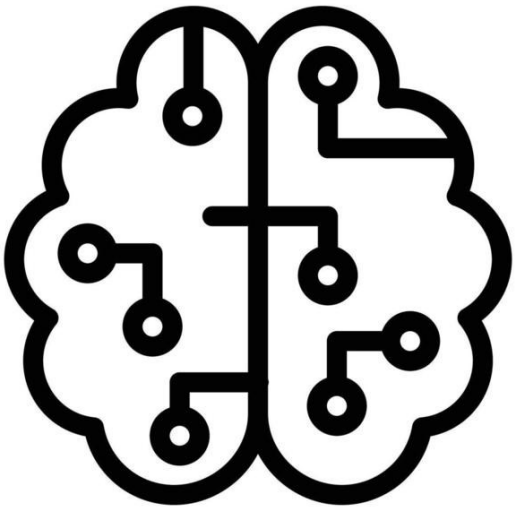


Fuzzy Logic

BREAST CANCER FUZZY SYSTEM

Full 1403

WRITE

Pariya Afsharipour

Reprared for:

Dr. Hosseini

فهرست مطالب

۱. **فصل ۱: تحلیل مسئله**
   - ۱-۱شرح پروژه
   - ۱-۲دیتاست مورد استفاده
   - ۱-۳ تحلیل داده‌های ورودی مسئله
   - ۱-۴سایر منابع
۲. **فصل ۲: انتخاب ویژگی و توابع عضویت**
   - ۲-۱ پیش‌پردازش و انتخاب ویژگی با استفاده ازPCA
   - ۲-۲ انتخاب شکل تابع عضویت
   - ۲-۳ پارامترهای توابع عضویت
   - ۲-۴ کدهای مرتبط با FCM وPCA
۳. **فصل ۳: سیستم هوشمند فازی ممدانی**
   - ۳-۱ سیستم فازی ممدانی
   - ۳-۲ شیوه استفاده از توابع عضویت
   - ۳-۳ فازی‌سازی
   - ۳-۴ قوانین فازی ممدانی
   - ۳-۵ غیرفازی‌سازی
   - ۳-۶ ارزیابی سیستم
   - ۳-۷ بررسی یک نمونه
۴. **فصل ۴: سیستم هوشمند فازی سوگنو**
   - ۴-۱ معرفی سیستم فازی سوگنو
   - ۴-۲ شیوه استفاده از توابع عضویت
   - ۴-۳ پایگاه قوانین
   - ۴-۴ ارزیابی سیستم
۵. **فصل ۵: بهبود عملکرد سوگنو با استفاده از تغییرپذیری**
   - ۵-۱ افزودن یادگیری به سیستم فازی با الگوریتم ژنتیک
   - ۵-۲ اجرای الگوریتم‌های DE برای بهینه‌سازی وزن‌ها
۶. **فصل ۶: استفاده از روش یادگیری میشیگان برای ایجاد پایگاه قوانین**
   - ۶-۱ الگوریتم میشیگان
   - ۶-۲ کدنویسی و ارزیابی عملکرد
۷. **فصل ۷: استفاده از روش یادگیری پیتزبورگ برای ایجاد پایگاه قوانین**
   - ۷-۱ الگوریتم پیتزبورگ
   - ۷-۲ کدنویسی و ارزیابی عملکرد
۸. **فصل ۸: سیستم فازی نوع ۲ و مقایسه با نوع ۱**
   - ۸-۱ معرفی سیستم فازی نوع ۲
   - ۸-۲ کدنویسی سیستم فازی نوع ۲
   - ۸-۳ مقایسه فازی نوع ۱ و نوع ۲

فصل اول:

تحلیل مسئله

۱-۱ شرح پروژه

هدف اصلی این پروژه طراحی یک سیستم فازی برای ارزیابی و برچسب‌گذاری نوع سرطان سینه است. سیستم طراحی‌شده قادر خواهد بود نوع توده‌های سرطانی را به دو دسته خوش‌خیم (Benign) و بدخیم (Malignant) طبقه‌بندی کند. این سیستم با استفاده از قوانین فازی و مجموعه داده‌های معتبر، سعی در ارائه یک ابزار تشخیص دارد که دقت بالایی داشته باشد.

۱-۲ دیتاست مورد استفاده

دیتاست مورد استفاده در این پروژه، Breast Cancer Wisconsin (Diagnostic) است که یکی از مجموعه داده‌های معروف و معتبر در زمینه تشخیص سرطان سینه به شمار می‌رود. این دیتاست شامل اطلاعاتی از توده‌های سرطانی است که به کمک روش‌های کلینیکی جمع‌آوری شده‌اند. ویژگی‌های موجود در این دیتاست به توصیف مشخصات فیزیکی و آماری سلول‌های توده‌ها می‌پردازد.

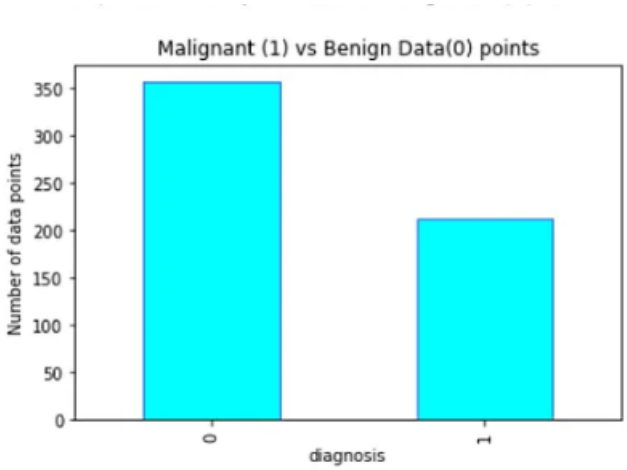
۱-۱-۱ ویژگی‌های دیتاست

دیتاست شامل ۵۶۹ نمونه است و شامل ۳۰ ویژگی عددی می‌شود. این ویژگی‌ها برای توصیف ویژگی‌های سلول‌های سرطانی استفاده می‌شوند. در جدول زیر، بخشی از ویژگی‌های موجود در دیتاست به همراه دامنه مقادیر آنها ارائه شده است:

شماره	ویژگی	توضیحات	حداقل مقدار	حداکثر مقدار
1	Radius Mean	میانگین شعاع سلول	6.981	28.11
2	Texture Mean	میانگین بافت سلول	9.71	39.28
3	Perimeter Mean	میانگین محیط سلول	43.79	188.5
4	Area Mean	میانگین مساحت سلول	143.5	2501
5	Smoothness Mean	میانگین صافی سلول	0.05263	0.1634
6	Compactness Mean	میانگین فشردگی سلول	0.01938	0.3454
7	Concavity Mean	میانگین فرورفتگی سلول	0	0.4268

8	Concave Points Mean	میانگین نقاط فرورفته	0	0.2012
9	Symmetry Mean	میانگین تقارن سلول	0.106	0.304

۱-۳ تحلیل داده های ورودی مسئله



در نمودار بالا می توانیم نسبت داده های ۲ کلاس در دیتاست مشاهده کنیم.

٪۶۷ داده ها برای کلاس خوش خیم و ٪۳۷ داده ها برای کلاس بدخیم است.

۱-۴ سایر منابع

برای پیاده‌سازی این پروژه از محیط Google Colab و زبان برنامه‌نویسی پایتون استفاده شده است.

فصل دوم:

انتخاب ویژگی و توابع عضویت

۲-۱ پیش پردازش و انتخاب ویژگی با استفاده از PCA

در پیش‌پردازش داده‌ها و انتخاب ویژگی‌ها از الگوریتم PCA برای کاهش ابعاد و پیچیدگی داده‌ها استفاده شده است. این الگوریتم به منظور کاهش ابعاد داده‌ها و حفظ بیشترین واریانس ممکن، ویژگی‌هایی را انتخاب کرده است که بیشترین اطلاعات را در خود دارند. ۸ ویژگی منتخب که بیشترین واریانس را دارند، به‌عنوان ورودی‌های مدل انتخاب شده‌اند.

۲-۲ انتخاب شکل تابع عضویت

در میان انواع شکل های تابع عضویت مانند مثلثی، ذوالذنقه، گوسی، زنگوله ای و... تابع گوسی برای این پروژه انتخاب شده است. تابع گوسی از فرمول زیر تبعیت می‌کند، که در آن μ مرکز و σ پهنای تابع را مشخص می‌کنند.

