به نام خدا

سیستم‌های هوشمند (دکتر رشاد حسینی)

تمرین سری **اول**

نیم‌سال اول ۱۴۰۲-۱۴۰۱

عرفـان پـنـــاهی ۸۱۰۱۹۸۳۶۹

این فایل شامل گزارش و نتایج شبیه سازی های انجام شده است.

***** فایل شبیه سازی با پایتون مربوط به هر قسمت این تمرین با عنوان HW1_Q#x_810198369.ipynb پیوست شده است.**

سوال ۱: رگرسیون لجستیک باینری (لینک گزارش)

سوال ۲: بهینه سازی در توابع غیر محدب (لینک گزارش)

سوال ۳: ماشین بردار پشتیبان (لینک گزارش)

سوال ۱: رگرسیون لجستیک باینری

قسمت الف: محاسبه‌ی گرادیان تحلیلی

در این قسمت گرادیان تابع هزینه خواسته شده را برحسب $\mu(\omega, b)$ بدست می‌آوریم.

$$J(\omega, b) = \frac{1}{n}\sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i(b + x_i^T\omega)\right)\right)$$

$$\mu(\omega, b) = \frac{1}{1 + \exp\left(-y_i(b + x_i^T\omega)\right)}$$

$$(*)\ \nabla_b \mu(\omega, b) = \frac{y_i \exp\left(-y_i(b + x_i^T\omega)\right)}{\left(1 + \exp\left(-y_i(b + x_i^T\omega)\right)\right)^2}$$

$$\nabla_b J(\omega, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i \exp\left(-y_i(b + x_i^T\omega)\right)}{1 + \exp\left(-y_i(b + x_i^T\omega)\right)} = -\frac{1}{n}\sum_{i=1}^{n} \frac{\nabla_b \mu(\omega, b)}{\mu(\omega, b)}$$

البته می‌توانیم این گرادیان را برای ساده تر شدن پیاده سازی به صورت زیر نیز بنویسیم.

$$\nabla_b J(\omega, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i \exp\left(-y_i(b + x_i^T\omega)\right)}{1 + \exp\left(-y_i(b + x_i^T\omega)\right)} = -\frac{1}{n}\sum_{i=1}^{n} y_i \left(\frac{1}{\mu(\omega, b)} - 1\right)\mu(\omega, b)$$

$$= -\frac{1}{n}\sum_{i=1}^{n} y_i\left(1 - \mu(\omega, b)\right)$$

$$(*)\ \nabla_\omega \mu(\omega, b) = \frac{y_i x_i^T \exp\left(-y_i(b + x_i^T\omega)\right)}{\left(1 + \exp\left(-y_i(b + x_i^T\omega)\right)\right)^2}$$

$$\nabla_\omega J(\omega, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i x_i^T \exp\left(-y_i(b + x_i^T\omega)\right)}{1 + \exp\left(-y_i(b + x_i^T\omega)\right)} = -\frac{1}{n}\sum_{i=1}^{n} \frac{\nabla_\omega \mu(\omega, b)}{\mu(\omega, b)}$$

