محمد اسدالهی

Mohamad.asa1994@gmail.com

دانشگاه علوم و تحقیقات تهران

تیرماه 1400

فهرست

مقدمه:

در این پروژه ابتداد داده ها پیش پردازش خواهد شدو سپس داده ها به دو بخش داده های آموزش و داده های تست تقسیم میشوند. سپس مدلی بر اساس آنچه در بخش پیاده سازی MLP تشریح میشود ساخته خواهد شد و شبکه عصبی رابا استفاده از داده های آموزشی و الگوریتم Error Backpropagation آموزش می دهیم(پارامتر های مدل یا همان وزن های شبکه را بروز رسانی می کنیم). سپس با استفاده از داده های تست که از قبل مشخص کرده ایم و در فرایند آموزش از آنها استفاده نشده است اقدام به ارزیابی مدل می کنیم. باید توجه شود که مدل از قبل داده های تست را مشاهده نکرده است. در آخر نمودار منحنی ROC را برای مدل رسم خواهیم کرد و همچنین درصد موفقیت مدل در پیش بینی داده های تست. و در پایان اقدام به چاپ پیش بینی برای داده هایی که برای قسمت تست در نظر گرفته ایم می کنیم. و درپایان بر اساس نتایج بدست آمده به نتیجه گیری در مورد میزان موفقیت شبکه عصبی چندلایه پرسپترون برای پیش بینی بیماری با استفاده از دیتاست استفاده شده و همچنین تحلیل کارکرد آن خواهیم پرداخت

توضیحات و مشخصات دیتاست:

دیتا ست استفاده شده در این پروژه مربوط به بیماران سرطانی در شهر ویسکانسین است. که دارای 699 نمونه (بیمار) می باشد ک به صورت پراکنده توسط دکتر Wolberg در کلینیکی که در آن کار میکرده است جمع آوری شده اند. هر گروه از این دیتابیس در بین سال های زیر جمع آوری شده است.

گروه اول 367 :1 نمونه (January 1989)

گروه دوم 70 نمونه (October 1989)

گروه سوم 31 نمونه (February 1990)

گروه چهارم 17 نمونه (April 1990)

گروه پنجم 48 نمونه (August 1990)

گروه ششم 49 نمونه (Updated January 1991)

گروه هفتم 31 نمونه (June 1991)

گروه هشتم 86 نمونه (November 1991)

هر نمونه(بیمار)از این دیتاست دارای 9 ویژگی برای بیماری سرطان همچون میزان چسبندگی تومور میزان میوزسرطانی در خون و غیره می باشد. ستون آخر دیتاست دارای دو عدد 2 و 4 می باشد که هر کدام به ترتیب به سرطان سینه نوع benign و malignant اشاره دارند. همچنین بعضی از نمونه های این دیتا ست دارای مقادیر ناموجود(Missing Values) می باشند.

دیتاست مورد نظر در لینک زیر قرار دارد:

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29

Original Wisconsin Breast Cancer Database

Data Set Characteristics:	**Multivariate**	**Number of Instances:**	**699**	**Area:**	**Life**
Attribute Characteristics:	**Integer**	**Number of Attributes:**	**10**	**Date Donated**	**1992-07-15**
Associated Tasks:	**Classification**	**Missing Values?**	**Yes**	**Number of Web Hits:**	**710744**

Set Information:

Samples arrive periodically as Dr. Wolberg reports his clinical cases. The database therefore reflects this chronological grouping of the data. This grouping information appears immediately below, having been removed from the data itself:

Group 1: 367 instances (January 1989)
Group 2: 70 instances (October 1989)
Group 3: 31 instances (February 1990)
Group 4: 17 instances (April 1990)
Group 5: 48 instances (August 1990)
Group 6: 49 instances (Updated January 1991)
Group 7: 31 instances (June 1991)
Group 8: 86 instances (November 1991)

Total: 699 points (as of the donated datbase on 15 July 1992)

Attribute Information:

1. Sample code number: id number
2. Clump Thickness: 1 - 10
3. Uniformity of Cell Size: 1 - 10
4. Uniformity of Cell Shape: 1 - 10
5. Marginal Adhesion: 1 - 10
6. Single Epithelial Cell Size: 1 - 10
7. Bare Nuclei: 1 - 10
8. Bland Chromatin: 1 - 10
9. Normal Nucleoli: 1 - 10
10. Mitoses: 1 - 10
11. Class: (2 for benign, 4 for malignant)

پیش پردازش اطلاعات دیتاست:

ابتدا دیتاست باید مورد پیش پردازش قرار بگیرد. اولین مرحله حذف ستون "id number" بیماران است، که ستون اول دیتاست را تشکیل میدهد خواهد بود. همچنین مقادیر برچسب های نوع بیماری که ستون اخر را تشکیل می دهد از 2 و 4 باید به 0 و 1 تغییر پیدا کند. مهمترین مرحله پیش پردازش داده ها پرکردن مقادیر مجهول یا ناموجود با استفاده از میانگین ستون یا همان ویژگی مورد نظر خواهد بود.

بخش واکشی داده های دیتاست توسط کد های زیر انجام گرفته است:

```
WRITTEN BY MOHAMMAD ASADOLAHI
records = []
with open('./cancer.data') as f:
    for line in f.readlines():
        records.append(np.fromstring(line.rstrip('\n'), dtype=float,
sep=","))
```

پس از اجرای این کد تمامی رکورد های دیتاست در یک لیست به طول 699 آرایه به نام "records" قرارخواهد گرفت.

توسط کد زیر تمامی رکورد ها را برای پیش پردازش به یک دیتا فریم تبدیل می کنیم. باید توجه شود که استفاده از کتابخانه دیتا فریم تنها جهت پیش پردازش داده ها بوده است و تنها برای تصویر سازی از داده ها انجام گرفته است و بقیه پیش پردازش داده ها جز حذف ستون ها به صورت دستی انجام گرفته است:

```
WRITTEN BY MOHAMMAD ASADOLAHI
samples = pd.DataFrame(records, columns=["id", "Clump Thickness", "Cell
Size Uniformity","Cell Shape Uniformity","Marginal Adhesion", "Single
Epithelial Cell Size","Bare Nuclei","Bland Chromatin", "Normal Nucleoli",
"Mitoses", "class"])
```

پس از اجری کد بالا تمامی رکورد های واکشی شده در دیتا فریم "samples" رونویسی(کپی) میشوند.سپس برای مصور سازی داده ها از کد زیر استفاده میشود که 5 رکورد بالای دیتا فریم را چاپ می کند:

```
WRITTEN BY MOHAMMAD ASADOLAHI
samples.head(5)  # show top 5 fetch records
```

	id	Clump Thickness	Cell Size Uniformity	Cell Shape Uniformity	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	class
0	1000025.0	5.0	1.0	1.0	1.0	2.0	1.0	3.0	1.0	0.0	2.0
1	1002945.0	5.0	4.0	4.0	5.0	7.0	10.0	3.0	2.0	1.0	2.0
2	1015425.0	3.0	1.0	1.0	1.0	2.0	2.0	3.0	1.0	1.0	2.0
3	1016277.0	6.0	8.0	8.0	1.0	3.0	4.0	3.0	7.0	1.0	2.0
4	1017023.0	4.0	1.0	1.0	3.0	2.0	1.0	3.0	1.0	1.0	2.0

نکته: این جدول توسط اجرای کد فوق در محیط jupiter notebook چاپ شده است که به صورت اتوماتیک برای دیتافریم ها اقدام به رسم جداول فرمت شده میکند و نتیجه اجرای کد فوق در سایر محیط های برنامه نویسی پایتون چاپ جدول فوق به صورت متنی همانند زیر خواهد بود:

```
 id          Clump      Cell Size   Cell Shape   Marginal   Single Epithelial   Bare      Bland      Normal     Mitoses  class
             Thickness  Uniformity  Uniformity   Adhesion   Cell Size           Nuclei    Chromatin  Nucleoli
1000025.0    5.0        1.0         1.0          1.0          2.0               1.0       3.0        1.0        0.0      2.0
1002945.0    5.0        4.0         4.0          5.0          7.0               10.0      3.0        2.0        1.0      2.0
1015425.0    3.0        1.0         1.0          1.0          2.0               2.0       3.0        1.0        1.0      2.0
1016277.0    6.0        8.0         8.0          1.0          3.0               4.0       3.0        7.0        1.0      2.0
1017023.0    4.0        1.0         1.0          3.0          2.0               1.0       3.0        1.0        1.0      2.0
```

سپس اقدام به حذف ستون id که همان شناسه بیمار است و در دسته بندی بیماران نقشی ندارد، می کنیم:

```
WRITTEN BY MOHAMMAD ASADOLAHI
samples = samples.drop("id", axis=1)  # remove id witch is useless for
classification
```

سپس تمامی class یه رده داده ها را در متغییری قرا میدهیم تا نوع بیماری هرکدام از بیماران را برای آموزش و تست مدل نگهداری کنیم و این ستون را از لیست "samples" حذف می کنیم:

```
WRITTEN BY MOHAMMAD ASADOLAHI
labels = samples["class"]  # transfer labels to another dataframe

samples = samples.drop("class", axis=1)  # remove class from features
vector
```

همانطور که قبلا ذکر شد در این دیتابیس در قسمت "class" یا رده برای سرطان نوع "benign" عدد 2 و برای سرطان نوع "malignant" درج شده است که آنها را که در آرایه درج کردیم به ترتیب به اعداد 0 و 1 جایگزین میکنیم.