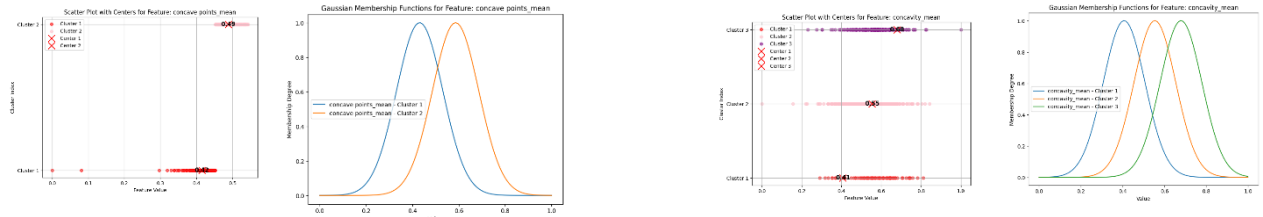
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

۲-۳ پارامتر های توابع عضویت

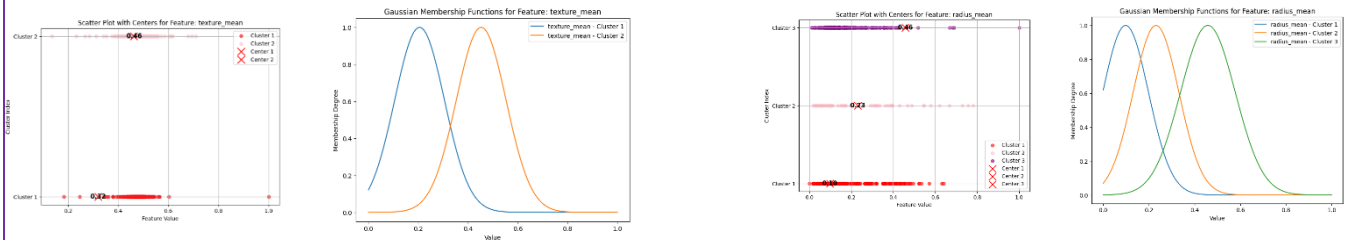
به منظور تنظیم پارامترهای توابع عضویت در این پروژه، از الگوریتم FCM استفاده شده است. این الگوریتم یکی از روش‌های قدرتمند در خوشه‌بندی داده‌ها است و به هر داده یک درجه عضویت نسبت به هر خوشه اختصاص می‌دهد. در این پروژه، الگوریتم FCM داده‌های هر ویژگی را تحلیل می‌کند و با ایجاد خوشه‌های فازی، مقادیر میانگین و پراکندگی داده‌ها در هر خوشه را مشخص می‌کند. این مقادیر به عنوان پارامترهای توابع گوسی برای تعریف مرزهای عضویت به کار می‌روند.

با استفاده از FCM هر ورودی به ۲ تا ۳ خوشه تقسیم شده است.

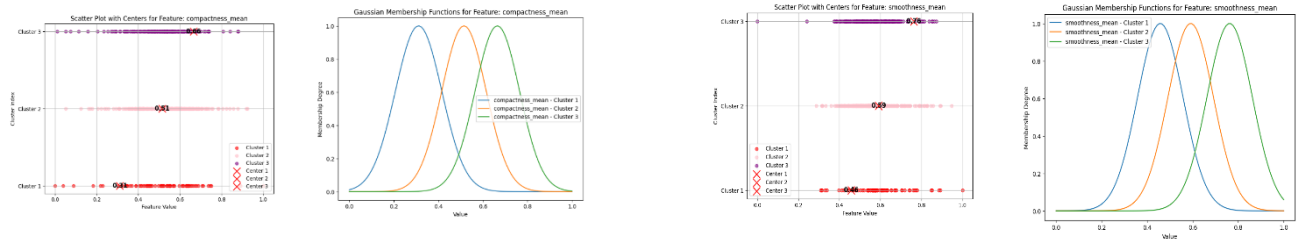
در ادامه پلات های نتیجه این الگوریتم با استفاده از کتابخانه matplotlib رسم شده اند:



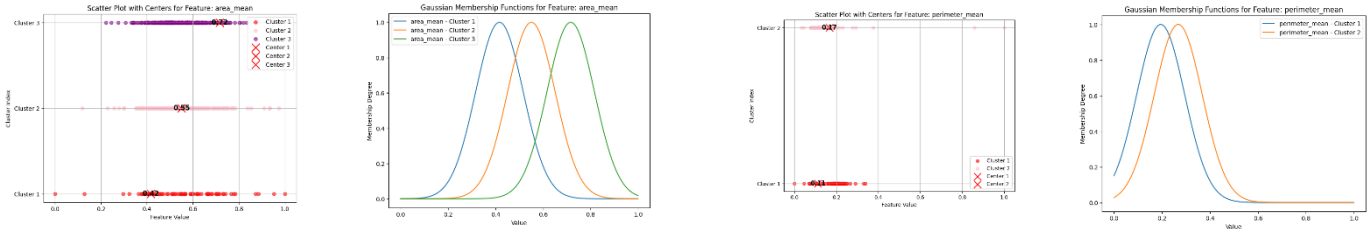
نمودار گوسی و مرکز خوشه برای میانگین فرورفتگی وتعداد فرورفتگی



نمودار گوسی و مرکز خوشه برای ..... و.....



نمودار گوسی و مرکز خوشه برای میانگین فشردگی ومیانگین نرمی



نمودار گوسی و خوشه برای میانگین مساحت ومیانگین محیطی

همچنین جدول زیر بازه هر کدام از عبارت زبانی مربوط به هر ویژگی را نمایش می دهد:

ویژگی	Cluster	**مقدار بازه**	عبارت فازی	توضیحات

radius_mean	1	$0.00 - 0.17$	کوچک	اندازه کوچک توده
	2	$0.17 - 0.34$	متوسط	اندازه متوسط توده
	3	$0.35 - 1.00$	بزرگ	اندازه بزرگ توده
texture_mean	1	$0.00 - 0.29$	نرم	بافت نرم توده
	2	$0.42 - 0.47$	سخت	بافت سخت توده
perimeter_mean	1	$0.23 - 0.40$	کوچک	محیط کوچک‌تر
	2	$0.18 - 0.22$	متوسط	محیط بزرگ تر
area_mean	1	$0.00 - 0.48$	کوچک	مساحت کوچک توده
	۲	$0.49 - 0.63$	متوسط	مساحت متوسط توده
	۳	$0.63 - 1.00$	بزرگ	مساحت بزرگ توده
smoothness_mean	1	$0.00 - 0.52$	نرم	توده نرم
	۲	$0.53 - 0.68$	متوسط	توده متوسط
	۳	$0.68 - 1.00$	سخت	توده سخت
compactness_mean	1	$0.00 - 0.42$	کم ترام	تراکم کم توده
	۲	$0.53 - 0.68$	متراکم	تراکم متوسط توده
	۳	$0.68 - 1.00$	پرتراکم	تراکم زیاد توده (بدخیم)
concavity_mean	1	$0.00 - 0.48$	منظم	میزان فرورفتگی کم
	۲	$0.48 - 0.61$	شبه منظم	میزان فرورفتگی متوسط
	۳	$0.62 - 1.00$	نامنظم	میزان فرورفتگی زیاد
concave points_mean	1	$0.00 - 0.39$	کم	تعداد نقاط فرورفتگی کم
	2	$0.62 - 0.70$	زیاد	تعداد نقاط فرورفتگی زیاد

اطلاعات به دست آمده از اجرای الگوریتم FCM داخل فایل جیسون ذخیره شد تا در ادامه برای طراحی سیستم مورد استفاده قرار گیرد.

۲-۴ کد

در ادامه به به توضیح کد این بخش می پردازیم:

۲-۴-۱ کتابخانه های مورد نیاز

برای استفاده از الگوریتم FCM نیاز به نصب کتابخانه scikit-fuzzy داشتم و همچنین برای دسترسی به دیتاست breast cancer winscontin از طریق سایت kaggle به شکل زیر عمل کرده ام.

```
!pip install scikit-fuzzy
!pip install kaggle
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d uciml/breast-cancer-wisconsin-data
!unzip breast-cancer-wisconsin-data.zip

from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
import pandas as pd
import numpy as np
import skfuzzy as fuzz
```

۲-۴-۲ استفاده از الگوریتم pca و fcm

پس از انتخاب ویژگی‌های ورودی و کاهش ابعاد با استفاده ازPCA ، داده‌ها را نرمال‌سازی می شوند. سپس، الگوریتم FCM روی هر ویژگی اعمال می‌شود تا داده‌ها به چند خوشه تقسیم شوند. الگوریتم FCMمراکز و توابع عضویت خوشه‌ها را محاسبه می‌کند. برای هر خوشه، میانگین و انحراف معیار (به عنوان پارامترهای تابع گوسی) استخراج و ذخیره می‌شود.

```
data = pd.read_csv("data.csv")

X = data.iloc[:, 2:-1].values
```

```
feature_names = data.columns[2:-1]

pca = PCA(n_components=8)
X_pca = pca.fit_transform(X)

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X_pca)

fcm_params = {}

for feature_idx in range(X_scaled.shape[1]):
    x_vals = X_scaled[:, feature_idx].reshape(1, -1)
    best_clusters = 3
    if feature_names[feature_idx] in ["texture_mean", "perimeter_mean", "concave
points_mean"]:
        best_clusters = 2

    cntr, u, _, _, _, _, _ = fuzz.cluster.cmeans(
        x_vals, c=best_clusters, m=2, error=0.005, maxiter=1000, init=None
    )

    feature_gaussians = []
    for i in range(best_clusters):
        cluster_data = x_vals[0, np.argmax(u, axis=0) == i]

        if len(cluster_data) > 1:
            mean = np.mean(cluster_data)
            sigma = max(np.std(cluster_data), 0.1)
        else:
            mean = np.mean(cluster_data)
            sigma = 0.1

        feature_gaussians.append((mean, sigma))

    fcm_params[feature_names[feature_idx]] = feature_gaussians
```

فصل سوم

سیستم هوشمند فازی ممدانی

۳-۱ سیستم فازی ممدانی

سیستم فازی ممدانی که بر اساس قوانین "اگر-آنگاه" عمل می‌کند. در این سیستم، مقادیر ورودی به مجموعه‌های فازی تبدیل می‌شوند، سپس با استفاده از قوانین فازی خروجی فازی تولید می‌شود و در نهایت به یک مقدار دقیق تبدیل می‌گردد.