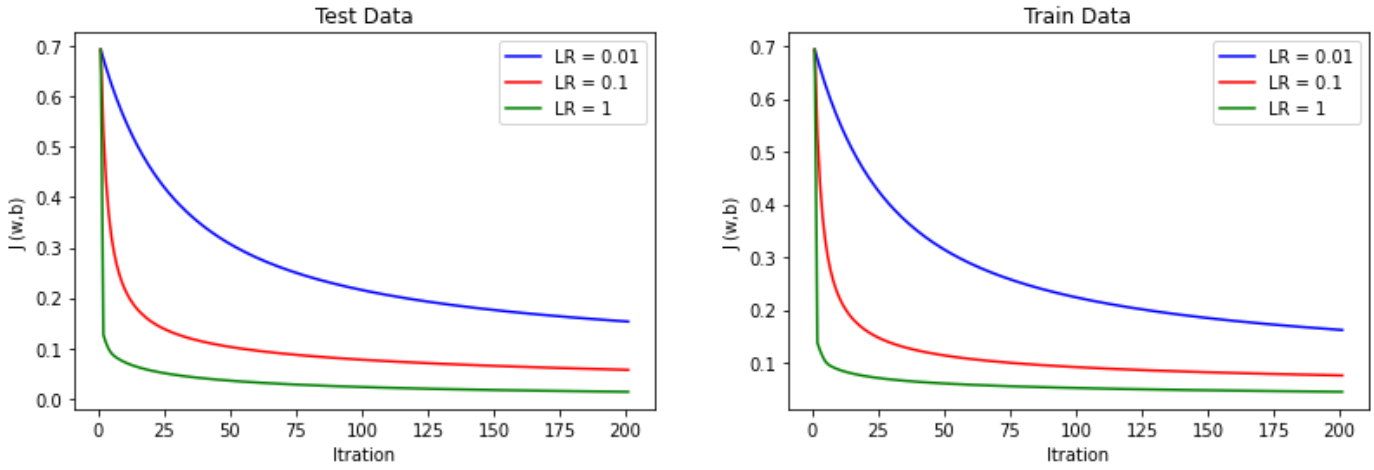
البته می‌توانیم این گرادیان را برای ساده تر شدن پیاده سازی به صورت زیر نیز بنویسیم.

$$\nabla_b J(\omega,b) = \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i x_i^T \exp\left(-y_i(b+x_i^T\omega)\right)}{1+\exp\left(-y_i(b+x_i^T\omega)\right)} = -\frac{1}{n}\sum_{i=1}^{n} y_i x_i^T\left(\frac{1}{\mu(\omega,b)}-1\right)\mu(\omega,b)$$
$$= -\frac{1}{n}\sum_{i=1}^{n} y_i x_i^T\left(1-\mu(\omega,b)\right)$$

قسمت ب: محاسبه‌ی گرادیان نزولی

در این قسمت با توجه به اینکه داده های test.csv ، label ندارند، تعدادی از داده های train را به عنوان داده test در نظر می‌گیریم. به ان منظور ابتدا سطر هایی که label آن‌ها اعدادی غیر از ۲ و ۷ است را حذف می‌کنیم. در نتیجه حدوداً ۸۵۰۰ داده دارای label داریم. از این مجموعه ۸۰۰۰ داده را به‌عنوان داده train و مابقی را به عنوان داده test در نظر می‌گیریم.

پیاده سازی روش گرادیان نزولی: حال روش گرادیان نزولی را با استفاده از گرادیان تابع هزینه بدست آمده در قسمت الف، با مقادیر اولیه صفر برای هر کدام از داده های train و test پیاده سازی می‌کنیم. برای هر کدام از مجموعه داده ها ۳ گام (Learning Rate) متفاوت در نظر می‌گیریم. تصاویر ۱-۱ نمودار مقادیر تابع هزینه برحسب تعداد تکرار را برای داده های train و test و با هر ۳ گام نشان می‌دهد.



تصویر ۱-۱: مقدار تابع هزینه به ازای تعداد تکرار برای ۳ گام مختلف

توضیحات در خصوص کد نوشته شده: برای بهینه سازی تابع هزینه، تابع opt_J نوشته شده است که در آن به تعداد دلخواه تکرار تا رسیدن به مقدار تابع هزینه کوچک وجود دارد. با توجه به اینکه ممکن است تابع مورد نظر convex نباشد نمی‌توانیم شرط توقف را روی گرادیان و یا افزایش مقدار تابع هزینه در نظر بگیریم. به این منظور تعداد تکرار را ثابت و علاوه بر آن یک شرط توقف برای اینکه اگر تابع هزینه از یک مقدار خاص کمتر شد، الگوریتم متوقف شود، در نظر می‌گیریم.

نتایج: همانطور که در تصویر ۱-۱ مشاهده می‌شود برای داده های train ماشین بهتر learn شده و مقدار تابع هزینه کمی نزولی‌تر (پایین‌تر) است. دلیل این امر بیشتر بودن تعداد داده های train است. همچنین مشاهده می‌شود که هر چه مقدار Learning Rate به ۱ نزدیک‌تر می‌شود، مقدار تابع هزینه با سرعت بیشتری کمینه می‌شود.

