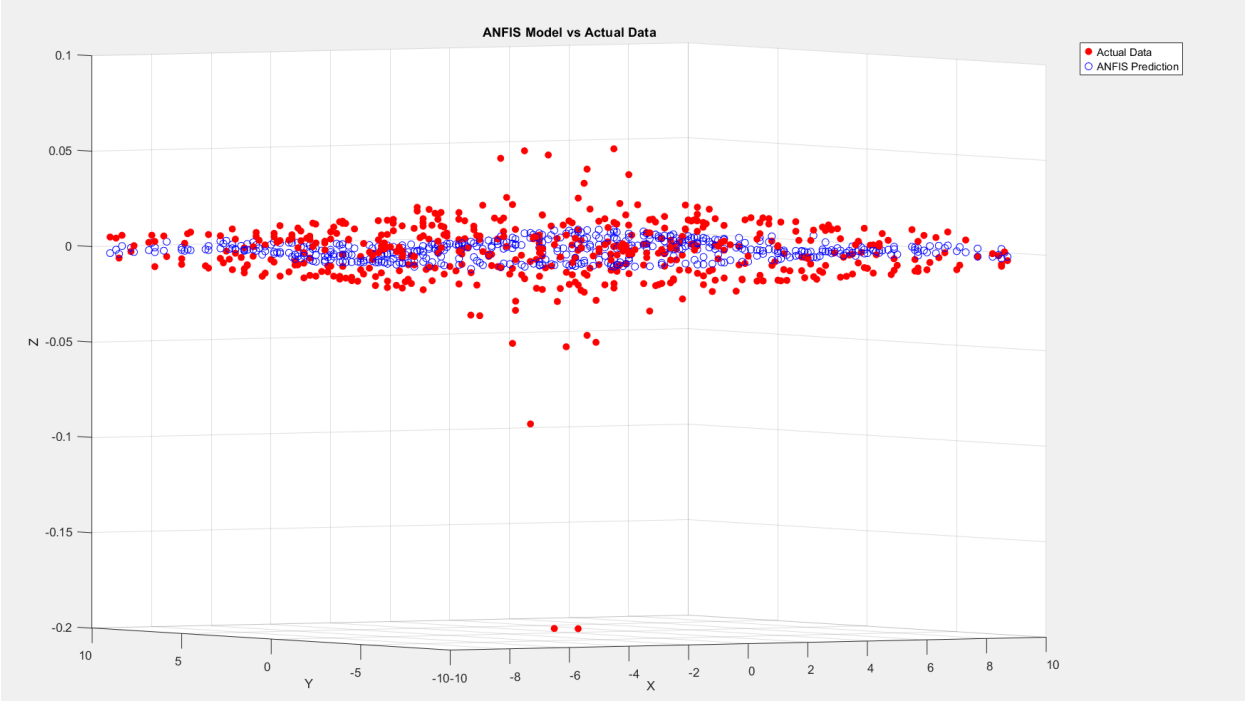
برای محاسبه RMSE مقدار خروجی مدل از مقدار test هدف که ستون سوم بود کم شد و به توان دو رسید و میانگین و رادیکال گرفته شد(مطابق فرمول RMSE)

```
% Compute RMSE
rmse = sqrt(mean((test_target - predicted_output).^2));
disp(['RMSE: ', num2str(rmse)]);
```

مقدار RMSE:

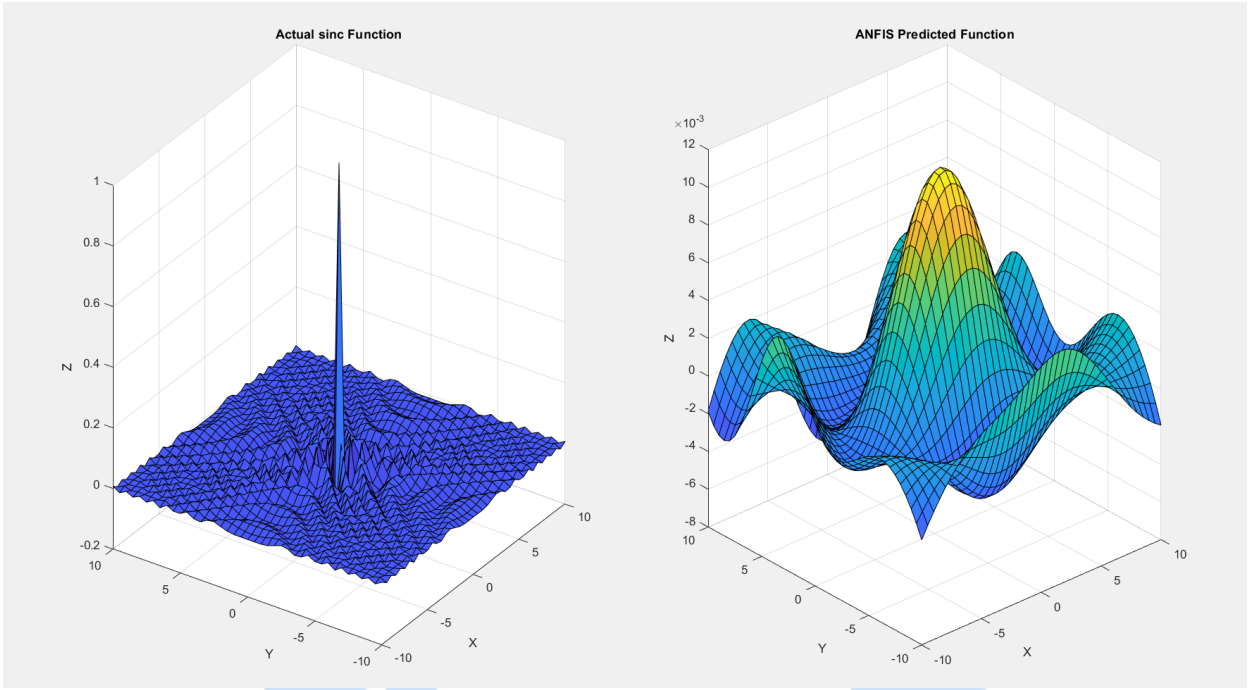
```
RMSE: 0.018516
```

در نهایت به صورت scatter دیتا های test و دیتاهای پیش بینی شده توسط مدل رسم شدند:



همانطور که مشاهده می‌شود، مدل در ورودی های نزدیک صفر و کوچک خیلی خوب عمل نکرده است.

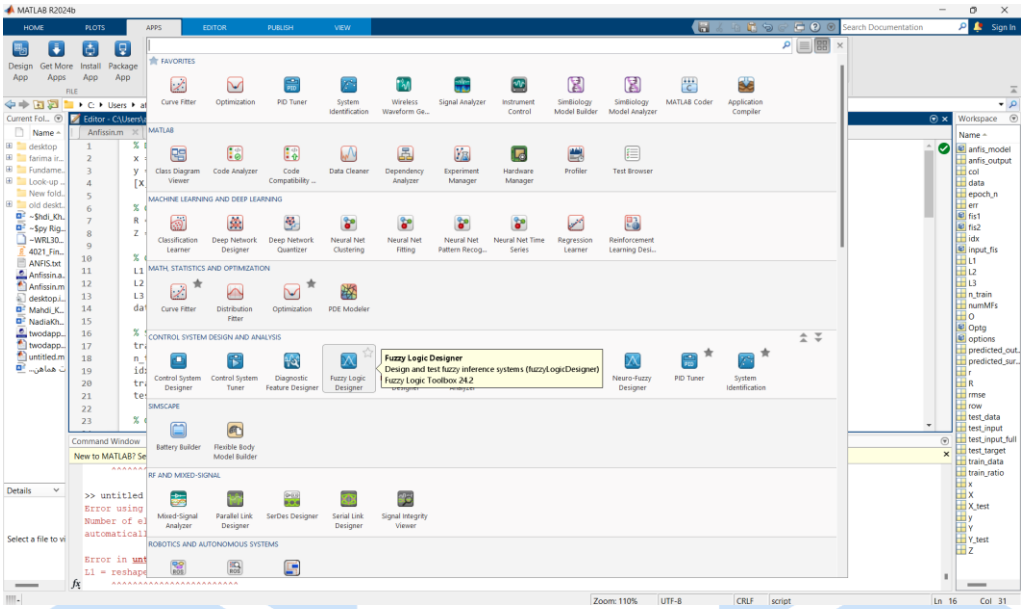
در انتها، خروجی مدل و تابع سینک واقعی در کنار هم رسم شدند:



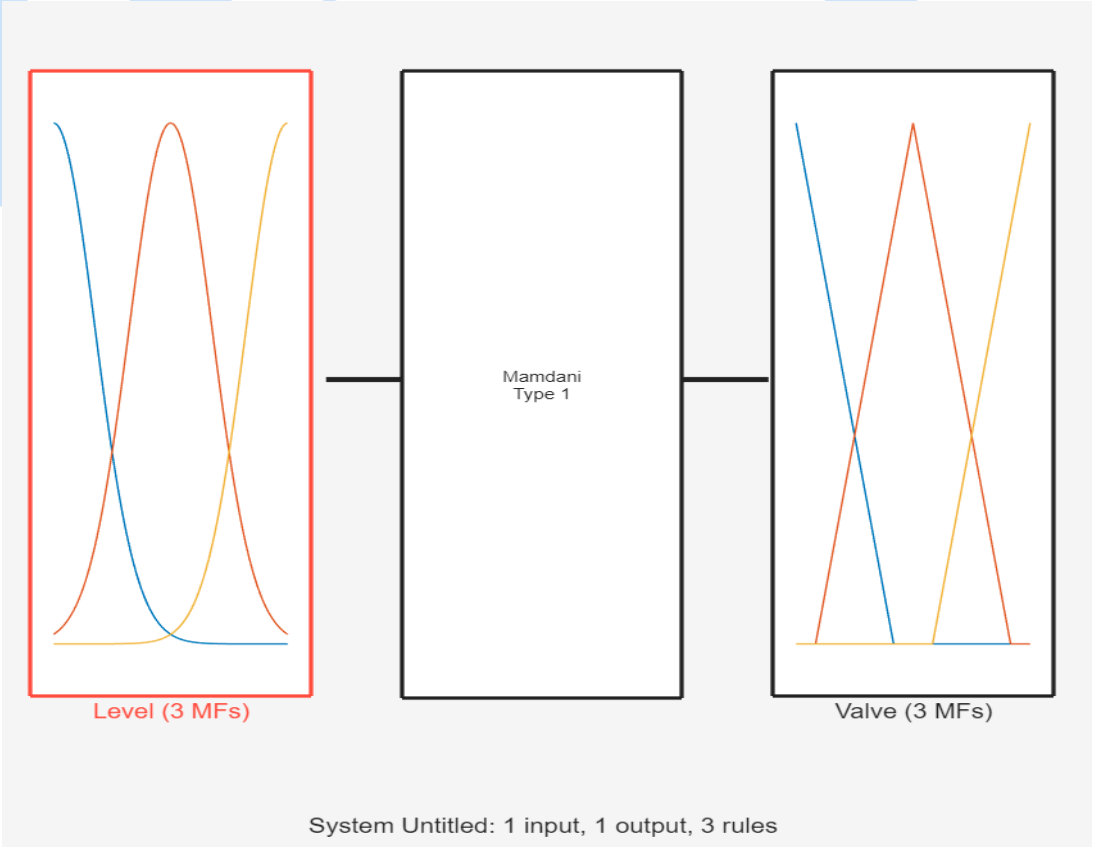
در نقاط نزدیک به صفر، مدل ANFIS عملکرد ضعیفی دارد، زیرا تابع sinc در r=0 دارای مقدار خاصی است که برابر با ۱ می‌باشد، در حالی که در سایر نقاط توسط sin(pi*r)/pi*r تعریف می‌شود. این تغییر ناگهانی و رفتار غیرخطی تابع در نزدیکی صفر، یادگیری مدل را سخت کرده است. علاوه بر این، اگر داده‌های کافی در این ناحیه حساس برای آموزش مدل موجود نباشد، مدل نمی‌تواند این ناحیه را به‌درستی یاد بگیرد. برای بهتر کردن مدل، ابتدا باید تعداد داده‌های آموزشی در نزدیکی r=0 افزایش یابد تا مدل بتواند این ناحیه را بهتر یاد بگیرد. افزایش تعداد توابع عضویت در ANFIS می‌تواند به بهبود توانایی مدل برای یادگیری تغییرات شدید تابع کمک کند.

پرسش پنج -۲

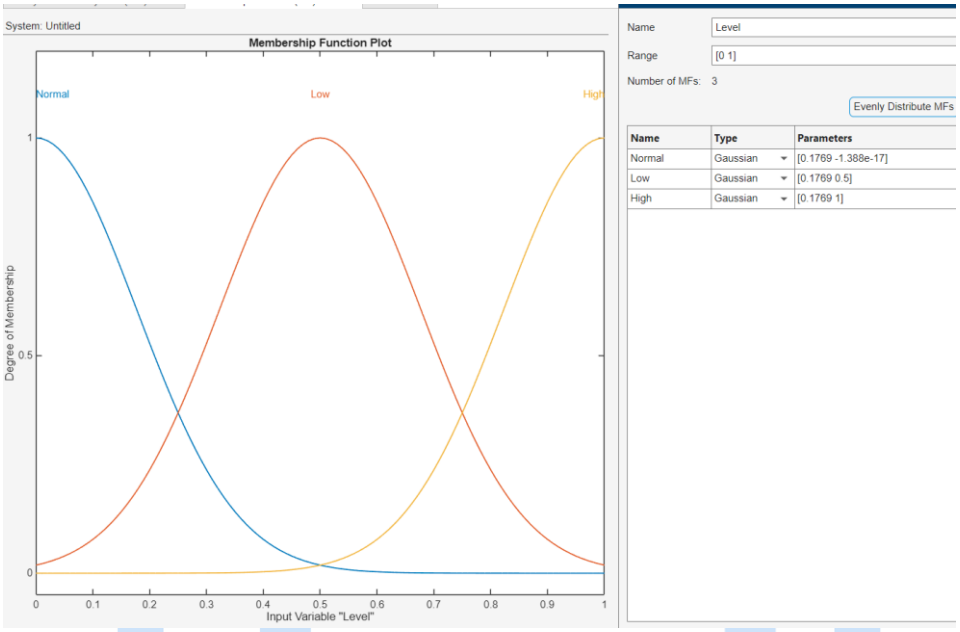
برای این بخش از GUI متلب برای سیستم های فازی استفاده شد.