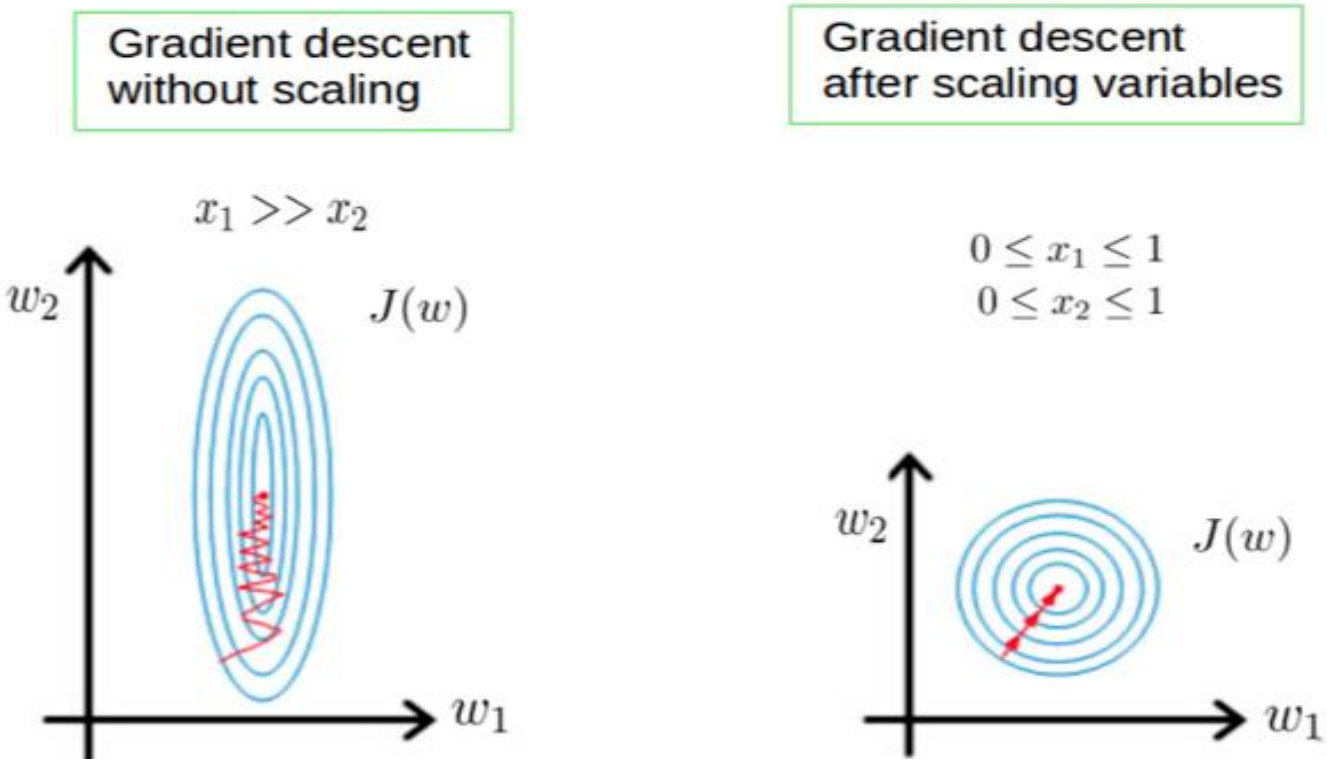
```
WRITTEN BY MOHAMMAD ASADOLAHI
labels[labels == 2] = 0  # "replace 2 with 0 for benign cancer"

labels[labels == 4] = 1  # "replace 4 with 1 for malignant cancer"
```

پس از انجام این مراحل در تمامی ستون های آرایه "labels" هرجا که مقدار 0 قرار دارد که نشان دهنده مقادیر نا موجود یا "missing values" است را با میانگین آن ستون جایگزین می کنیم، برای مثال در تمامی رکورد هایی که در ستون " Cell Size Uniformity" دارای مقدار 0 هستند را با میانگین این ستون جایگزین می کنیم.

```
WRITTEN BY MOHAMMAD ASADOLAHI
# replacing missig values with the mean of column-------------------------
------------------------
for each in samples.columns:
    samples[each] = samples[each].replace(to_replace=0,
                                          value=int(samples[each].mean()))
```

بعد از این مراحل ما مقادیر تمامی ویژگی ها را برای اینکه سرعت همگرایی را در گرادیان کاهشی افزایش دهیم به یک مقیاس میبریم (Feature Scaling) برای مثال تمامی آنها را به مقادیر بین 0 و 1 میبریم تا سرعت همگرایی افزایش پیدا کند.

شکل زیر تاثیر هم مقیاس کردن ویژگی ها را در همگرایی الگوریتم نشان می دهد:



با استفاده از تقسیم تمامی مقادیر بر بزرگتریم مقدار هر ویژگی (ستون) ما تمامی داده ها را به مقیاس بین 0 و 1 می بریم:

```
WRITTEN BY MOHAMMAD ASADOLAHI
# scaling all numeral features between [0 to 1] --------------------------
-------------------------
for each in samples.columns:
    samples[each] = samples[each] / samples[each].max()
```

تقسیم داده های آموزشی برای آموزش و ارزیابی:

در این مرحله داده ها را به دو قسمت داده های آموزشی(برای آموزش شبکه عصبی که حدودا 65 درصد کل داده های دیتاست است) و داده های تست(برای تست و ارزیابی مدل آموزش دیده و همچنین ارزیابی در حین آموزش مدل که حدودا 35 درصد کل داده های دیتا ست است) تقسیم می کنیم:

```
WRITTEN BY MOHAMMAD ASADOLAHI
# split samples into train and test samples
trainSamples = (samples[0:450]).to_numpy()  # 450 sample to train the
model
trainLabels = (labels[0:450]).to_numpy()

testSamples = (samples[450:699]).to_numpy()  # 250 sample to test the
model
testlabels = (labels[450:699]).to_numpy()
```

تا به اینجا تمامی پیش پردازش ها بر روی دیتاست ما انجام شده و این داده ها آماده استفاده برای آموزش و ارزیابی نتایج آموزش بر روی شبکه عصبی چند لایه پرسپترون هستند. در مرحله بعدی اقدام به تعریف مدل شبکه عصبی پرسپترون چندلایه و سپس آموزش آن خواهیم کرد.

طراحی و پیاده سازی MLP

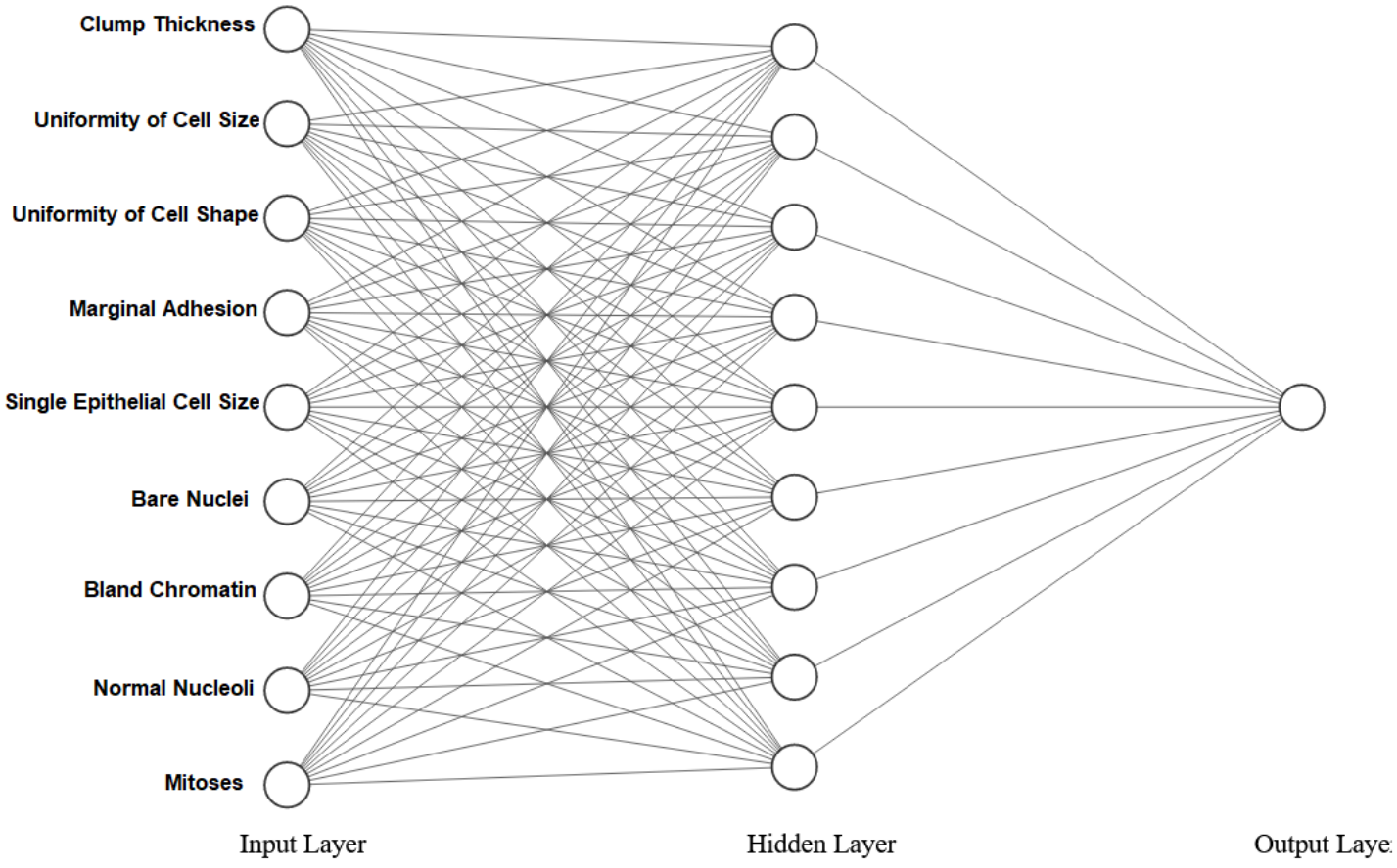
مدل ماشین یادگیر در این پروژه یک شبکه عصبی multi layer perceptron (شبکه عصبی چندلایه پرستپترون) دارای سه لایه است که دارای یک لایه ورودی دارای 9 نرون می باشد که به تعداد بردار ویژگی های نمونه های ما می باشد. یعنی اینکه هر نمونه(بیمار در دیتاست این پروژه) دارای 9 ویژگی عددی در مورد بیماری سرطان سینه می باشدکه به ترتیب ورودی:

Clump Thickness,Uniformity of Cell Size,Uniformity of Cell Shape,Marginal Adhesion,Single Epithelial Cell Size,Bare Nuclei,Bland Chromatin,Normal Nucleoli Mitoses

هستند که در بخش دیتاست به تفضیل در موردآنها توضیح داده شده است. لایه دوم یا لایه ی پنهان این شبکه دارای 9 نرون می باشد که در پیاده سازی به نرون های "Dense" معروف هستند. زیرا هر کدام از نرون های لایه ی میانی به تمامی 9 نرون لایه قبلی (لایه ورودی) متصل هستند که در مجموع 9*9 اتصال برابر با 81 اتصال بین لایه اول و لایه دوم می باشد. چون در این مثال دسته بند ما بین دو نوع سرطان بخصوص سعی در تفکیک دارد لذا نوع دسته بندی ما دسته بندی دو دویی(BinaryClassification) می باشد که میتوان لایه خروجی را به دو روش تنظیم و ارایه کرد.

روش پیاده سازی:

در پیاده سازی ایم مدل از شبکه MLP برای این پروژه استفاده شد که دارای یک نرون در خروجی می باشد که پس از پیش خور کردن شبکه در خروجی خود یک عدد را ارایه می کند . خروجی این نرون درلایه آخر باید چیزی بین 0 که نمایانگر سرطان نوع "benign" و 1 که نشان دهنده سرطان نوع "malignant" خواهد بود. معماری شبکه گفته شده به مانند شکل زیر خواهد بود:



❖ نکته: شکل های نوع اتصال و قرار گیری نرون ها توسط پروژه متن باز dotnets رندر شده اند که در لینک https://github.com/martisak/dotnets قراردارد

کلاس پیاده سازی این شبکه عصبی به صورت زیر خواهد بود که وزن های بین نرون ها را در یک آرایه چند بعدی به اسم "weights" تعریف کرده ایم و همچنین از تابع محرک سیگموئید برای تمامی نرون ها استفاده کرده ایم. همچنین تمامی وزن ها به صورت تصادفی در سازنده این کلاس مقدار دهی می شوند.

```
WRITTEN BY MOHAMMAD ASADOLAHI
class NueralNetwork
    weights = []

    def __init__(self, layers)

    def sigmoid(self, number)

    def deriviateOfSigmoid(self, number)

    def printWeights(self)

    def compile(self, trainSamples, trainLabels, testSamples, testlabels,
epoches, learningRate=0.1)

    def predict(self, sample)

    def modelEvaluation(self, testSamples, testlabels)

    def feedForward(self, trainSample)

    def erroCalculator(self, netOutputs, desiredOutput)

    def backpropagate(self, input, desiredOutput, learningRate=0.1)
```

اجزای کلاس شبکه عصبی :

متغییر لیست چند بعدی **weights**: این متغیر وظیفه نگهداری وزن های بین نرون های شبکه عصبی را برعهده دارد.

__init__(self, layers) : این تابع که سازنده کلاس است تعداد لایه ها را به صورت یک لیست دریافت می کند و بر اساس تعداد گرفته شده اقدام به ایجاد لایه ها و همچنین مقدار دهی تصادفی بین لایه ها و نرون ها می کند در این پروژه این تابع مقدار متغیر **weights** را به صورت زیر مقدار دهی خواهد کرد:

```
WRITTEN BY MOHAMMAD ASADOLAHI
[array([[ 12.40376444,  82.28405409, -62.19286597,   3.18326553,
         -9.85446117, -50.47964307, 103.15399772, 180.93838596,
          10.14029747],
        [ 12.77256222,  82.55255044, -61.91126835,   2.68765257,
         -10.06369078, -50.4815444 , 102.94327616, 180.78192684,
          10.71957854],
        [ 12.78375102,  82.46410563, -62.82107112,   2.69580113,
          -9.97386456, -51.0719636 , 102.87698224, 181.22501082,
          10.00277228],
        [ 12.35248774,  82.33302778, -62.74739693,   2.45865952,
          -9.6971388 , -50.97908196, 103.44093765, 181.31320924,
           9.91778887],
        [ 12.53184963,  82.05125519, -62.58873587,   2.8604168 ,
          -9.41452307, -50.58717918, 103.01310549, 180.77389687,
          10.02115182],
        [ 12.00678516,  82.45476818, -62.22937257,   2.73678092,
         -10.15253909, -51.04482825, 103.11142479, 181.02049004,
           9.93956456],
        [ 12.8489854 ,  82.17569999, -62.72592933,   2.88137026,
          -9.44669232, -51.18200104, 103.25671771, 181.16490819,
          10.20306054],
        [ 12.45489192,  82.14440191, -62.57847831,   2.50510231,
          -9.50816758, -50.65406115, 103.32982156, 181.35383862,
          10.58959207],
        [ 12.17810223,  82.44514622, -62.80123727,   2.54738438,
          -9.39452101, -50.56188335, 103.37383924, 181.42745559,
          10.13049363]]), array([[ -4.62620424, -10.29166324,  25.24665129,
2.80969684,
         -1.06824238,  23.70465938,  -4.56248369, -17.28385876,
        -14.85397574]])]
```

که یک ارایه دارای دو زیر آرایه یکی با 9*9 متغیر (وزن بین لایه ورودی و 9 نرون لایه پنهان) و دیگری با 9*1 متغیر (وزن بین 9 نرون پنهان و تک نرون لایه خروجی) خواهد بود.

def sigmoid() : این تابع وظیفه پیاده سازی تابع سیگموئید که تابع محرک تمامی نرون های ما در این پروژه است را برعهده دارد.

def deriviateOfSigmoid() : این تابع مشتق تابع سیگموئید است که در گرادیان کاهشی از آن استفاده می کنیم.

def printWeights() : این تابع وظیفه چاپ متغیر weights را که شامل وزن بین نرون ها است را برعهده دارد.

def feedForward() : این تابع برای پیش خور کردن شبکه عصبی برای نمونه داد شده و محاسبه خروجی های هر نرون به ترتیب از لایه ورودی به سمت لایه خروجی پیاده سازی شده است.

def predict() : این تابع با فراخوانی تابع feedForward اقدام به پیش خور کردن شبکه عصبی برای نمو نه داده شده مقدار خروجی نرون لایه خروجی را انجام می دهد.

def erroCalculator() : این تابع برای یک نمونه داده شده اقدام به پیش خور کردن این نمونه با استفاده از تابع feedForward می کند و با استفاده از خروجی دریافت شده از این تابع و تحلیح خروجی هر نرون مقدار خطای هر نرون را محاسبه می کند.

def backpropagate() : این تابع با فراخوانی تابع feedForward و erroCalculator برای هر نمونه و با استفاده از مشتق تابع محرک اقدام به اصلاح وزن ها در صورت عدم تشخیص درست کلاس نمونه داده شده، می نماید.

def modelEvaluation() : این تابع با استفاده از محاسبه درصد داده های تست (رکورد هایی که در فرآیند آموزش شرکت داده نشده اند) درست تشخیص داده شده توسط تقسیم آنها بر کل داده های پیش بینی شده میزان کارایی مدل را در تشخیص درست داده ها بر حسب درصد مشخص می نماید.

def compile() : این تابع با استفاده از تابع backpropagate و modelEvaluation اقدام به آموزش شبکه عصبی بر روی تمامی داده های آموزشی در تعداد مشخص شده گام ها (epoches) می نماید و همچنین در آخر اقدام به رسم نمودار موفقیت مدل می کند.