قسمت ج: محاسبه‌ی دقت

برای این قسمت، از b و ω تخمین زده شده با داده های train استفاده می‌کنیم و accuracy ماشین طراحی شده را برای داده های test محاسبه می‌کنیم. این کار را برای هر ۳ گام انجام می‌دهیم. تصویر ۱-۲ این مقدار را برای هر ۳ گام نشان می‌دهد.

```
Accuracy for test data (Machine is learned using train data):
  Learning Rate = 0.01 : 96.88581314879%
  Learning Rate = 0.1 : 98.0968858131499%
  Learning Rate = 1 : 98.44290657439558%
```

تصویر ۱-۲: مقدار accuracy روی داده های test برای هر ۳ گام (ماشین learn شده با داده های train)

نتایج: همانطور که در تصویر ۱-۲ مشاهده می‌شود برای به ازای Learning Rate های نزدیک به ۱ ماشین بهتر learn شده و accuracy روی داده های test بیشتر است.

همچنین درصد درستی ماشین های learn شده با داده های train و test را برای داده های آموزش خود به دست می‌آوریم. تصاویر ۱-۳ و ۱-۴ این مقادیر را نشان می‌دهد.

```
Accuracy for train data (Machine is learned using train data):
  Learning Rate = 0.01 : 96.60000000001145%
  Learning Rate = 0.1 : 97.70000000001181%
  Learning Rate = 1 : 98.5750000000121%
```

تصویر ۱-۳: مقدار accuracy روی داده های train برای هر ۳ گام (ماشین learn شده با داده های train)

```
Accuracy for test data (Machine is learned using test data):
  Learning Rate = 0.01 : 97.23183391003569%
  Learning Rate = 0.1 : 98.6159169550184%
  Learning Rate = 1 : 100.00000000000114%
```

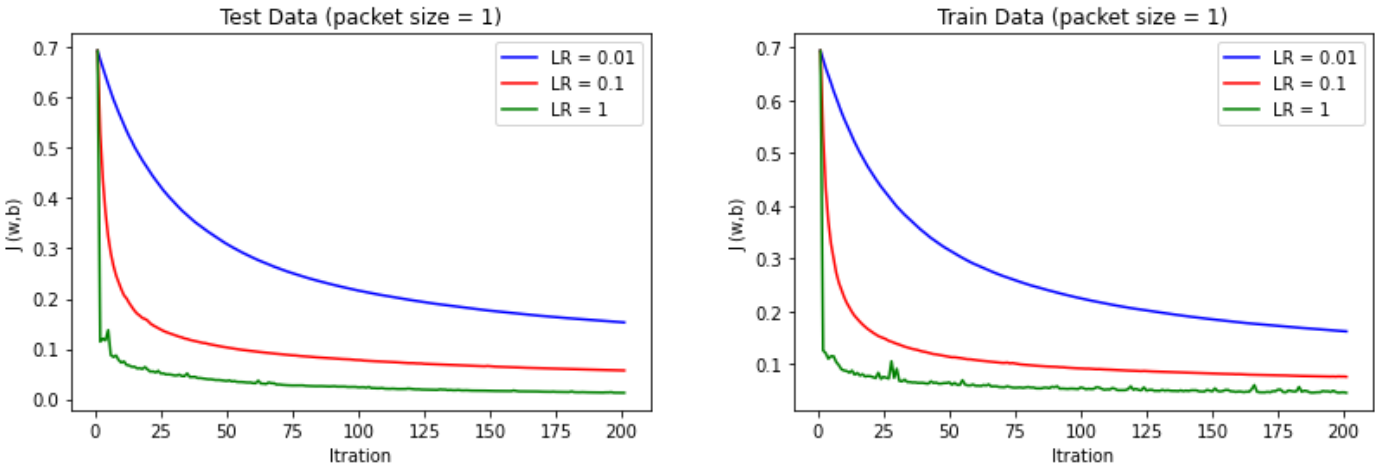
تصویر ۱-۴: مقدار accuracy روی داده های test برای هر ۳ گام (ماشین learn شده با داده های test)

قسمت د: گرادیان نزولی تصادفی

در این قسمت همان مراحل قسمت ب و ج را با یک تفاوت در الگوریتم انجام می‌دهیم. به این منظور در هر تکرار یک بسته به طول دلخواه و رندوم از کل داده‌های train انتخاب کرده و update کردن پارامتر های b و ω را با استفاده از آن جلو می‌بریم. برای دو طول بسته ۱ و ۱۰۰ این کار را انجام می‌دهیم و accuracy های قسمت ج را برای ماشین های طراحی شده جدید بدست می‌آوریم.