۳-۲ شیوه استفاده از توابع عضویت

برای طراحی این سیستم ممدانی و ۲ فصل بعد از نتیجه الگوریتم FCM و PCA فصل قبل استفاده شده است. به منظور استفاده از نتایج با کد زیر فایل json در گوگل کولب لود می شود:

```
with open("fcm_results.json", "r") as json_file:
    fcm_params = json.load(json_file)
```

۳-۳ فازی سازی

این مرحله، مقادیر ورودی به کمک توابع عضویت به مجموعه‌های فازی نگاشت می‌شوند. این کار باعث می‌شود داده‌های واقعی به زبان فازی تبدیل شوند.

در fuzzify_data_point برای هر ویژگی، یک تابع فازی تعریف شده که بر اساس بازه‌های مشخص، درجه عضویت مقادیر را در دسته‌های مختلف (مثل کوچک، متوسط یا بزرگ) محاسبه می‌کند. در نهایت، تابع convert_fuzzy_to_categories برای هر ویژگی، دسته‌ای که بیشترین درجه عضویت را دارد به عنوان دسته نهایی انتخاب می‌کند:

```
def fuzzify_radius_mean(value):
    if value < 12:
        return {"small": 1.0, "medium": 0.0, "large": 0.0}
    elif 12 <= value <= 18:
        return {"small": (18 - value) / 6, "medium": (value - 12) / 6, "large":
0.0}
    elif 18 < value <= 25:
        return {"small": 0.0, "medium": (25 - value) / 7, "large": (value - 18) /
7}
    else:
        return {"small": 0.0, "medium": 0.0, "large": 1.0}
```

```
def fuzzify_texture_mean(value):
    if value < 10:
        return {"soft": 1.0, "medium": 0.0, "hard": 0.0}
    elif 10 <= value <= 20:
        return {"soft": (20 - value) / 10, "medium": (value - 10) / 10, "hard":
0.0}
    elif 20 < value <= 30:
        return {"soft": 0.0, "medium": (30 - value) / 10, "hard": (value - 20) /
10}
    else:
        return {"soft": 0.0, "medium": 0.0, "hard": 1.0}

def fuzzify_perimeter_mean(value):
    if value < 50:
        return {"small": 1.0, "medium": 0.0, "large": 0.0}
    elif 50 <= value <= 100:
        return {"small": (100 - value) / 50, "medium": (value - 50) / 50, "large":
0.0}
    elif 100 < value <= 150:
        return {"small": 0.0, "medium": (150 - value) / 50, "large": (value - 100)
/ 50}
    else:
        return {"small": 0.0, "medium": 0.0, "large": 1.0}

def fuzzify_area_mean(value):
    if value < 400:
        return {"small": 1.0, "medium": 0.0, "large": 0.0}
    elif 400 <= value <= 700:
        return {"small": (700 - value) / 300, "medium": (value - 400) / 300,
"large": 0.0}
    elif 700 < value <= 1000:
        return {"small": 0.0, "medium": (1000 - value) / 300, "large": (value -
700) / 300}
    else:
        return {"small": 0.0, "medium": 0.0, "large": 1.0}

def fuzzify_smoothness_mean(value):
    if value < 0.1:
        return {"soft": 1.0, "medium": 0.0, "hard": 0.0}
    elif 0.1 <= value <= 0.2:
        return {"soft": (0.2 - value) / 0.1, "medium": (value - 0.1) / 0.1, "hard":
0.0}
    elif 0.2 < value <= 0.3:
        return {"soft": 0.0, "medium": (0.3 - value) / 0.1, "hard": (value - 0.2) /
0.1}
    else:
        return {"soft": 0.0, "medium": 0.0, "hard": 1.0}
```

```
def fuzzify_compactness_mean(value):
    if value < 0.05:
        return {"low": 1.0, "medium": 0.0, "high": 0.0}
    elif 0.05 <= value <= 0.15:
        return {"low": (0.15 - value) / 0.1, "medium": (value - 0.05) / 0.1,
"high": 0.0}
    elif 0.15 < value <= 0.3:
        return {"low": 0.0, "medium": (0.3 - value) / 0.15, "high": (value - 0.15)
/ 0.15}
    else:
        return {"low": 0.0, "medium": 0.0, "high": 1.0}

def fuzzify_concavity_mean(value):
    if value < 0.1:
        return {"low": 1.0, "medium": 0.0, "high": 0.0}
    elif 0.1 <= value <= 0.2:
        return {"low": (0.2 - value) / 0.1, "medium": (value - 0.1) / 0.1, "high":
0.0}
    elif 0.2 < value <= 0.4:
        return {"low": 0.0, "medium": (0.4 - value) / 0.2, "high": (value - 0.2) /
0.2}
    else:
        return {"low": 0.0, "medium": 0.0, "high": 1.0}

def fuzzify_concave_points_mean(value):
    if value < 0.05:
        return {"low": 1.0, "medium": 0.0, "high": 0.0}
    elif 0.05 <= value <= 0.15:
        return {"low": (0.15 - value) / 0.1, "medium": (value - 0.05) / 0.1,
"high": 0.0}
    elif 0.15 < value <= 0.25:
        return {"low": 0.0, "medium": (0.25 - value) / 0.1, "high": (value - 0.15)
/ 0.1}
    else:
        return {"low": 0.0, "medium": 0.0, "high": 1.0}


def fuzzify_data_point(crisp_data_point):
    return {
        "radius_mean": fuzzify_radius_mean(crisp_data_point["radius_mean"]),
        "texture_mean": fuzzify_texture_mean(crisp_data_point["texture_mean"]),
        "perimeter_mean":
fuzzify_perimeter_mean(crisp_data_point["perimeter_mean"]),
        "area_mean": fuzzify_area_mean(crisp_data_point["area_mean"]),
        "smoothness_mean":
fuzzify_smoothness_mean(crisp_data_point["smoothness_mean"]),
        "compactness_mean":
fuzzify_compactness_mean(crisp_data_point["compactness_mean"]),
```

```
        "concavity_mean":
fuzzify_concavity_mean(crisp_data_point["concavity_mean"]),
        "concave_points_mean":
fuzzify_concave_points_mean(crisp_data_point["concave_points_mean"]),
    }



def convert_fuzzy_to_categories(fuzzy_data):
    result = {}
    for feature, memberships in fuzzy_data.items():
        max_category = max(memberships, key=memberships.get)
        result[feature] = max_category
    return result
```

۳-۴ قوانین فازی ممدانی

مجموعه‌ای از قوانین "اگر-آنگاه" هستند که روابط بین ورودی‌ها و خروجی‌ها را تعریف می‌کنند

۴۰ قانون با استفاده از عبارت های فازی متناظر با متغیر های فازی این سیستم نوشته شده است.

۴ نمونه از این قوانین در کد زیر قرار دارد:

```
def apply_fuzzy_rules(data_point):
    results = []
    if (data_point['radius_mean'] == "small" and
        data_point['texture_mean'] == "medium" and
        data_point['smoothness_mean'] == "soft"):
        results.append("Benign")

    if (data_point['radius_mean'] == "medium" and
        data_point['texture_mean'] == "soft" and
        data_point['compactness_mean'] == "low"):
        results.append("Relatively Benign")

    if (data_point['radius_mean'] == "large" and
        data_point['texture_mean'] == "soft" and
        data_point['compactness_mean'] == "high"):
        results.append("Malignant")

    if (data_point['area_mean'] == "large" and
        data_point['smoothness_mean'] == "hard" and
        data_point['perimeter_mean'] == "small"):
        results.append("Malignant")
```

۳-۵ غیرفازی سازی

در این مرحله، خروجی‌های فازی به یک مقدار دقیق تبدیل می‌شوند. این مقدار نشان‌دهنده نتیجه نهایی سیستم فازی است.

در این مسئله از میانگین وزنی به منظور غیرفازی استفاده شده است

```
def defuzzify(results, fuzzy_map):
    result_counts = Counter(results)

    weighted_sum = sum(fuzzy_map[result] * count for result, count in
result_counts.items())

    total_weight = sum(result_counts.values())

    return weighted_sum / total_weight if total_weight > 0 else 0


fuzzy_map = {
    'Malignant': 90,
    'Relatively Malignant' : 70,
    'Relatively Benign': 30,
    'Benign': 20
}
```

۳-۶ ارزیابی سیستم

عملکرد سیستم ممدانی با داده های دیتاست مورد ارزیابی قرار گرفته اند:

```
from sklearn.metrics import mean_squared_error, roc_auc_score, roc_curve

data = pd.read_csv("data.csv")
data.rename(columns={'concave points_mean': 'concave_points_mean'}, inplace=True)

features = ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
'smoothness_mean',
            'compactness_mean', 'concavity_mean', 'concave_points_mean']
X = data[features]
y = data['diagnosis'].map({'B': 0, 'M': 1

def apply_fuzzy_system(row):
    row = dict(row)
```

```
    fuzzy_data_point = fuzzify_data_point(row)
    data_point = convert_fuzzy_to_categories(fuzzy_data_point)
    classification_results = apply_fuzzy_rules(data_point)
    fuzzy_output = defuzzify(classification_results, fuzzy_map)
    return 0 if fuzzy_output < 50 else 1


predictions = X.apply(apply_fuzzy_system, axis=1)

mse = mean_squared_error(y, predictions)
print(f"Mean Squared Error (MSE): {mse:.4f}")
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

auc = roc_auc_score(y, predictions)
print(f"Area Under the Curve (AUC): {auc:.4f}")
```