جزییات پیاده سازی توابع گفته شده و کلاس شبکه عصبی در کد زیر تشریح شده است:

```
WRITTEN BY MOHAMMAD ASADOLAHI
class NueralNetwork:
    weights = []

    def __init__(self, layers):
        for item in range(1, len(layers)):
            self.weights.append(np.random.randn(layers[item], layers[item - 1]))
        print(len(self.weights))

    def sigmoid(self, number):
        return 1 / (1 + np.exp(-number))

    def deriviateOfSigmoid(self, number):
        return number * (1 - number)

    def printWeights(self):
        print(self.weights)

    def compile(self, trainSamples, trainLabels, testSamples, testlabels,
epoches, learningRate=0.1):
        modelAccuracy = []
        for epoch in range(epoches):
            for item in range(len(trainSamples)):
                self.backpropagate(trainSamples[item], trainLabels[item],
learningRate)
            modelAccuracy.append(self.modelEvaluation(testSamples, testlabels))
            # if (epoch % (epoches / 10)) == 0:
            #     print(f"epoch:{epoch} accuracy is{modelAccuracy[epoch]}")
            print(f"epoch: {epoch + 1} accuracy is {modelAccuracy[epoch - 1]}")
        plt.plot([x for x in modelAccuracy], label="Model Accuracy over epoches")
```

```
        plt.xlabel('x - Epoche')
        plt.ylabel('y - Accuracy ')
        plt.title('Model Accuracy over epoches')
        plt.legend()
        plt.show()

    def predict(self, sample):
        return self.feedForward(sample)[1][0]

    def modelEvaluation(self, testSamples, testlabels):
        correct = 0
        for index in range(len(testSamples)):
            prediction = self.predict(testSamples[index])
            if (prediction > 0.5 and testlabels[index] == 1) or (prediction <=
0.5 and testlabels[index] == 0):
                correct += 1
        return correct / len(testSamples)

    def feedForward(self, trainSample):
        input = np.copy(trainSample)
        outputs = []
        output = []
        for layer in self.weights:
            for node in layer:
                output.append(self.sigmoid(input.dot(node)))
            input = np.copy(output)
            outputs.append(input)
            output.clear()
        return outputs

    def erroCalculator(self, netOutputs, desiredOutput):
        outputError = (desiredOutput - netOutputs[len(self.weights) - 1]) *
self.deriviateOfSigmoid(
            netOutputs[len(self.weights) - 1])
        hiddenLayerError = []
        for item in range(len(self.weights[0])):
            hiddenLayerError.append(
                (outputError[0] * self.weights[1][0][item]) *
self.deriviateOfSigmoid(netOutputs[0][item]))
        return [np.array(hiddenLayerError), outputError]

    def backpropagate(self, input, desiredOutput, learningRate=0.1):
        netOutputs = self.feedForward(input)
        netErros = self.erroCalculator(netOutputs, desiredOutput)
        for bridge in range(9):
            for line in range(9):
                self.weights[0][bridge][line] = self.weights[0][bridge][line] + (
                        learningRate * ((input[line]) * netErros[0][line]))
        for line in range(9):
            self.weights[1][0][line] = self.weights[1][0][line] + ((learningRate
* (netOutputs[0][line])) * netErros[1])
```

با استفاده از کد زیر اقدام به ساخت شبکه عصبی با 9 نرون خروجی و یک لایه پنهان دارای 9 نرون و یک نرون در لایه خروجی می کنیم:

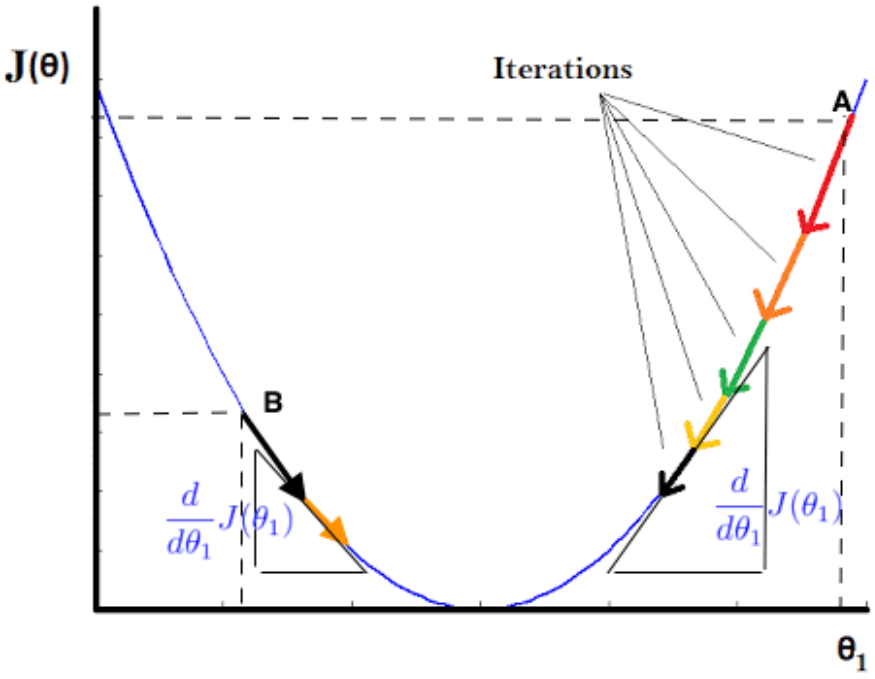
```
WRITTEN BY MOHAMMAD ASADOLAHI
# define NN model with 9 input neurons and 9 dense neurons in hidden layer
and 1 neuron in output layer
nueralNetwork = NueralNetwork([9, 9, 1])
```

روش آموزش :

روش آموزش مدل در این پروژه که مدل یک پرسپترون چندلایه است استفاده از الگوریتم پس انتشار خطا یا Error Backpropagation که روشی برای کاهش گرادیان کاهشی جهت حداقل کردن کل مربعات خطا خروجی است که تعمیمی از قانون دلتا برای آموزش شبکه های پرسپترون است. هنگامی که در شبکه های پرسپترون چند لایه FeedFoorward خطایی در خروجی رخ دهد این خطا باید به سمت ورودی برگشت داده شود و باعث اصلاح وزن ها شود.

گرادیان کاهشی (Gradient Descent) یک الگوریتم بهینه‌سازی برای پیدا کردن کمینه یک تابع است. در این الگوریتم کار با یک نقطه تصادفی روی تابع آغاز می‌شود و روی جهت منفی از گرادیان تابع حرکت می‌کند تا به کمینه محلی/سراسری برسد.

همان‌طور که می‌دانید مشتق، نشان‌دهنده‌ی شیبِ خطِ مماس بر یک نقطه از یک تابع است .برای اینکه کمترین میزانِ خطا را به دست آوریم فرض می‌کنیم یک نقطه‌ی دلخواه (یک وزن دلخواه) را در این تابع در نظر گرفته‌ایم. مثلاً نقطه‌ی Aدر شکل زیر

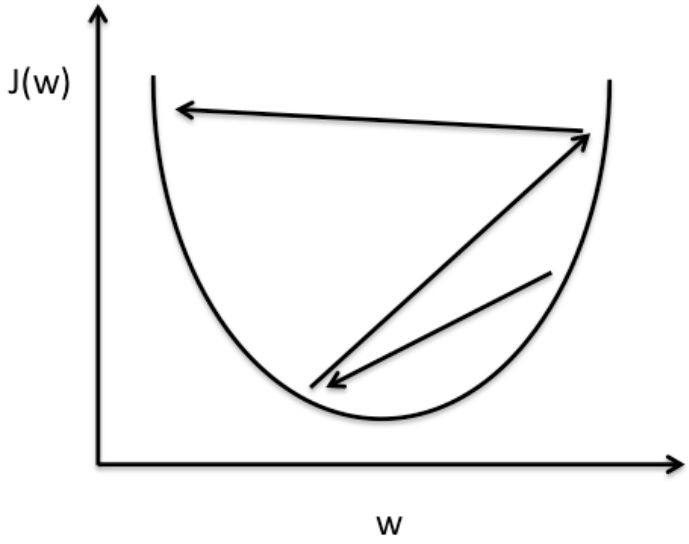


در این نقطه مشتق که همان شیب خطِ مماس بر یک نقطه است یک عدد منفی بوده، چون خط به سمت پایین است .الگوریتمِ پس انتشار می‌داند که اگر شیبِ خط در یک نقطه (با توجه به وزن‌ها) منفی بود بایستی مقدار آن وزن را افزایش دهد تا شیب خط به صفر برسد .شیب صفر یعنی کمترین میزان خطای ممکن در آن محدوده. همان‌طور که در شکلِ بالا مشخص است، کمترین میزانِ خطای ممکن برابر به صفر خواهد بود که در قسمت دره تابع واع شده است، با مثبت بودنِ شیب خط، یعنی همان مشتق در آن نقطه، الگوریتمِ پس انتشار می‌فهمد که باید وزن را کم کند تا شیب به صفر برسد.

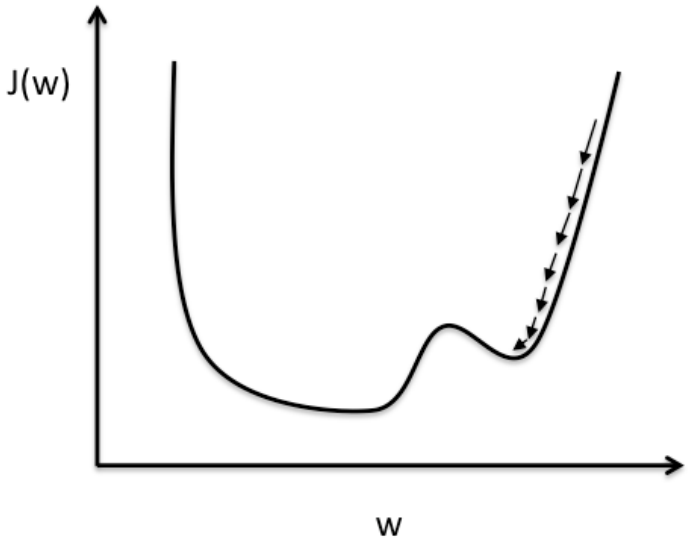
همان طور که در یک مثال بالا دیدید، الگوریتمِ پس انتشار می‌تواند با استفاده از این این تکنیک یک نقطه‌ی کمینه برای خطا پیدا کند که البته کمترین مقدار در کل فضا نبوده (الگوریتم گرادیان کاهشی میتواند در کمینه محلی گیر کند) ولی به هر حال معقول به نظر می‌رسید .به این نقطه‌ی معقول یک کمینه‌ی محلی (**local minimum**) برای خطا می‌گویند .در شکل بالا نقطه ضفر یک کمینه‌ی سراسری، یعنی بهترین نقطه موجود در کل شکل (**global minimum**) است. البته رسیدن به این نقطه‌ی سراسری برای الگوریتمِ پس انتشارِ خطا کار دشوار و زمان‌بری است.