۱- طول بسته : ۱

تصویر ۱-۵، نمودار تابع هزینه به ازای تکرار برای طول بسته ۱ و گام های مختلف را نشان می‌دهد.



تصویر ۱-۵: مقدار تابع هزینه به ازای تعداد تکرار برای ۳ گام مختلف، طول بسته ۱

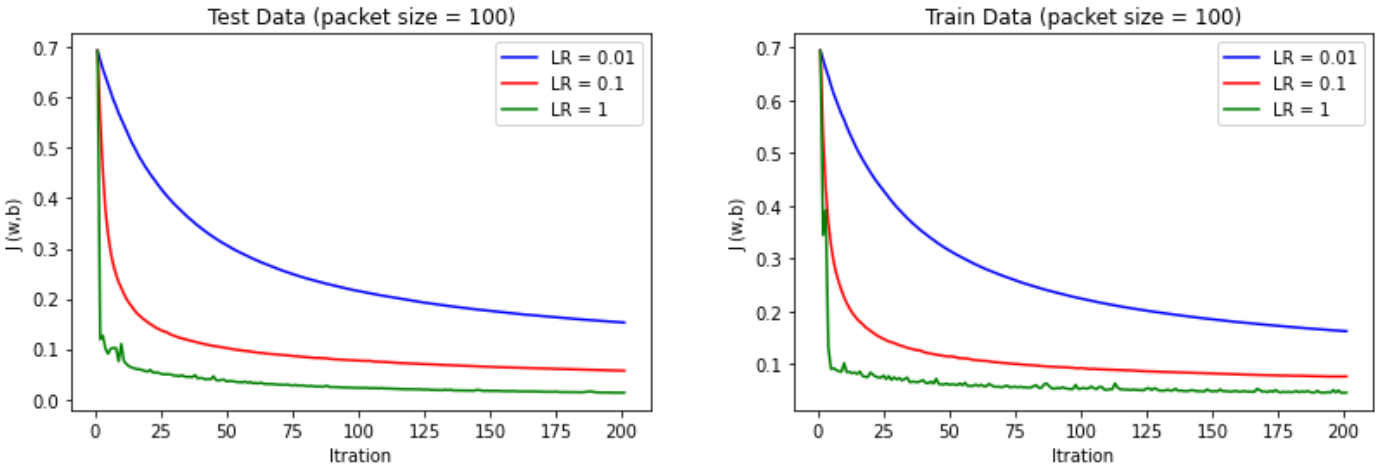
تصویر ۱-۶، مقادیر accuracy را برای هر سه گام و برای طول بسته ۱ نشان می‌دهد.

```
Accuracy for (Packet size = 1):
  Learning Rate = 0.01 : 96.88581314879%
  Learning Rate = 0.1 : 98.0968858131499%
  Learning Rate = 1 : 97.75086505190421%
```

تصویر ۱-۶: مقدار accuracy روی داده های test برای هر ۳ گام، طول بسته ۱ (ماشین learn شده با داده های train)

۲- طول بسته : ۱۰۰

تصویر ۱-۷، نمودار تابع هزینه به ازای تکرار برای طول بسته ۱۰۰ و گام های مختلف را نشان می‌دهد.



تصویر ۱-۳: مقدار تابع هزینه به ازای تعداد تکرار برای ۳ گام مختلف، طول بسته ۱۰۰

تصویر ۱-۸، مقادیر accuracy را برای هر سه گام و برای طول بسته ۱۰۰ نشان می‌دهد.

```
Accuracy for (Packet size = 100):
  Learning Rate = 0.01 : 97.23183391003569%
  Learning Rate = 0.1 : 98.0968858131499%
  Learning Rate = 1 : 98.78892733564125%
```

تصویر ۱-۸: مقدار accuracy روی داده های test برای هر ۳ گام، طول بسته ۱۰۰ (ماشین learn شده با داده های train)

نتایج: همانطور که در تصاویر مشاهده می‌شود برای داده های train بیشتر در هر تکرار (طول بسته بزرگتر) ماشین بهتر learn شده و مقدار تابع هزینه کمی نزولی‌تر (پایین‌تر) است.