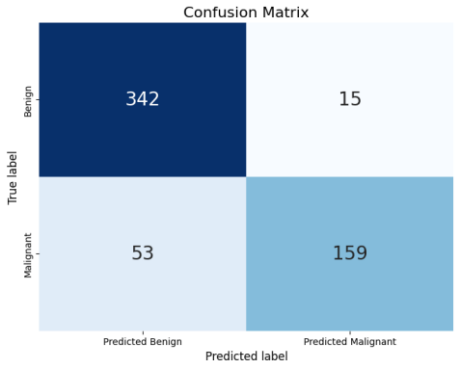
نتایج به دست آمده برای سیستم ممدانی تشخیص سرطان سینه:

Mean Squared Error (MSE): 0.1195

Root Mean Squared Error (RMSE): 0.3457

Area Under the Curve (AUC): 0.8540

همچنین ماتریس سردرگمی این سیستم به شکل زیر می باشد:



ماتریس سردرگمی

۳-۸ بررسی یک نمونه

```
crisp_data_point = {
    "radius_mean": 17.99,
    "texture_mean": 15.23,
    "perimeter_mean": 80.23,
    "area_mean": 500.0,
    "smoothness_mean": 0.18,
```

```
    "compactness_mean": 0.12,
    "concavity_mean": 0.2,
    "concave_points_mean": 0.2,
}

fuzzy_data_point = fuzzify_data_point(crisp_data_point)
data_point = convert_fuzzy_to_categories(fuzzy_data_point)
classification_results = apply_fuzzy_rules(data_point)
final_output = defuzzify(classification_results, fuzzy_map)
```

مقدار نهایی برابر با ۷۲ بود که کلاس را به درستی پیش بینی کرد.

فصل ۴

سیستم هوشمند فازی سوگنو

۴-۱ سیستم فازی سوگنو

سیستم فازی سوگنو یک نوع سیستم فازی است که در آن قوانین فازی به صورت توابع خطی یا غیرخطی برای محاسبه خروجی استفاده می‌شود. برخلاف سیستم فازی ممدانی که در آن خروجی‌ها به صورت مجموعه‌های فازی تولید می‌شوند و سپس با روش‌هایی مانند centroidبه یک مقدار دقیق تبدیل می‌شوند، در سیستم فازی سوگنو، خروجی‌ها مستقیماً از ترکیب توابع ریاضی به‌دست می‌آیند.

۴-۲ شیوه استفاده از توابع عضویت و فازی سازی

این بخش مانند بخش ۳-۲ و ۳-۳ در فصل می باشد. به دلیل تکرای بودن اورده نشده است.

۴-۳ پایگاه قوانین

قوانین فازی سوگنو، نوع دیگری از قوانین فازی هستند که در آنها قسمت "آنگاه" قانون به جای یک عبارت فازی، یک تابع ریاضی (معمولاً خطی) است. این بدان معناست که خروجی هر قانون یک مقدار عددی دقیق است.

```
def fuzzy_sogeno_rules(crisp_data_point, fuzzy_data_point):
    rules = []
    weights = []

    if fuzzy_data_point['radius_mean']['small'] > 0:
        weights.append(fuzzy_data_point['radius_mean']['small'])
        rules.append(2)

    if fuzzy_data_point['radius_mean']['medium'] > 0:
        weights.append(fuzzy_data_point['radius_mean']['medium'])
        rules.append(4 * crisp_data_point['radius_mean'])

    if fuzzy_data_point['radius_mean']['large'] > 0:
        weights.append(fuzzy_data_point['radius_mean']['large'])
        rules.append(8 * crisp_data_point['radius_mean'])

    if fuzzy_data_point['texture_mean']['soft'] > 0:
        weights.append(fuzzy_data_point['texture_mean']['soft'])
```

```
        rules.append(3)

    if fuzzy_data_point['texture_mean']['medium'] > 0:
        weights.append(fuzzy_data_point['texture_mean']['medium'])
        rules.append(5 * crisp_data_point['texture_mean'])

    if fuzzy_data_point['texture_mean']['hard'] > 0:
        weights.append(fuzzy_data_point['texture_mean']['hard'])
        rules.append(7 * crisp_data_point['texture_mean'])

    if fuzzy_data_point['perimeter_mean']['small'] > 0:
        weights.append(fuzzy_data_point['perimeter_mean']['small'])
        rules.append(2)

    if fuzzy_data_point['perimeter_mean']['medium'] > 0:
        weights.append(fuzzy_data_point['perimeter_mean']['medium'])
        rules.append(6 * crisp_data_point['perimeter_mean'])

    if fuzzy_data_point['perimeter_mean']['large'] > 0:
        weights.append(fuzzy_data_point['perimeter_mean']['large'])
        rules.append(10 * crisp_data_point['perimeter_mean'])

    if fuzzy_data_point['area_mean']['medium'] > 0:
        weights.append(fuzzy_data_point['area_mean']['medium'])
        rules.append(4 * crisp_data_point['area_mean'])

    if fuzzy_data_point['area_mean']['large'] > 0:
        weights.append(fuzzy_data_point['area_mean']['large'])
        rules.append(9 * crisp_data_point['area_mean'])

    if fuzzy_data_point['smoothness_mean']['soft'] > 0:
        weights.append(fuzzy_data_point['smoothness_mean']['soft'])
        rules.append(1.5)

    if fuzzy_data_point['smoothness_mean']['medium'] > 0:
        weights.append(fuzzy_data_point['smoothness_mean']['medium'])
        rules.append(3 * crisp_data_point['smoothness_mean'])

    if fuzzy_data_point['smoothness_mean']['hard'] > 0:
        weights.append(fuzzy_data_point['smoothness_mean']['hard'])
        rules.append(5 * crisp_data_point['smoothness_mean'])

    if fuzzy_data_point['compactness_mean']['low'] > 0:
        weights.append(fuzzy_data_point['compactness_mean']['low'])
        rules.append(1)

    if fuzzy_data_point['compactness_mean']['medium'] > 0:
        weights.append(fuzzy_data_point['compactness_mean']['medium'])
```

```
        rules.append(3 * crisp_data_point['compactness_mean'])

    if fuzzy_data_point['compactness_mean']['high'] > 0:
        weights.append(fuzzy_data_point['compactness_mean']['high'])
        rules.append(6 * crisp_data_point['compactness_mean'])

    if fuzzy_data_point['concavity_mean']['low'] > 0:
        weights.append(fuzzy_data_point['concavity_mean']['low'])
        rules.append(1.5)

    if fuzzy_data_point['concavity_mean']['medium'] > 0:
        weights.append(fuzzy_data_point['concavity_mean']['medium'])
        rules.append(4 * crisp_data_point['concavity_mean'])

    if fuzzy_data_point['concavity_mean']['high'] > 0:
        weights.append(fuzzy_data_point['concavity_mean']['high'])
        rules.append(8 * crisp_data_point['concavity_mean'])

    if fuzzy_data_point['concave_points_mean']['low'] > 0:
        weights.append(fuzzy_data_point['concave_points_mean']['low'])
        rules.append(2)

    if fuzzy_data_point['concave_points_mean']['high'] > 0:
        weights.append(fuzzy_data_point['concave_points_mean']['high'])
        rules.append(6 * crisp_data_point['concave_points_mean'])

    if sum(weights) == 0:
        return 0.5
    resistance = sum(w * r for w, r in zip(weights, rules)) / sum(weights)
    return resistance
```

نکته : در سوگنو بخشی به عنوان defuzzyfication نداریم.

۴-۵ ارزیابی سیستم

```
data = pd.read_csv("data.csv")
data.rename(columns={'concave points_mean': 'concave_points_mean'}, inplace=True)

features = [
    'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
    'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave_points_mean'
]
X = data[features]
y = data['diagnosis'].map({'B': 0, 'M': 1})

def apply_fuzzy_system(row):
    crisp_data_point = row
```

```
    fuzzy_data_point = fuzzify_data_point(crisp_data_point)
    classification_result = fuzzy_sogeno_rules(crisp_data_point, fuzzy_data_point)
    return 0 if classification_result <50 else 1
X = data[features]
y = data['diagnosis'].map({'B': 0, 'M': 1})

predictions = X.apply(apply_fuzzy_system, axis=1)

accuracy = accuracy_score(y, predictions)
print(f"Accuracy of the Fuzzy System: {accuracy * 100:.2f}%")

error = (predictions != y).mean()
print(f"Mean Absolute Error (MAE): {error:.4f}")

accuracy = accuracy_score(y, predictions)
print(f"Accuracy of the Fuzzy System: {accuracy * 100:.2f}%")

mse = mean_squared_error(y, predictions)
print(f"Mean Squared Error (MSE): {mse:.4f}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

auc = roc_auc_score(y, predictions)
print(f"Area Under the Curve (AUC): {auc:.4f}")
```

نتایج به دست آمده به شکل زیر است:

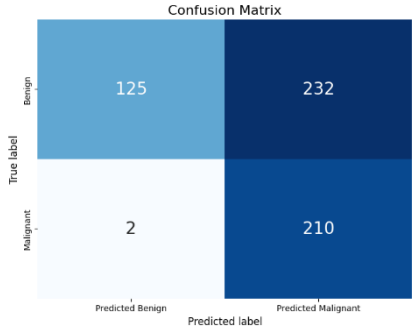
Accuracy of the Fuzzy System: 58.88%

Mean Squared Error (MSE): 0.4112

Root Mean Squared Error (RMSE): 0.6413

Area Under the Curve (AUC): 0.6704

همچنین نتیجه ماتریس سردرگمی به شکل این است:



ماتریس سردرگمی

همانطور که مشخص است دقت سیستم ممدانی بهتر است.