برای همین معمولاً الگوریتم در شبکه‌های عصبی اینگونه آموزش می‌بیند که به تعداد تکرار مشخص یا تا رسیدن به یک خطای کمِ مشخص الگوریتم را ادامه بدهد و بعد از آن توقف کند. یعنی شبکه عصبی آنقدر تکرار را انجام می‌دهد تا به یک خطای معقول مشخصِ کم برسد . مثلاً در مثالِ بالا می‌گوییم اگر خطا زیر نقطه B شد دیگر کافی است. اگر این طور نشد یعنی خطا به اندازه‌ی دلخواهِ ما کم نشده است و حالا می‌توانیم برای تکرار محدودیت بگذاریم. مثلاً می‌گوییم تا ۱۰ هزار مرتبه تکرار را انجام بده (یعنی ۱۰ هزار مرتبه وزن‌ها و انحراف را آپدیت کن) و بعد از آن دیگر یادگیری را ادامه نده.

به طور خلاصه در گرادیان کاهشی با استفاده از مشتق تابع که به سمت شیب منفی تابع سیر میکند و تکرار این مرحله هر بار نسبت به کمینه محلی یا سراسری تابع حرکت میکنیم. همچنین سرعت حرکت را پارامتری به نام نرخ یادگیری تعیین می کند. که باید به درستی تعیین شود. تعیین نرخ یادگیری بالا ممکن است باعث **overshooting** یا دور شدن از کمینه و حرکت به سمت بیشینه تابع گردد و از سوی دیگر تعیین مقدار یادگیری پایین باعث کند شدن بیش از حد الگوریتم و حتی گیر کردن در کمینه محلی خواهد شد در شکل زیر این دو مشکل نشان داده شده اند. از این رو بسیاری از روش های جدید برای غلبه بر این مشکلات همانند الگوریتم **"ADAM"** با نرخ یادگیری تنظیم شونده ارایه شده اند که توضیح و استفاده از آنها خارج از حوصله این مقاله است.



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

برای آموزش شبکه عصبی پرسپترون در مرحله اول با استفاده از ویژگی های هر نمونه در دیتاست شبکه عصبی را تغذیه می کنیم و خروجی تمامی نرون ها را لایه به لایه محاسبه میکنیم. پس از محاسبه خروجی شبکه اختلاف بین خروجی شبکه و خروجی دلخواه یا واقعی را محاسبه می کنیم.

مثلا آیا شبکه به درستی بیماری را طبقه بندی کرده است یا خیر در اینجا ممکن است که خطا صفر باشد و بیماری بدرستی طبقه بندی شود که خطای شبکه صفر خواهد بود که در این صورت نیازی به تغییر وزن ها وجود ندارد و به سراغ نمونه بعدی در دیتا بیس می رویم. اما اگر خروجی شبکه با خروجی مورد نظر اختلاف داشت میزان خطا را با استفاده از فرمول های زیر محاسبه می کنیم:

میزان خطا برای نرون های خروجی به صورت $e=(t-y)f'(y)$ می باشد که $f'(y)$ مشتق تابع محرک خواهد بود لذا اگر تابع محرک را سیگموئید در نظر بگیریم میزان خطا به صورت زیر محاسبه خواهد شد:

$$E=(t-Y)Y(1-Y)$$

t برابر با خروجی مورد نظر برای نمونه پیش خور شده و Y خروجی شبکه خواهد بود.

میزان خطا برای نرون های لایه پنهان به صورت $e=(e'.w)h'$ می باشد که e' یعنی خطای نرون بعدی و w وزن بین نرون بعدی و نرون فعلی است و همچنین h نیز خروجی نرون فعلی خواهد بود که h' مشتق تابع محرک خواهد بود لذا اگر تابع محرک را سیگموئید در نظر بگیریم میزان خطا به صورت زیر محاسبه خواهد شد:

$$e=(e'.w)h(1-h)$$

بعد از محاسبه خطای هر نرون بر اساس خروجی نرون باید وزن های شبکه به صورت زیر بر اساس خطای نرون بروزرسانی شود

$$Wij \leftarrow Wij + \alpha * ej * oi$$

همچنین بایاس ها در صورت وجود باید به صورت زیر بروز رسانی شوند.

$$Bi \leftarrow bi + \alpha * ei$$

البته در این پروژه از بایاس استفاده نشده است و نرون ها دارای هیچ گونه بایاسی نیستند.

مراحل آموزش:

برای استفاده از الگوریتم **Backpropagation** برای آموزش شبکه عصبی پرسپترون در این پروژه توسط داده های دیتاست به ترتیب مراحل زیر را برای هر بیمار طی میکنیم:

- مرحله اول) پیش خورد کردن ورودی **FeedFoorward**

این کار توسط تابع **feedForward()** به صورت زیر انجام می شود:

```
WRITTEN BY MOHAMMAD ASADOLAHI
def feedForward(self, trainSample):
    input = np.copy(trainSample)
    outputs = []
    output = []
    for layer in self.weights:
        for node in layer:
            output.append(self.sigmoid(input.dot(node)))
        input = np.copy(output)
        outputs.append(input)
        output.clear()
    return outputs
```

- مرحله دوم) محاسبه خطای مربوط به هر نرون

این کار توسط تابع **erroCalculator()** به صورت زیر انجام می شود:

```
WRITTEN BY MOHAMMAD ASADOLAHI
def erroCalculator(self, netOutputs, desiredOutput):
    outputError = (desiredOutput - netOutputs[len(self.weights) - 1]) *
self.deriviateOfSigmoid(
        netOutputs[len(self.weights) - 1])
    hiddenLayerError = []
    for item in range(len(self.weights[0])):
        hiddenLayerError.append(
            (outputError[0] * self.weights[1][0][item]) *
self.deriviateOfSigmoid(netOutputs[0][item]))
    return [np.array(hiddenLayerError), outputError]
```

- **مرحله سوم) تنظیم وزن ها بر اساس خطای رخ داده**

این کار توسط تابع backpropagate() به صورت زیر انجام می شود. اگر نوع بیماری به صورت صحیح پیش بینی شده باشد نیازی به بروز کردن وزن ها نداریم اما اگر خروجی شبکه عصبی با کلاس واقعی سمپل پیش خور شده مغایرت داشته باشد بر اساس میزان خطای هر نرون و خروجی هر نرون با استفاده از مشتق تابع سیگموئید اقدام به بروز رسانی بردار وزن ها میکنیم(مقدار پیش فرض ضریب یادگیری برای اصلاح وزن ها مقدار 0.1 در نظر گرفته شده است):

```
WRITTEN BY MOHAMMAD ASADOLAHI
def backpropagate(self, input, desiredOutput, learningRate=0.1):
    netOutputs = self.feedForward(input)
    netErros = self.erroCalculator(netOutputs, desiredOutput)
    for bridge in range(9):
        for line in range(9):
            self.weights[0][bridge][line] = self.weights[0][bridge][line]
+ (
                    learningRate * ((input[line]) * netErros[0][line]))
    for line in range(9):
        self.weights[1][0][line] = self.weights[1][0][line] +
((learningRate * (netOutputs[0][line])) * netErros[1])
```

با استفاده از تابع compile() در تعداد گام مشخصی مثلا 100 گام عمل backpropagate را برای تمامی داده های دیتاست تکرار می کنیم . همچنین بعد از هر گام که تمامی داده های دیتاست مورد استفاده قرار گرفت اقدام به ارزیابی مدل بر اساس تقسیم بیمار های درست طبقه بندی شده بر کل بیمار ها، می کنیم. سپس پس از انجام تمامی گام ها نموداری را که مقدار کارایی مدل در هر گام را نشان می دهد را رسم می کنیم.

```
WRITTEN BY MOHAMMAD ASADOLAHI
def compile(self, trainSamples, trainLabels, testSamples, testlabels,
epoches, learningRate=0.1):
    modelAccuracy=[]
    for epoch in range(epoches):
        for item in range(len(trainSamples)):
            self.backpropagate(trainSamples[item], trainLabels[item],
learningRate)
        modelAccuracy.append(self.modelEvaluation(testSamples,
testlabels))
        # if (epoch % (epoches / 10)) == 0:
        #     print(f"epoch:{epoch} accuracy is{modelAccuracy[epoch]}")
        print(f"epoch:{epoch+1} accuracy is{modelAccuracy[epoch-1]}")
    plt.plot([x for x in modelAccuracy], label="Model Accuracy over
epoches")
    plt.xlabel('x - Epoche')
    plt.ylabel('y - Accuracy ')
    plt.title('Model Accuracy over epoches')
    plt.legend()
    plt.show()
```