ابتدا سیستم فازی mamdani ساخته شد و یک خروجی و یک ورودی برای آن قرار داده شد:

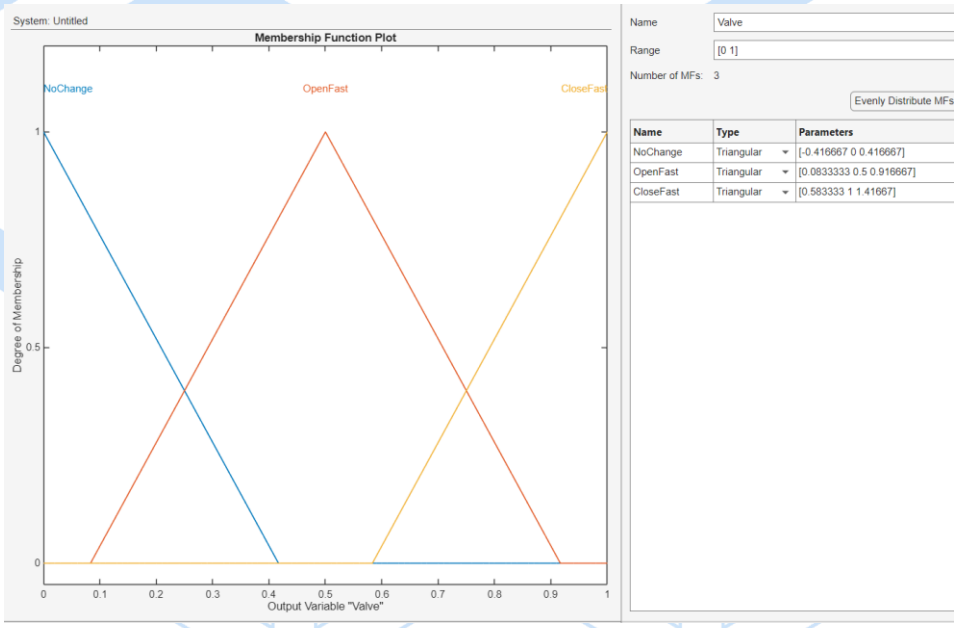


سپس توابع عضویت ورودی تعریف شدند:



سه تابع عضویت گوسی از رنج ۰ تا ۱ که به صورت مساوی پخش شدند.

به طور مشابه توابع عضویت خروجی هم نغریف شدند:



سه تابع مثلثی باز هم از رنج ۰ تا ۱ و پخش شده به صورت مساوی.

سپس قواعد سیستم تنظیم شدند:

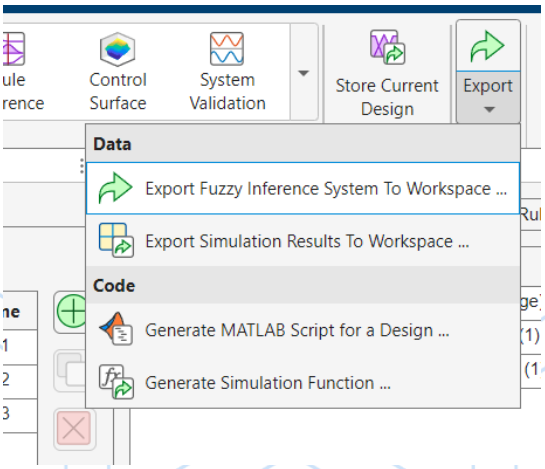
PROPERTY EDITOR: FIS

Type:	Mamdani Type-1
Name	VlaveFIS
And method	min
Or method	max
Implication method	min
Aggregation method	max
Defuzzification method	centroid
Inputs:	1
Outputs:	1
Rules:	3

در انتها نیز rule های سیستم هم تعریف شدند:

	Rule	Weight	Name
1	If Level is Normal then Valve is NoChange	1	rule1
2	If Level is Low then Valve is OpenFast	1	rule2
3	If Level is High then Valve is CloseFast	1	rule3

حال این سیستم به workspace نرم‌افزار منتقل شد:



حال با دستور زیر سیستم فازی برای ورودی ۰.۸ بررسی شد:

```
>> output = evalfis(VlaveFIS, [0.8])

output =

    0.6499
```

که خروجی سیستم ۰.۶۴۹۹ بود.