فصل ۵

بهبود عملکرد سوگنو با استفاده تغییرپذیری

۵-۱ افزودن یادگیری به سیستم فازی با استفاده از الگوریتم ژنتیک

سیستم‌های فازی مانند سوگنو به طور ذاتی فاز یادگیری ندارند، زیرا قوانین و پارامترهای آن‌ها معمولاً به صورت دستی و بر اساس دانش کارشناسی تعیین می‌شوند. با این حال، می‌توان از الگوریتم‌های بهینه‌سازی مانند الگوریتم ژنتیک برای تنظیم دقیق (tuning) ضرایب و وزن‌های قوانین استفاده کرد.

۵-۲ کد

با استفاده از قابلیت tuning process برای هر قانون وزنی در نظر گرفته شده تا میزان مشارکت هر قانون در میانگین وزنی متفاوت باشد. با اینکار قوانین قوی تر و محکم تر دارای وزن های بیشتری هستند و ان هایی که ضعیف تر هستند با داده های کمتری سازگاری دارند حذف می شوند.

۵-۲-۱ تغییر ساختار قوانین سوگنو

به منظور اعمال وزن ساختار ۲۳ قانون نوشته شده سوگنو به شکل زیر تغییر دادم. ۳ مورد برای مثال در زیر آورده شده است:

```
def fuzzy_sogeno_rules_with_weights(crisp_data_point, fuzzy_data_point, weights):
    rules = []
    rule_weights = []
    idx = 0

    if fuzzy_data_point['radius_mean']['small'] > 0:
        rule_weights.append(weights[idx])
        rules.append(2 * weights[idx])
        idx += 1
    if fuzzy_data_point['radius_mean']['medium'] > 0:
        rule_weights.append(weights[idx])
        rules.append(4 * crisp_data_point['radius_mean'] * weights[idx])
        idx += 1
    if fuzzy_data_point['radius_mean']['large'] > 0:
        rule_weights.append(weights[idx])
        rules.append(8 * crisp_data_point['radius_mean'] * weights[idx])
        idx += 1
```

۵-۲-۲ تعریف تابع fitness مورد نیاز در الگوریتم de:

تابع fitness در الگوریتم تکامل تفاضلی برای ارزیابی کیفیت کروموزوم‌ها طراحی شده است. این تابع دقت سیستم فازی سوگنو را با استفاده از وزن‌های پیشنهادی هر کروموزوم می‌سنجد. ابتدا داده‌های ورودی فازی‌سازی می‌شوند و خروجی سیستم با وزن‌های کروموزوم محاسبه می‌شود. سپس با مقایسه خروجی با برچسب‌های واقعی، دقت محاسبه شده و یک نویز کوچک برای ایجاد تنوع به آن افزوده می‌شود. این دقت نشان‌دهنده میزان عملکرد هر کروموزوم در بهینه‌سازی وزن‌های سیستم است.

```
def fitness(chromosome, data, labels):
    def apply_fuzzy_system_with_weights(row, weights):
        crisp_data_point = row
        fuzzy_data_point = fuzzify_data_point(crisp_data_point)
        return fuzzy_sogeno_rules_with_weights(crisp_data_point, fuzzy_data_point,
weights)

    predictions = data.apply(lambda row: apply_fuzzy_system_with_weights(row,
chromosome), axis=1)
    predictions_binary = predictions.apply(lambda x: 1 if x >= 50 else 0)
    accuracy = accuracy_score(labels, predictions_binary)
    return accuracy + np.random.uniform(0, 0.01)
```

۵-۲-۳ پیاده سازی الگوریتم de :

تابع differential_evolution برای بهینه‌سازی وزن‌های سیستم فازی با استفاده از الگوریتم تکامل تفاضلی طراحی شده است. ابتدا جمعیتی از کروموزوم‌ها (وزن‌ها) به‌صورت تصادفی مقداردهی می‌شوند. در هر نسل، سه کروموزوم تصادفی برای تولید وکتور جهش انتخاب می‌شوند. الگوریتم تا زمانی که بهبود قابل‌توجهی در Fitness حاصل شود یا تعداد نسل‌ها به حداکثر برسد، اجرا می‌شود و در نهایت بهترین کروموزوم و Fitness بازگردانده می‌شوند.

```
def differential_evolution(data, labels, num_generations=50, population_size=200,
F=0.9, CR=0.95):
    num_weights = 23
    population = np.random.uniform(0, 1, (population_size, num_weights))
    print(f"Initial Population:\n{population}")
    best_fitness = 0
    best_chromosome = None

    for generation in range(num_generations):
        print(f"Generation {generation + 1}: Start")
        fitness_scores = np.array([fitness(chromosome, data, labels) for chromosome
in population])
        print(f"Fitness Scores: {fitness_scores}")
```

```
        if np.std(fitness_scores) < 0.001:
            print("Early stopping: No significant improvement.")
            break

        new_population = []
        for i in range(population_size):
            indices = list(range(population_size))
            indices.remove(i)
            a, b, c = population[np.random.choice(indices, 3, replace=False)]
            mutant = a + F * (b - c)
            mutant = np.clip(mutant, 0, 1)

            trial = np.copy(population[i])
            for j in range(num_weights):
                if np.random.rand() < CR:
                    trial[j] = mutant[j]

            trial_fitness = fitness(trial, data, labels)
            if trial_fitness > fitness_scores[i]:
                new_population.append(trial)
                if trial_fitness > best_fitness:
                    best_fitness = trial_fitness
                    best_chromosome = trial
            else:
                new_population.append(population[i])

        population = np.array(new_population)

        if generation % 5 == 0:
            new_individuals = np.random.uniform(0, 1, (10, num_weights))
            population = np.vstack([population, new_individuals])
            population = population[:population_size]

        print(f"New Best Fitness: {best_fitness:.4f}")

    if best_chromosome is None:
        best_chromosome = population[0]
        best_fitness = fitness_scores[0]

    print(f"Best Fitness: {best_fitness}")
    return best_chromosome, best_fitness
```

۵-۳-۴ اجرای الگوریتم DE

این بخش از کد وظیفه اجرای الگوریتم تکامل تفاضلی برای یافتن بهترین وزن‌های سیستم فازی را دارد.

```
best_weights, best_accuracy = differential_evolution(X, y, num_generations=100,
population_size=20)
print(f"Best Weights: {best_weights}")
print(f"Best Training Accuracy: {best_accuracy:.4f}")

predictions = X.apply(
    lambda row: 1 if fuzzy_sogeno_rules_with_weights(row, fuzzify_data_point(row),
best_weights) >= 0.5 else 0,
    axis=1
)
```

اندازه جمعیت ۲۰ در نظر گرفته شد و به تعداد ۱۰۰ نسل عملیات تکاملی صورت گرفت.

نتیجه به دست آمده برای آخرین نسل به این شکل است:

Best Weights: [1. 0.34126449 0.31133564 0.44101582 0.54606682 0.66252893
0.17500638 0. 0.36561042 1. 1. 0. 0. 0. 0.08249459 0. 1. 0.86665403
0.1819447 0. 0. 0.30803898 0.94292311]

۵-۳-۵ ارزیابی عملکرد

```
overall_accuracy = accuracy_score(y, predictions)
print(f"Overall Accuracy: {overall_accuracy:.4f}")

mse = mean_squared_error(y, predictions)
print(f"Mean Squared Error (MSE): {mse:.4f}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

auc = roc_auc_score(y, predictions)
print(f"Area Under the Curve (AUC): {auc:.4f}")
```

Overall Accuracy: 0.3726

Mean Squared Error (MSE): 0.6274

Root Mean Squared Error (RMSE): 0.7921

Area Under the Curve (AUC): 0.5000

فصل ۵

استفاده از روش یادگیری میشیگان

برای ایجاد پایگاه قوانین

۶-۱ الگوریتم میشیگان

الگوریتم میشیگان یکی از روش‌های الگوریتم ژنتیک در یادگیری فازی است که در آن هر کروموزوم یک قانون فازی مجزا را نشان می‌دهد. برخلاف الگوریتم پیتزبورگ، که مجموعه‌ای از قوانین را به‌طور هم‌زمان ارزیابی می‌کند، در میشیگان هر قانون به صورت جداگانه ارزیابی و بهینه‌سازی می‌شود.

۶-۲ توضیحات کد

این کد برای ایجاد یک سیستم یادگیری فازی با استفاده از الگوریتم ژنتیک به روش میشیگان طراحی شده است. ابتدا ویژگی‌های عددی از یک دیتاست فازی‌سازی می‌شوند. سپس، الگوریتم ژنتیک برای تولید قوانینی که می‌توانند داده‌ها را به دسته‌های benign و malignant طبقه‌بندی کنند، اجرا می‌شود.

توابع با یکدیگر همکاری می‌کنند تا یک سیستم یادگیری فازی را با استفاده از الگوریتم ژنتیک پیاده‌سازی کنند. الگوریتم قوانین اولیه را تولید می‌کند، آن‌ها را بهبود می‌دهد، و بهترین نسل را برای طبقه‌بندی داده‌ها انتخاب می‌کند.

نکته : اطلاعات بخش fuzzy_info از فایل json به دست آمده در فصل دوم است.