نتیایج آموزش:

با فراخوانی تابع compile() و مشخص کردن بیمار ها و نوع بیماری آنها که 450 مورد از کل بیمار های دیتاست را تشکیل می دادند و از قبل برای مرحله آموزش جدا شده بودند و همچنین تعداد گام 100 اقدام به اموزش شبکه عصبی می کنیم:

البته توجه شود که بیمارهایی که قبلا برای تست کارایی شبکه عصبی در پیش بینی بیماری کنار گذاشته شده بودند (شامل 250 بیمار از کل بیمار های دیتاست) را نیز به این تابع می دهیم. البته این بیماران در فرآیند آموزش شبکه عصبی شرکت نمیکنند و فقط برای ارزیابی کارایی شبکه عصبی در پیش بینی بیماری برای رکورد هایی که تا کنون شبکه عصبی با آنها مواجه نشده است، مورد استفاده قرار می گیرند.

```
WRITTEN BY MOHAMMAD ASADOLAHI
# train the model with train samples and evaluate the model with test
labels at each step
nueralNetwork.compile(trainSamples, trainLabels, testSamples, testlabels,
100, 0.01)
```

پس از اجرای کد فوق فرآیند اموزش شروع شده و کارایی این شبکه عصبی در پیش بینی بیماری بیمارانی که تا کنون شبکه عصبی با آنها مواجه نشده است در هر گام مشخص می شود:

epoch: 1 accuracy is 0.3413654618473896

epoch: 2 accuracy is 0.3413654618473896

epoch: 3 accuracy is 0.7068273092369478

epoch: 4 accuracy is 0.751004016064257

epoch: 5 accuracy is 0.7630522088353414

epoch: 6 accuracy is 0.7791164658634538

epoch: 7 accuracy is 0.7791164658634538

epoch: 8 accuracy is 0.7871485943775101

epoch: 9 accuracy is 0.7951807228915663

epoch: 10 accuracy is 0.7951807228915663

epoch: 11 accuracy is 0.8032128514056225

epoch: 12 accuracy is 0.8032128514056225

epoch: 13 accuracy is 0.8032128514056225

epoch: 14 accuracy is 0.8032128514056225

epoch: 15 accuracy is 0.8112449799196787

epoch: 16 accuracy is 0.8112449799196787

epoch: 17 accuracy is 0.8112449799196787

epoch: 18 accuracy is 0.8152610441767069

epoch: 19 accuracy is 0.8192771084337349

epoch: 20 accuracy is 0.8273092369477911

epoch: 21 accuracy is 0.8313253012048193

epoch: 22 accuracy is 0.8353413654618473

epoch: 23 accuracy is 0.8393574297188755

epoch: 24 accuracy is 0.8433734939759037

epoch: 25 accuracy is 0.8514056224899599

epoch: 26 accuracy is 0.8554216867469879

epoch: 27 accuracy is 0.8634538152610441

epoch: 28 accuracy is 0.8714859437751004

epoch: 29 accuracy is 0.8795180722891566

epoch: 30 accuracy is 0.8835341365461847

epoch: 31 accuracy is 0.8875502008032129

epoch: 32 accuracy is 0.891566265060241

epoch: 33 accuracy is 0.891566265060241

epoch: 34 accuracy is 0.891566265060241

epoch: 35 accuracy is 0.891566265060241

epoch: 36 accuracy is 0.891566265060241

epoch: 37 accuracy is 0.8955823293172691

epoch: 38 accuracy is 0.8955823293172691

epoch: 39 accuracy is 0.8995983935742972

epoch: 40 accuracy is 0.9036144578313253

epoch: 41 accuracy is 0.9036144578313253

epoch: 42 accuracy is 0.9116465863453815

epoch: 43 accuracy is 0.9156626506024096

epoch: 44 accuracy is 0.9156626506024096

epoch: 45 accuracy is 0.9317269076305221

epoch: 46 accuracy is 0.9477911646586346

epoch: 47 accuracy is 0.9477911646586346

epoch: 48 accuracy is 0.9518072289156626

epoch: 49 accuracy is 0.9518072289156626

epoch: 50 accuracy is 0.9598393574297188

epoch: 51 accuracy is 0.963855421686747

epoch: 52 accuracy is 0.963855421686747

epoch: 53 accuracy is 0.963855421686747

epoch: 54 accuracy is 0.9678714859437751

epoch: 55 accuracy is 0.9678714859437751

epoch: 56 accuracy is 0.9678714859437751

epoch: 57 accuracy is 0.9678714859437751

epoch: 58 accuracy is 0.9678714859437751

epoch: 59 accuracy is 0.9718875502008032

epoch: 60 accuracy is 0.9718875502008032

epoch: 61 accuracy is 0.9718875502008032

epoch: 62 accuracy is 0.9718875502008032

epoch: 63 accuracy is 0.9759036144578314

epoch: 64 accuracy is 0.9759036144578314

epoch: 65 accuracy is 0.9759036144578314

epoch: 66 accuracy is 0.9759036144578314

epoch: 67 accuracy is 0.9759036144578314

epoch: 68 accuracy is 0.9759036144578314

epoch: 69 accuracy is 0.9759036144578314

epoch: 70 accuracy is 0.9718875502008032

epoch: 71 accuracy is 0.9718875502008032

epoch: 72 accuracy is 0.9718875502008032

epoch: 73 accuracy is 0.9759036144578314

epoch: 74 accuracy is 0.9759036144578314

epoch: 75 accuracy is 0.9759036144578314

epoch: 76 accuracy is 0.9759036144578314

epoch: 77 accuracy is 0.9759036144578314

epoch: 78 accuracy is 0.9759036144578314

epoch: 79 accuracy is 0.9759036144578314

epoch: 80 accuracy is 0.9759036144578314

epoch: 81 accuracy is 0.9759036144578314

epoch: 82 accuracy is 0.9759036144578314

epoch: 83 accuracy is 0.9759036144578314

epoch: 84 accuracy is 0.9759036144578314

epoch: 85 accuracy is 0.9759036144578314

epoch: 86 accuracy is 0.9759036144578314

epoch: 87 accuracy is 0.9759036144578314

epoch: 88 accuracy is 0.9718875502008032

epoch: 89 accuracy is 0.9718875502008032

epoch: 90 accuracy is 0.9718875502008032

epoch: 91 accuracy is 0.9718875502008032

epoch: 92 accuracy is 0.9718875502008032

epoch: 93 accuracy is 0.9718875502008032

epoch: 94 accuracy is 0.9718875502008032

epoch: 95 accuracy is 0.9718875502008032

epoch: 96 accuracy is 0.9718875502008032

epoch: 97 accuracy is 0.9718875502008032

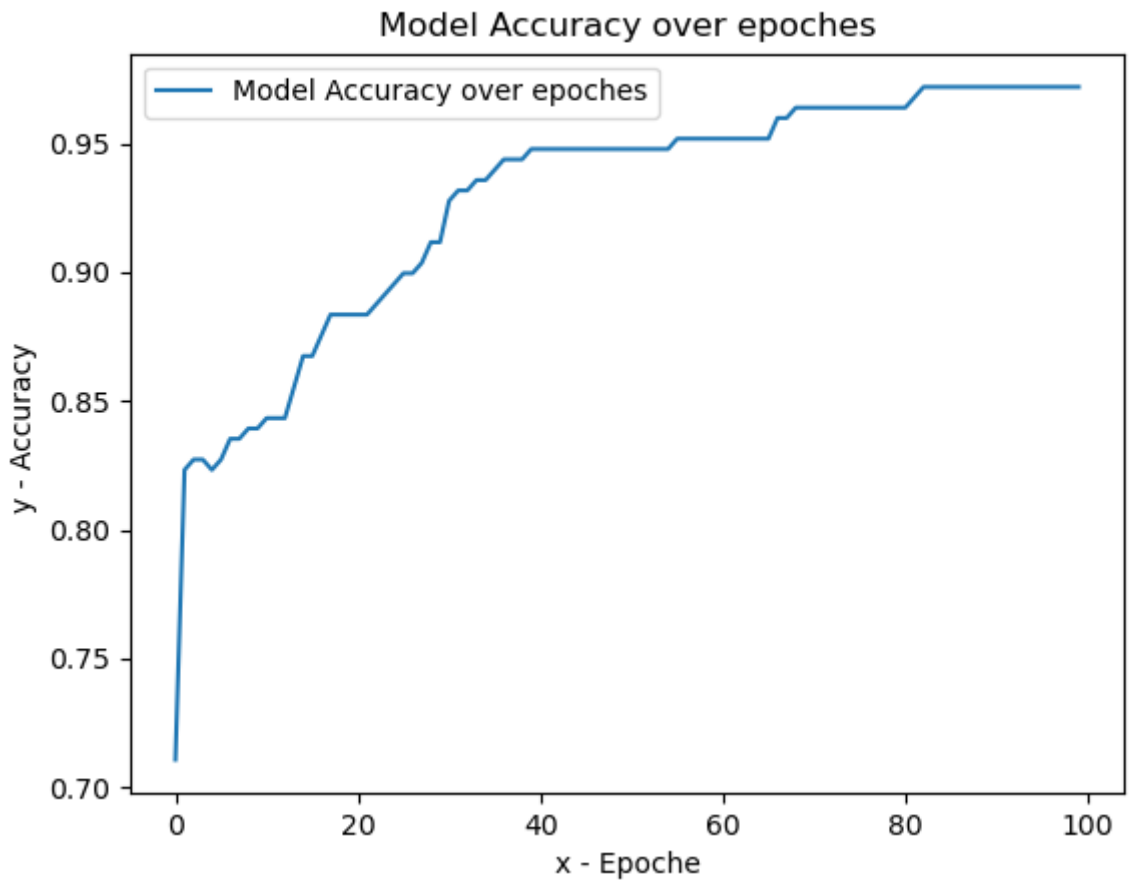
epoch: 98 accuracy is 0.9718875502008032

epoch: 99 accuracy is 0.9718875502008032

epoch: 100 accuracy is 0.9718875502008032

نمودار پیشرفت مدل:

پس از پایان فرآیند آموزش اقدام به رسم نمودار کارایی شبکه عصبی استفاده شده برای پیش بینی بیماری در هر گام می کنیم:



پیش بینی با استفاده از MLP

و با استفاده از کد زیر اقدام به پیش بینی بیماری برای بیمارانی که از قبل برای تست شبکه عصبی ما کنار گذاشته شده بودند می کنیم:

```
WRITTEN BY MOHAMMAD ASADOLAHI
#using our MLP to predict class of our test samples
for index in range(len(testSamples)):
    prediction=nueralNetwork.predict(testSamples[index])
    if(prediction>0.5):
        prediction=1
    else:
        prediction=0
    print(f"sample: {testSamples[index]}  predicted class: {prediction}
real calss: {testlabels[index]}")
```

خروجی کد بالا برای اجرایی که در این مقاله جزییات آن آورده شده است به صورت زیر خواهد بود:

sample: [0.4 0.1 0.1 0.3 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.5 0.5 0.8 0.6 1. 1. 0.7 0.1] predicted class: 1 real calss: 1.0

sample: [0.2 0.3 0.1 0.1 0.3 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [1. 0.2 0.2 0.1 0.2 0.6 0.1 0.1 0.2] predicted class: 1 real calss: 1.0

sample: [1. 0.6 0.5 0.8 0.5 1. 0.8 0.6 0.1] predicted class: 1 real calss: 1.0

sample: [0.8 0.8 0.9 0.6 0.6 0.3 1. 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.2 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.3 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.5 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.2 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [1. 0.9 0.8 0.7 0.6 0.4 0.7 1. 0.3] predicted class: 1 real calss: 1.0

sample: [1. 0.6 0.6 0.2 0.4 1. 0.9 0.7 0.1] predicted class: 1 real calss: 1.0

sample: [0.6 0.6 0.6 0.5 0.4 1. 0.7 0.6 0.2] predicted class: 1 real calss: 1.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.2 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.2 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.2 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.8 0.7 1. 0.4 1. 0.7 0.5 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.3 0.2 0.4 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.9 1. 1. 1. 1. 0.5 1. 1. 1.] predicted class: 1 real calss: 1.0

sample: [0.8 0.7 0.8 0.5 0.5 1. 0.9 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.2 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.3 0.1 0.3 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [1. 1. 1. 1. 0.6 1. 0.8 0.1 0.5] predicted class: 1 real calss: 1.0

sample: [0.3 0.6 0.4 1. 0.3 0.3 0.3 0.4 0.1] predicted class: 0 real calss: 1.0

sample: [0.6 0.3 0.2 0.1 0.3 0.4 0.4 0.1 0.1] predicted class: 1 real calss: 1.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.8 0.9 0.4 0.3 1. 0.7 0.1 0.1] predicted class: 1 real calss: 1.0

sample: [0.4 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 1. 1. 1. 0.6 1. 0.6 0.5 0.2] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.2 1. 0.4 0.5 0.2 0.1 0.1] predicted class: 1 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.2 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.2 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.3 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.8 1. 1. 1. 0.7 0.5 0.4 0.8 0.7] predicted class: 1 real calss: 1.0

sample: [0.1 0.1 0.1 0.1 0.2 0.4 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.6 0.7 1. 0.3 1. 0.8 1. 0.2] predicted class: 1 real calss: 1.0

sample: [0.4 1. 0.4 0.7 0.3 1. 0.9 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.2 0.2 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.7 0.8 0.3 0.4 1. 0.9 0.1 0.1] predicted class: 1 real calss: 1.0

sample: [0.1 0.1 0.1 0.1 0.3 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.3 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [1. 0.4 0.5 0.4 0.3 0.5 0.7 0.3 0.1] predicted class: 1 real calss: 1.0

sample: [0.7 0.5 0.6 1. 0.4 1. 0.5 0.3 0.1] predicted class: 1 real calss: 1.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.2 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.1 0.3 0.2 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.7 0.4 0.4 0.3 0.4 1. 0.6 0.9 0.1] predicted class: 1 real calss: 1.0

sample: [0.4 0.2 0.2 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.1 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.3 0.2 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.2 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.2 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.3 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.3 0.2 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 1. 1. 1. 0.4 1. 0.7 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.2 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.7 0.8 0.3 0.7 0.4 0.5 0.7 0.8 0.2] predicted class: 1 real calss: 1.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.2 0.2 0.2 0.2 0.1 0.4 0.2 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.4 0.2 0.1 0.2 0.5 0.2 0.1 0.2] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.3 0.1 0.1 0.2 0.1 0.4 0.8 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.2 0.2 0.2 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.3 0.2 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.7 1. 1. 0.5 1. 1. 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.3 0.1 0.2 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.3 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.8 0.4 0.4 0.1 0.6 1. 0.2 0.5 0.2] predicted class: 1 real calss: 1.0

sample: [1. 1. 0.8 1. 0.6 0.5 1. 0.3 0.1] predicted class: 1 real calss: 1.0

sample: [0.8 1. 0.4 0.4 0.8 1. 0.8 0.2 0.1] predicted class: 1 real calss: 1.0

sample: [0.7 0.6 1. 0.5 0.3 1. 0.9 1. 0.2] predicted class: 1 real calss: 1.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [1. 0.9 0.7 0.3 0.4 0.2 0.7 0.7 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.2 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.2 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.7 1. 0.6 0.5 1. 0.7 0.5 0.1] predicted class: 1 real calss: 1.0

sample: [0.6 1. 0.5 0.5 0.4 1. 0.6 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.6 0.3 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.8 1. 1. 1. 0.6 1. 1. 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.2 0.1] predicted class: 0 real calss: 0.0

sample: [0.9 0.8 0.8 0.9 0.6 0.3 0.4 0.1 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 1. 0.8 0.5 0.4 0.1 1. 0.1 0.1] predicted class: 0 real calss: 1.0

sample: [0.2 0.5 0.7 0.6 0.4 1. 0.7 0.6 0.1] predicted class: 1 real calss: 1.0

sample: [1. 0.3 0.4 0.5 0.3 1. 0.4 0.1 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.2 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.8 0.6 0.3 0.4 1. 0.7 0.1 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.2 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.3 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.2 0.4 0.1 0.1 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.4 0.6 0.8 0.4 0.1 0.8 1. 0.1] predicted class: 0 real calss: 1.0

sample: [0.5 0.3 0.2 0.8 0.5 1. 0.8 0.1 0.2] predicted class: 1 real calss: 1.0

sample: [1. 0.5 1. 0.3 0.5 0.8 0.7 0.8 0.3] predicted class: 1 real calss: 1.0

sample: [0.4 0.1 0.1 0.2 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 1. 1. 1. 1. 1. 1. 0.1 0.1] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [1. 0.4 0.3 1. 0.3 1. 0.7 0.1 0.2] predicted class: 1 real calss: 1.0

sample: [0.5 1. 1. 1. 0.5 0.2 0.8 0.5 0.1] predicted class: 0 real calss: 1.0

sample: [0.8 1. 1. 1. 0.6 1. 1. 1. 1.] predicted class: 1 real calss: 1.0

sample: [0.2 0.3 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.3 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.1 0.3 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.3 0.3 0.3 0.3 0.2 0.6 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.7 0.1 0.2 0.3 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.2 0.1 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.3 0.1 0.3 0.4 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.6 0.6 0.5 0.7 0.6 0.7 0.7 0.3] predicted class: 1 real calss: 1.0

sample: [0.2 0.1 0.1 0.1 0.2 0.5 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.6 0.2 0.3 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.8 0.7 0.4 0.4 0.5 0.3 0.5 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.4 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [1. 1. 0.7 0.8 0.7 0.1 1. 1. 0.3] predicted class: 1 real calss: 1.0

sample: [0.4 0.2 0.4 0.3 0.2 0.2 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.2 0.2 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 1. 1. 1. 1. 0.2 1. 1. 1.] predicted class: 1 real calss: 1.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.2 0.3 0.4 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.2 0.1 0.3 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.2 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.4 0.5 0.1 0.8 0.1 0.3 0.6 0.1] predicted class: 0 real calss: 0.0

sample: [0.7 0.8 0.8 0.7 0.3 1. 0.7 0.2 0.3] predicted class: 1 real calss: 1.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.3 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.3 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.3 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.2 0.2 0.2 0.2 0.1 0.1 0.1 0.2] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.7 0.4 0.1 0.6 0.1 0.7 1. 0.3] predicted class: 0 real calss: 1.0