۶-۲-۱ کد توابع

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error, roc_auc_score, roc_curve

def fuzzify_dataset(data):
    return data.apply(lambda row: fuzzify_data_point(row), axis=1)

def convert_fuzzy_to_categories(fuzzy_data):
    result = {}
    for feature, memberships in fuzzy_data.items():
        max_category = max(memberships, key=memberships.get)
        result[feature] = max_category
    return result

def initialize_population(pop_size, fuzzy_info):
    population = []
    for _ in range(pop_size):
```

```
        rule_features = np.random.choice(list(fuzzy_info.keys()), size=4,
replace=False)
        rule_conditions = [
            {
                "feature": feature,
                "fuzzy": np.random.choice([f["fuzzy"] for f in
fuzzy_info[feature]]),
            }
            for feature in rule_features
        ]
        output_fuzzy = np.random.choice(["benign", "malignant"])
        population.append({"conditions": rule_conditions, "output": output_fuzzy})
    return population

def convert_to_readable_rules(rule):
    readable_conditions = " AND ".join([
        f"{condition['feature']} IS {condition['fuzzy']}"
        for condition in rule["conditions"]
    ])
    return f"IF {readable_conditions}, THEN OUTPUT IS {rule['output']}"

def evaluate_rule(rule, data):
    predictions = []
    for _, crisp_data_point in data.iterrows():
        fuzzy_data_point = fuzzify_data_point(crisp_data_point)
        match = True
        for condition in rule["conditions"]:
            feature = condition["feature"]
            fuzzy_label = condition["fuzzy"]
            if fuzzy_data_point[feature][fuzzy_label] == 0:
                match = False
                break
        predictions.append(rule["output"] if match else None)
    return predictions

def evaluate_population(population, data):
    final_predictions = []
    for _, crisp_data_point in data.iterrows():
        for rule in population:
            fuzzy_data_point = fuzzify_data_point(crisp_data_point)
            match = True
            for condition in rule["conditions"]:
                feature = condition["feature"]
                fuzzy_label = condition["fuzzy"]
                if fuzzy_data_point[feature][fuzzy_label] == 0:
                    match = False
                    break
            if match:
```

```
                final_predictions.append(1 if rule["output"] == "malignant" else 0)
                break
        else:
            final_predictions.append(0)
    return final_predictions

def crossover(parent1, parent2):
    point = np.random.randint(1, len(parent1["conditions"]))
    child1_conditions = parent1["conditions"][:point] +
parent2["conditions"][point:]
    child2_conditions = parent2["conditions"][:point] +
parent1["conditions"][point:]
    child1 = {"conditions": child1_conditions, "output": parent1["output"]}
    child2 = {"conditions": child2_conditions, "output": parent2["output"]}
    return child1, child2

def mutate(rule, fuzzy_info, mutation_rate):
    if np.random.rand() < mutation_rate:
        idx = np.random.randint(len(rule["conditions"]))
        feature = rule["conditions"][idx]["feature"]
        new_fuzzy = np.random.choice([f["fuzzy"] for f in fuzzy_info[feature]])
        rule["conditions"][idx]["fuzzy"] = new_fuzzy
    if np.random.rand() < mutation_rate:
        rule["output"] = "benign" if rule["output"] == "malignant" else "malignant"
    return rule

def genetic_algorithm(data, labels, pop_size=20, num_generations=100,
mutation_rate=0.1):
    fuzzy_info = {
        "radius_mean": [{"fuzzy": "small"}, {"fuzzy": "medium"}, {"fuzzy":
"large"}],
        "texture_mean": [{"fuzzy": "soft"}, {"fuzzy": "medium"}, {"fuzzy":
"hard"}],
        "perimeter_mean": [{"fuzzy": "small"}, {"fuzzy": "medium"}, {"fuzzy":
"large"}],
        "area_mean": [{"fuzzy": "small"}, {"fuzzy": "medium"}, {"fuzzy": "large"}],
        "smoothness_mean": [{"fuzzy": "soft"}, {"fuzzy": "medium"}, {"fuzzy":
"hard"}],
        "compactness_mean": [{"fuzzy": "low"}, {"fuzzy": "medium"}, {"fuzzy":
"high"}],
        "concavity_mean": [{"fuzzy": "low"}, {"fuzzy": "medium"}, {"fuzzy":
"high"}],
        "concave_points_mean": [{"fuzzy": "low"}, {"fuzzy": "medium"}, {"fuzzy":
"high"}],
    }
    population = initialize_population(pop_size, fuzzy_info)

    for generation in range(num_generations):
```

```
        print(f"\nGeneration {generation + 1}")
        for i, rule in enumerate(population):
            readable_rule = convert_to_readable_rules(rule)
            print(f"Rule {i + 1}: {readable_rule}")

        fitness_scores = evaluate_population(population, data)

        print(f"Best Fitness in Generation {generation + 1}:
{max(fitness_scores)}")

        new_population = []
        for i in range(0, pop_size, 2):
            parent1, parent2 = population[i], population[i + 1]
            child1, child2 = crossover(parent1, parent2)
            new_population.extend([mutate(child1, fuzzy_info, mutation_rate),
mutate(child2, fuzzy_info, mutation_rate)])
        population = new_population

    return population

raw_data = pd.read_csv("data.csv")
labels = raw_data['diagnosis'].map({'B': 0, 'M': 1})
raw_data.rename(columns={'concave points_mean': 'concave_points_mean'},
inplace=True)

features = [
    'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
    'smoothness_mean', 'compactness_mean', 'concavity_mean', 'concave_points_mean'
]

final_population = genetic_algorithm(raw_data[features], labels)

print("\nFinal Population:")
for i, rule in enumerate(final_population):
    readable_rule = convert_to_readable_rules(rule)
    print(f"Rule {i + 1}: {readable_rule}")
```

بر اساس ۱۰۰ نسل این الگوریتم بهترین قوانین به دست آمده شامل این ۲۰ قوانین هستند:

Final Population:

Rule 1: IF area_mean IS small AND radius_mean IS medium AND compactness_mean IS medium AND concavity_mean IS low, THEN OUTPUT IS benign

Rule 2: IF radius_mean IS small AND area_mean IS medium AND texture_mean IS medium AND perimeter_mean IS large, THEN OUTPUT IS malignant

Rule 3: IF texture_mean IS soft AND area_mean IS large AND perimeter_mean IS medium AND smoothness_mean IS hard, THEN OUTPUT IS benign

Rule 4: IF concave_points_mean IS high AND concavity_mean IS medium AND radius_mean IS large AND compactness_mean IS low, THEN OUTPUT IS malignant

Rule 5: IF perimeter_mean IS medium AND concavity_mean IS low AND perimeter_mean IS large AND texture_mean IS medium, THEN OUTPUT IS malignant

Rule 6: IF texture_mean IS soft AND compactness_mean IS medium AND radius_mean IS small AND concavity_mean IS high, THEN OUTPUT IS benign

Rule 7: IF concave_points_mean IS low AND texture_mean IS medium AND concavity_mean IS low AND compactness_mean IS low, THEN OUTPUT IS benign

Rule 8: IF area_mean IS small AND perimeter_mean IS large AND area_mean IS medium AND radius_mean IS medium, THEN OUTPUT IS benign

Rule 9: IF compactness_mean IS low AND concavity_mean IS low AND concave_points_mean IS high AND radius_mean IS medium, THEN OUTPUT IS malignant

Rule 10: IF radius_mean IS large AND compactness_mean IS high AND area_mean IS medium AND texture_mean IS medium, THEN OUTPUT IS benign

Rule 11: IF smoothness_mean IS medium AND concavity_mean IS low AND radius_mean IS medium AND radius_mean IS large, THEN OUTPUT IS malignant

Rule 12: IF smoothness_mean IS hard AND texture_mean IS soft AND texture_mean IS soft AND concave_points_mean IS low, THEN OUTPUT IS benign

Rule 13: IF radius_mean IS small AND concave_points_mean IS low AND area_mean IS large AND compactness_mean IS low, THEN OUTPUT IS benign

Rule 14: IF smoothness_mean IS hard AND perimeter_mean IS large AND compactness_mean IS low AND concave_points_mean IS high, THEN OUTPUT IS malignant

Rule 15: IF compactness_mean IS medium AND perimeter_mean IS small AND smoothness_mean IS medium AND texture_mean IS hard, THEN OUTPUT IS malignant

Rule 16: IF smoothness_mean IS hard AND area_mean IS large AND concave_points_mean IS medium AND area_mean IS small, THEN OUTPUT IS benign

Rule 17: IF concave_points_mean IS medium AND concavity_mean IS low AND compactness_mean IS high AND smoothness_mean IS soft, THEN OUTPUT IS benign

Rule 18: IF compactness_mean IS high AND area_mean IS small AND radius_mean IS small AND concavity_mean IS low, THEN OUTPUT IS benign

Rule 19: IF concavity_mean IS medium AND perimeter_mean IS large AND concave_points_mean IS low AND perimeter_mean IS medium, THEN OUTPUT IS malignant

Rule 20: IF concavity_mean IS high AND area_mean IS small AND area_mean IS large AND compactness_mean IS low, THEN OUTPUT IS malignant

۲-۲-۶ ارزیابی عملکرد

```
accuracy = accuracy_score(labels, final_predictions)
print(f"Accuracy: {accuracy:.4f}")

final_predictions = evaluate_population(final_population, raw_data[features])

mse = mean_squared_error(labels, final_predictions)
print(f"Mean Squared Error (MSE): {mse:.4f}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

auc = roc_auc_score(labels, final_predictions)
print(f"Area Under the Curve (AUC): {auc:.4f}")
```

نتایج به دست آمده به شکل زیر است:

Accuracy: 0.6292

Mean Squared Error (MSE): 0.1459

Root Mean Squared Error (RMSE): 0.3819

Area Under the Curve (AUC): 0.8129

فصل ۷

استفاده از روش یادگیری پیتزبورگ

برای ایجاد پایگاه قوانین

۷-۱ الگوریتم پیتزبورگ

الگوریتم تکاملی پیتزبورگ یکی از رویکردهای تکاملی برای یادگیری قوانین فازی است که در آن هر کروموزوم یک مجموعه از قوانین فازی را نمایش می‌دهد. برخلاف الگوریتم میشیگان، که در آن هر کروموزوم نماینده یک قانون منفرد است، در پیتزبورگ ، قوانین به صورت یکجا و به عنوان بخشی از یک سیستم واحد تکامل می‌یابند

۷-۲ توضیحات کد

این کد برای ایجاد یک سیستم یادگیری فازی با استفاده از الگوریتم ژنتیک به روش پیتزبورگ طراحی شده است. ابتدا ویژگی‌های عددی از یک دیتاست فازی‌سازی می‌شوند. سپس، الگوریتم ژنتیک برای تولید قوانینی که می‌توانند داده‌ها را به دسته‌های benign و malignant طبقه‌بندی کنند، اجرا می‌شود.

این توابع با یکدیگر همکاری می‌کنند تا یک سیستم یادگیری فازی را با استفاده از الگوریتم ژنتیک پیاده‌سازی کنند. الگوریتم قوانین اولیه را تولید می‌کند، آن‌ها را بهبود می‌دهد، و بهترین کروموزم را برای طبقه‌بندی داده‌ها انتخاب می‌کند.

۷-۲-۱ کد توابع

```
import numpy as np
import pandas as pd
import random
from sklearn.metrics import accuracy_score, mean_squared_error, roc_auc_score


def generate_rule(features, output_labels):
    rule_conditions = []
    selected_features = random.sample(features, 3)
    for feature in selected_features:
        rule_conditions.append({
            "feature": feature,
            "fuzzy": random.choice(["low", "medium", "high"]),
        })
    output = random.choice(output_labels)
    return {"conditions": rule_conditions, "output": output}

def initialize_population(pop_size, num_rules, features, output_labels):
    population = []
    for _ in range(pop_size):
```

```
        rules = [generate_rule(features, output_labels) for _ in range(num_rules)]
        population.append(rules)
    return population

def evaluate_rule(rule, data, labels):
    correct_predictions = 0
    for idx, data_point in data.iterrows():
        match = all(
            data_point[cond["feature"]] == cond["fuzzy"]
            for cond in rule["conditions"]
        )
        if match and rule["output"] == labels.iloc[idx]:
            correct_predictions += 1
    return correct_predictions / len(data)

def evaluate_population(population, data, labels):
    fitness_scores = []
    for rules in population:
        fitness = np.mean([evaluate_rule(rule, data, labels) for rule in rules])
        fitness_scores.append(fitness)
    return fitness_scores

def crossover(parent1, parent2):
    point = random.randint(1, len(parent1) - 1)
    child1 = parent1[:point] + parent2[point:]
    child2 = parent2[:point] + parent1[point:]
    return child1, child2

def mutate(rules, mutation_rate, features, output_labels):
    for rule in rules:
        if random.random() < mutation_rate:
            rule["conditions"][random.randint(0, len(rule["conditions"]) -
1)]["fuzzy"] = random.choice(
                ["low", "medium", "high"]
            )
        if random.random() < mutation_rate:
            rule["output"] = random.choice(output_labels)
    return rules

def genetic_algorithm(data, labels, features, output_labels, pop_size=20,
num_rules=10, num_generations=100, mutation_rate=0.4):
    population = initialize_population(pop_size, num_rules, features,
output_labels)
    best_fitness = 0
    best_solution = None

    for generation in range(num_generations):
        fitness_scores = evaluate_population(population, data, labels)
```

```
        sorted_indices = np.argsort(fitness_scores)[::-1]

        if fitness_scores[sorted_indices[0]] > best_fitness:
            best_fitness = fitness_scores[sorted_indices[0]]
            best_solution = population[sorted_indices[0]]

        print(f"Generation {generation + 1}: Best Fitness = {best_fitness}")

        best_rule_set = population[sorted_indices[0]]
        print("Best Rule Set:")
        for rule in best_rule_set:
            conditions = " AND ".join(
                [f"{cond['feature']} IS {cond['fuzzy']}" for cond in
rule["conditions"]]
            )
            print(f"IF {conditions}, THEN OUTPUT IS {rule['output']}")

        new_population = []
        for i in range(0, pop_size, 2):
            parent1 = population[sorted_indices[i]]
            parent2 = population[sorted_indices[i + 1]]
            child1, child2 = crossover(parent1, parent2)
            new_population.extend([mutate(child1, mutation_rate, features,
output_labels), mutate(child2, mutation_rate, features, output_labels)])
        population = new_population

    return best_solution, best_fitness

data = pd.read_csv("data.csv")
data.rename(columns={'concave points_mean': 'concave_points_mean'}, inplace=True)

labels = data["diagnosis"].map({"B": "benign", "M": "malignant"})
data = data.drop(columns=["diagnosis"])

features = [
    "radius_mean",
    "texture_mean",
    "perimeter_mean",
    "area_mean",
    "smoothness_mean",
    "compactness_mean",
    "concavity_mean",
    "concave_points_mean",
]
data = data[features]

def fuzzify(value):
    if value < 0.33:
```

```
        return "low"
    elif value < 0.66:
        return "medium"
    else:
        return "high"

for feature in features:
    data[feature] = data[feature].apply(lambda x: fuzzify(x / data[feature].max()))

output_labels = ["benign", "malignant"]
best_solution, best_fitness = genetic_algorithm(data, labels, features,
output_labels)

print("Best Solution:")
for rule in best_solution:
    conditions = " AND ".join(
        [f"{cond['feature']} IS {cond['fuzzy']}" for cond in rule["conditions"]]
    )
    print(f"IF {conditions}, THEN OUTPUT IS {rule['output']}")
print(f"Best Fitness: {best_fitness}")

final_predictions = []
for _, data_point in data.iterrows():
    prediction = None
    for rule in best_solution:
        match = all(
            data_point[cond['feature']] == cond['fuzzy']
            for cond in rule['conditions']
        )
        if match:
            prediction = 1 if rule['output'] == 'malignant' else 0
            break
    final_predictions.append(prediction if prediction is not None else 0)

final_predictions = ['malignant' if pred == 1 else 'benign' for pred in
final_predictions]

accuracy = accuracy_score(labels, final_predictions)
print(f"Accuracy: {accuracy:.4f}")

mse = mean_squared_error(labels.map({'benign': 0, 'malignant': 1}), [1 if pred ==
'malignant' else 0 for pred in final_predictions])
print(f"Mean Squared Error (MSE): {mse:.4f}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
auc = roc_auc_score(labels.map({'benign': 0, 'malignant': 1}), [1 if pred ==
'malignant' else 0 for pred in final_predictions])
print(f"Area Under the Curve (AUC): {auc:.4f}")
```

بر اساس ۱۰۰ نسل این الگوریتم بهترین کروموزم به دست آمده شامل این ۱۰ قوانین است:

Best Solution:

IF perimeter_mean IS medium AND concave_points_mean IS low AND concavity_mean IS low, THEN OUTPUT IS malignant

IF radius_mean IS medium AND compactness_mean IS medium AND concave_points_mean IS medium, THEN OUTPUT IS malignant

IF area_mean IS high AND compactness_mean IS low AND concavity_mean IS low, THEN OUTPUT IS malignant

IF concave_points_mean IS medium AND perimeter_mean IS low AND compactness_mean IS low, THEN OUTPUT IS malignant

IF texture_mean IS high AND radius_mean IS low AND smoothness_mean IS high, THEN OUTPUT IS malignant

IF perimeter_mean IS medium AND concavity_mean IS low AND area_mean IS high, THEN OUTPUT IS malignant

IF radius_mean IS high AND perimeter_mean IS low AND concave_points_mean IS medium, THEN OUTPUT IS malignant

IF concave_points_mean IS high AND perimeter_mean IS high AND radius_mean IS high, THEN OUTPUT IS malignant

IF perimeter_mean IS medium AND area_mean IS low AND texture_mean IS high, THEN OUTPUT IS malignant

IF radius_mean IS medium AND perimeter_mean IS high AND compactness_mean IS medium, THEN OUTPUT IS malignant

۲-۲-۷ ارزیابی عملکرد