sample: [0.5 1. 1. 0.8 0.5 0.5 0.7 1. 0.1] predicted class: 1 real calss: 1.0

sample: [0.3 1. 0.7 0.8 0.5 0.8 0.7 0.4 0.1] predicted class: 1 real calss: 1.0

sample: [0.3 0.2 0.1 0.2 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.2 0.1 0.3 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.3 0.2 0.1 0.3 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.4 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.2 0.1 0.2 0.1 0.2 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [1. 1. 1. 1. 0.5 1. 1. 1. 0.7] predicted class: 1 real calss: 1.0

sample: [0.5 1. 1. 1. 0.4 1. 0.5 0.6 0.3] predicted class: 1 real calss: 1.0

sample: [0.5 0.1 0.1 0.1 0.2 0.1 0.3 0.2 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.3 0.1] predicted class: 0 real calss: 0.0

sample: [0.4 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.8] predicted class: 0 real calss: 0.0

sample: [0.1 0.1 0.1 0.3 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 1. 1. 0.5 0.4 0.5 0.4 0.4 0.1] predicted class: 0 real calss: 1.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.2 0.1 0.2 0.1 0.2] predicted class: 0 real calss: 0.0

sample: [0.3 0.1 0.1 0.1 0.3 0.2 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.2 0.1 0.1 0.1 0.2 0.1 0.1 0.1 0.1] predicted class: 0 real calss: 0.0

sample: [0.5 1. 1. 0.3 0.7 0.3 0.8 1. 0.2] predicted class: 1 real calss: 1.0

sample: [0.4 0.8 0.6 0.4 0.3 0.4 1. 0.6 0.1] predicted class: 1 real calss: 1.0

sample: [0.4 0.8 0.8 0.5 0.4 0.5 1. 0.4 0.1] predicted class: 1 real calss: 1.0

نتیجه گیری:

در این پروژه تمامی گام های استاندارد مدل های ماشین یادگیر در پروژه های واقعی پیاده سازی شد. ابتدا داده ها مورد پیش پردازش قرار گرفت. و پس از آن یک مدل ماشین یادگیر تعریف شد، سپس از داده های پردازش شده برای آموزش مدل ما که در این مثلا یک شبکه چند لایه پرسپترون بود استفاده شد. همچنین در فرآیند آموزش تعدادی از داده ها برای ارزیابی پیشرفت مدل در پیش بینی کلاس داده های که شبکه عصبی تا کنون آنها را ندیده بود استفاده شد و میزان پیشرفت شبکه عصبی در هر دوره (epoch) آموزش داده های آموزشی تعیین گردید. در آخر نیز از شبکه عصبی برای پیش بینی کلاس داده هایی که برای ارزیابی مدل کنار گذاشته شده بود استفاده شد و مشخص شد که شبکه عصبی مورد استفاده به خوبی قادر به پیش بینی کلاس این بیماران بوده است.

استفاده از شبکه های عصبی برای عمل دسته بندی از دیر باز مورد توجه محققین قرار گرفته است. پتانسیل شبکه های عصبی برای دسته بندی داده های خطی و غیر خطی و فراگرفتن الگوهای بسیار پیچیده توسط شبکه های عصبی آنها را به یکی از بهترین گزینه ها برای انجام عمل دسته بندی (classification) بدل کرده است. همانطور که در این پروژه دیدیم داده های دیتاست مورد استفاده که دارای اطلاعات 699 بیمار سرطانی بود توسط شبکه عصبی به خوبی و با دقت بسیار بالایی (دقت 97 درصدی) بین دو نوع سرطان سینه نوع "malignant"و" benign" دسته بندی شدند. هرچند که مدل شبکه عصبی در این پروژه که دارای یک لایه مخفی دارای 9 نرون بود مدل نسبتا ساده ای در بین شبکه های عصبی پنداشته می شود اما به خوبی این شبکه عصبی توانست عمل دسته بندی بیماران را با دقت 97 درصدی انجام دهد. فارغ از استفاده از هر روشی برای دسته بندی به دلیل وجود نویز، داده های پرت و همچنین داده های ناموجود(missing values) هیچ گاه در عمل classification نمی توان دقت دسته بندی را به 100 درصد رساند.

کد تمامی پروژه به صورت زیر است:

```
WRITTEN BY MOHAMMAD ASADOLAHI
#Wisconsin Breast Cancer  MLP Classifier
#WRITTEN BY MOHAMMAD ASADOLAHI
# Mohamad.asa1994@gmail.com
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt


class NueralNetwork:
    weights = []

    def __init__(self, layers):
        for item in range(1, len(layers)):
            self.weights.append(np.random.randn(layers[item], layers[item - 1]))

    def sigmoid(self, number):
        return 1 / (1 + np.exp(-number))

    def deriviateOfSigmoid(self, number):
        return number * (1 - number)

    def printWeights(self):
        print(self.weights)

    def compile(self, trainSamples, trainLabels, testSamples, testlabels,
epoches, learningRate=0.1):
        modelAccuracy = []
        for epoch in range(epoches):
            for item in range(len(trainSamples)):
                self.backpropagate(trainSamples[item], trainLabels[item],
learningRate)
            modelAccuracy.append(self.modelEvaluation(testSamples, testlabels))
            # if (epoch % (epoches / 10)) == 0:
            #     print(f"epoch:{epoch} accuracy is{modelAccuracy[epoch]}")
            print(f"epoch: {epoch + 1} accuracy is {modelAccuracy[epoch - 1]}")
        plt.plot([x for x in modelAccuracy], label="Model Accuracy over epoches")
        plt.xlabel('x - Epoche')
        plt.ylabel('y - Accuracy ')
        plt.title('Model Accuracy over epoches')
        plt.legend()
        plt.show()

    def predict(self, sample):
        return self.feedForward(sample)[1][0]

    def modelEvaluation(self, testSamples, testlabels):
        correct = 0
        for index in range(len(testSamples)):
            prediction = self.predict(testSamples[index])
            if (prediction > 0.5 and testlabels[index] == 1) or (prediction <=
0.5 and testlabels[index] == 0):
                correct += 1
        return correct / len(testSamples)
```

```
    def feedForward(self, trainSample):
        input = np.copy(trainSample)
        outputs = []
        output = []
        for layer in self.weights:
            for node in layer:
                output.append(self.sigmoid(input.dot(node)))
            input = np.copy(output)
            outputs.append(input)
            output.clear()
        return outputs

    def erroCalculator(self, netOutputs, desiredOutput):
        outputError = (desiredOutput - netOutputs[len(self.weights) - 1]) *
self.deriviateOfSigmoid(
            netOutputs[len(self.weights) - 1])
        hiddenLayerError = []
        for item in range(len(self.weights[0])):
            hiddenLayerError.append(
                (outputError[0] * self.weights[1][0][item]) *
self.deriviateOfSigmoid(netOutputs[0][item]))
        return [np.array(hiddenLayerError), outputError]

    def backpropagate(self, input, desiredOutput, learningRate=0.1):
        netOutputs = self.feedForward(input)
        netErros = self.erroCalculator(netOutputs, desiredOutput)
        for bridge in range(9):
            for line in range(9):
                self.weights[0][bridge][line] = self.weights[0][bridge][line] + (
                        learningRate * ((input[line]) * netErros[0][line]))
        for line in range(9):
            self.weights[1][0][line] = self.weights[1][0][line] + ((learningRate
* (netOutputs[0][line])) * netErros[1])


# fetching records from storage and preprocessing----------------------------
-----------------
records = []
with open('./cancer.data') as f:
    for line in f.readlines():
        records.append(np.fromstring(line.rstrip('\n'), dtype=float, sep=","))
samples = pd.DataFrame(records, columns=["id", "Clump Thickness", "Cell Size
Uniformity", "Cell Shape Uniformity",
                                         "Marginal Adhesion", "Single Epithelial
Cell Size", "Bare Nuclei",
                                         "Bland Chromatin", "Normal Nucleoli",
"Mitoses", "class"])

print(samples.head(5))  # show top 5 fetch records

samples = samples.drop("id", axis=1)  # remove id witch is useless for
classification

labels = samples["class"]  # transfer labels to another dataframe
samples = samples.drop("class", axis=1)  # remove class from features vector
```

```
labels[labels == 2] = 0  # "replace 2 with 0 for benign cancer"

labels[labels == 4] = 1  # "replace 4 with 1 for malignant cancer"



# replacing missig values with the mean of column--------------------------------
------------------
for each in samples.columns:
    samples[each] = samples[each].replace(to_replace=0,
                                          value=int(samples[each].mean()))

# scaling all numeral features between [0 to 1] ----------------------------------
-------------------
for each in samples.columns:
    samples[each] = samples[each] / samples[each].max()

# split samples into train and test samples
trainSamples = (samples[0:450]).to_numpy()  # 450 sample to train the model
,almost 65% of all records
trainLabels = (labels[0:450]).to_numpy()

testSamples = (samples[450:699]).to_numpy()  # 250 sample to test the model
,almost 35% of all records
testlabels = (labels[450:699]).to_numpy()

# define NN model with 9 input neurons and 9 dense neurons in hidden layer and 1
neuron in output layer
nueralNetwork = NueralNetwork([9, 9, 1])

# train the model with train samples and evaluate the model with test labels at
each step
nueralNetwork.compile(trainSamples, trainLabels, testSamples, testlabels, 100,
0.1)

# using our MLP to predict class of our test samples
for index in range(len(testSamples)):
    prediction = nueralNetwork.predict(testSamples[index])
    if (prediction > 0.5):
        prediction = 1
    else:
        prediction = 0
    print(f"sample: {testSamples[index]}  predicted class: {prediction}  real
calss: {testlabels[index]}")
```