```
accuracy = accuracy_score(labels, final_predictions)
print(f"Accuracy: {accuracy:.4f}")

mse = mean_squared_error(labels, final_predictions)
print(f"Mean Squared Error (MSE): {mse:.4f}")

rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
```

```
auc = roc_auc_score(labels, final_predictions)
print(f"Area Under the Curve (AUC): {auc:.4f}")
```

نتایج به دست آمده به شکل زیر است:

Accuracy: 0.3146

Mean Squared Error (MSE): 0.6854

Root Mean Squared Error (RMSE): 0.8279

Area Under the Curve (AUC): 0.3791

فصل ۸

سیستم فازی نوع ۲ و مقایسه با فازی نوع اول

۸-۱ فازی نوع ۲

فازی نوع ۲ نسخه پیشرفته‌ای از فازی نوع ۱ است که برای مدیریت عدم قطعیت‌های بیشتر طراحی شده است. در این سیستم، به جای مقدار دقیق عضویت، یک بازه از مقادیر ممکن تعریف می‌شود. این بازه که به آن "اثر عدم قطعیت (FOU) "گفته می‌شود، کمک می‌کند تا سیستم در شرایطی که داده‌ها دارای ابهام یا نویز هستند، بهتر تصمیم‌گیری کند. فازی نوع ۲ در کاربردهایی مانند پزشکی و سیستم‌های پیچیده استفاده می‌شود.

۸-۲ توضیحات کد

۱. آماده‌سازی داده‌ها : داده‌ها از فایل بارگذاری شده، مقیاس‌بندی می‌شوند و به دو مجموعه آموزشی و تست تقسیم می‌شوند.
۲. محاسبه عضویت فازی نوع ۲: تابع fuzzify_type2 برای هر ویژگی مقدار عضویت پایین و بالا را محاسبه می‌کند.
۳. ترکیب عضویت‌ها : تابع apply_fuzzy_rules از عضویت‌های پایین و بالا برای همه ویژگی‌ها یک خروجی نهایی فازی می‌سازد.
۴. طبقه‌بندی بر اساس آستانه: خروجی‌های فازی با مقایسه با یک آستانه (threshold) به دسته‌های مثبت (۱) یا منفی (۰) طبقه‌بندی می‌شوند.

۸-۲-۱ کد توابع

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score,
roc_curve
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("data.csv")
data.rename(columns={'concave points_mean': 'concave_points_mean'}, inplace=True)
data = data.drop(['id', 'Unnamed: 32'], axis=1)
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})
```

```
X = data.drop(['diagnosis'], axis=1)
y = data['diagnosis']

scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=42)

def fuzzify_type2(value, bounds, uncertainty=0.1):
    lower_bound, upper_bound = bounds
    lower_membership = max(0, min(1, (upper_bound - (value - uncertainty)) /
(upper_bound - lower_bound)))
    upper_membership = max(0, min(1, ((value + uncertainty) - lower_bound) /
(upper_bound - lower_bound)))
    return lower_membership, upper_membership

def apply_fuzzy_rules(row, feature_bounds):
    results = []
    for feature, bounds in feature_bounds.items():
        value = row[feature]
        lower, upper = fuzzify_type2(value, bounds)
        low_output = lower * 0.3
        high_output = upper * 0.7
        results.append((low_output + high_output) / 2)
    return np.mean(results)

feature_bounds = {feature: (X_train[feature].min(), X_train[feature].max()) for
feature in X_train.columns}

X_train['fuzzy_output'] = X_train.apply(lambda row: apply_fuzzy_rules(row,
feature_bounds), axis=1)
X_test['fuzzy_output'] = X_test.apply(lambda row: apply_fuzzy_rules(row,
feature_bounds), axis=1)

threshold = (X_train['fuzzy_output'].mean() + X_train['fuzzy_output'].median()) / 2

X_train['fuzzy_class'] = X_train['fuzzy_output'].apply(lambda x: 1 if x > threshold
else 0)
X_test['fuzzy_class'] = X_test['fuzzy_output'].apply(lambda x: 1 if x > threshold
else 0)
```

۲-۲-۸ ارزیابی عملکرد

```
train_accuracy = accuracy_score(y_train, X_train['fuzzy_class'])
test_accuracy = accuracy_score(y_test, X_test['fuzzy_class'])
roc_auc = roc_auc_score(y_test, X_test['fuzzy_output'])

print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Test Accuracy: {test_accuracy:.2f}")
print(f"ROC AUC: {roc_auc:.2f}")
```

نتایج به دست آمده به این شکل است:

Training Accuracy: 0.86

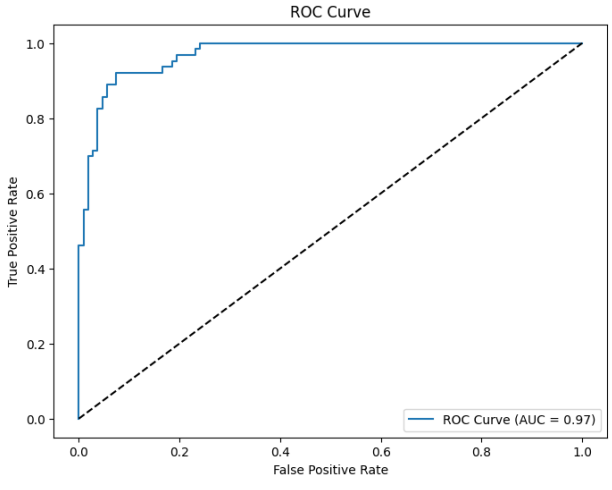
Test Accuracy: 0.87

ROC AUC: 0.97

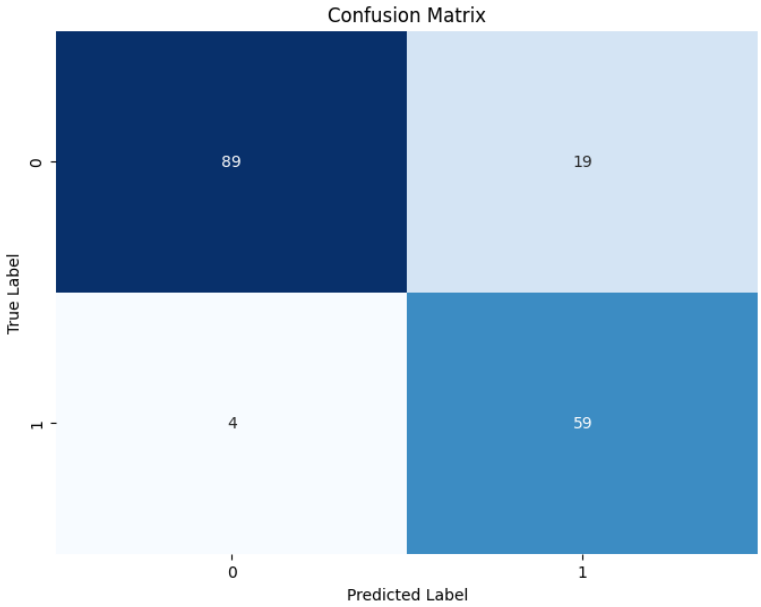
با استفاده از کد ماتریس سردرگمی و نمودار ROC نمایش داده شده است:

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

fpr, tpr, _ = roc_curve(y_test, X_test['fuzzy_output'])
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```



ماتریس سردرگمی



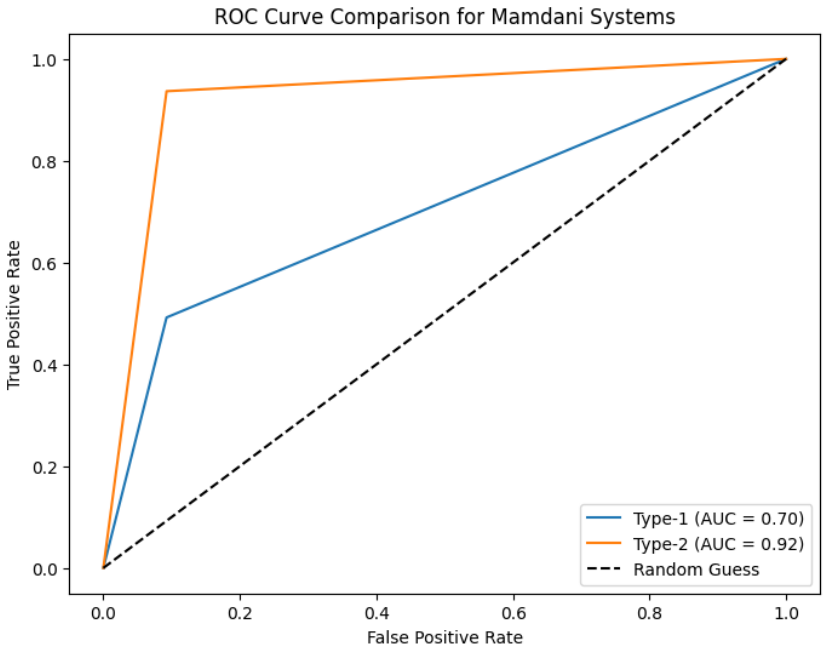
ماتریس سردرگمی

۳-۸ مقایسه فازی نوع ۱ و نوع ۲

به منظور ارزیابی ۲ مدل، روی داده های تست ارزیابی صورت گرفته است. که نتیجه به این شکل است:

```
plt.figure(figsize=(8, 6))
for system, data in metrics.items():
    plt.plot(data['fpr'], data['tpr'], label=f'{system} (AUC =
{data["roc_auc"]:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison for Mamdani Systems")
plt.legend(loc="lower right")
plt.show()


# Print Metrics
for system, data in metrics.items():
    print(f"{system} Accuracy: {data['accuracy']:.2f}")
    print(f"{system} AUC: {data['roc_auc']:.2f}")
```



.....

سیستم فازی نوع ۱در مقایسه با سیستم فازی نوع ۲ عملکرد کمتری در جداسازی نمونه‌های مثبت و منفی دارد و مقدار AUC کمتری نشان می‌دهد. در مقابل، سیستم فازی نوع ۲ با استفاده از محدوده‌های عدم قطعیت در عضویت‌ها، انعطاف‌پذیری بیشتری داشته و منحنی ROC آن به گوشه بالا-چپ نزدیک‌تر است که نشان‌دهنده دقت بالاتر در پیش‌بینی نمونه‌ها است. مقدار AUC بالاتر (مانند ۰,۹۷) نشان می‌دهد که سیستم نوع ۲ در مدیریت داده‌های پیچیده و جداسازی دقیق‌تر عملکرد بهتری دارد.