سوال ۲: بهینه سازی در توابع غیر محدب

$$f(x_1, x_2) = 2x_1^2 + 2x_2^2 - 17x_2\cos(0.2\pi x_1) - x_1x_2$$

قسمت الف: روش نیوتن (تحلیلی)

- **جهت گرادیان نزولی:** نقطه شروع : $(0,0)$

ابتدا گرادیان تابع هدف داده شده را بدست می‌آوریم:

$$\frac{d}{dx}f(x) = \begin{bmatrix}\frac{df(x)}{dx_1}\\ \frac{df(x)}{dx_2}\end{bmatrix} = \begin{bmatrix}4x_1^{(k)} + 3.4\pi x_2^{(k)}\sin\left(0.2\pi x_1^{(k)}\right) - x_2^{(k)}\\ 4x_2^{(k)} - 17\cos\left(0.2\pi x_1^{(k)}\right) - x_1^{(k)}\end{bmatrix}$$

$$\begin{bmatrix}x_1^{(0)}\\ x_2^{(0)}\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix} \rightarrow \begin{bmatrix}\frac{df(x)}{dx_1}\\ \frac{df(x)}{dx_2}\end{bmatrix} = \begin{bmatrix}0\\ -17\end{bmatrix}$$

حال ماتریس هسین را برای تابع هدف بدست می‌آوریم:

$$H = \begin{bmatrix}\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2}\\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2}\end{bmatrix} = \begin{bmatrix}4 + 0.68\pi^2 x_2\cos\left(0.2\pi x_1\right) & 3.4\pi\sin(0.2\pi x_1) - 1\\ 3.4\pi\sin(0.2\pi x_1) - 1 & 4\end{bmatrix}$$

$$H_{x_1=0,\, x_2=0} = \begin{bmatrix}4 & -1\\ -1 & 4\end{bmatrix} \rightarrow H^{-1} = \begin{bmatrix}\frac{4}{15} & \frac{1}{15}\\ \frac{1}{15} & \frac{4}{15}\end{bmatrix}$$

$$x^{(k+1)} = x^{(k)} - H^{-1}\frac{d}{dx}f(x) \rightarrow \begin{bmatrix}x_1^{(k+1)}\\ x_2^{(k+1)}\end{bmatrix} = \begin{bmatrix}x_1^{(k)}\\ x_2^{(k)}\end{bmatrix} - \begin{bmatrix}\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2}\\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2}\end{bmatrix}\begin{bmatrix}\frac{df(x)}{dx_1}\\ \frac{df(x)}{dx_2}\end{bmatrix}$$

$$\begin{bmatrix}x_1^{(1)}\\ x_2^{(1)}\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix} - \begin{bmatrix}\frac{4}{15} & \frac{1}{15}\\ \frac{1}{15} & \frac{4}{15}\end{bmatrix}\begin{bmatrix}0\\ -17\end{bmatrix} = \begin{bmatrix}\frac{17}{15}\\ \frac{68}{15}\end{bmatrix}$$

قسمت ب: روش نیوتن (شبیه سازی)

در این قسمت ابتدا به ازای نقطه شروع $(1,3)$، نقطه بهینه تابع را بدست می‌آوریم. به این منظور تابع opt_f نوشته شده است. با توجه به اینکه تابع مورد نظر convex نیست نمی‌توانیم شرط توقف را روی گرادیان و یا افزایش مقدار تابع هزینه در نظر بگیریم. به این منظور تعداد تکرار را ثابت در نظر می‌گیریم.

تصویر ۲-۱ نقطه و مقدار بهینه بدست آمده تابع را نشان می‌دهد.

```
minimum of f(x1,x2) : f(0.13087466007249957,4.268357652256218) = -36.40349774185023
```

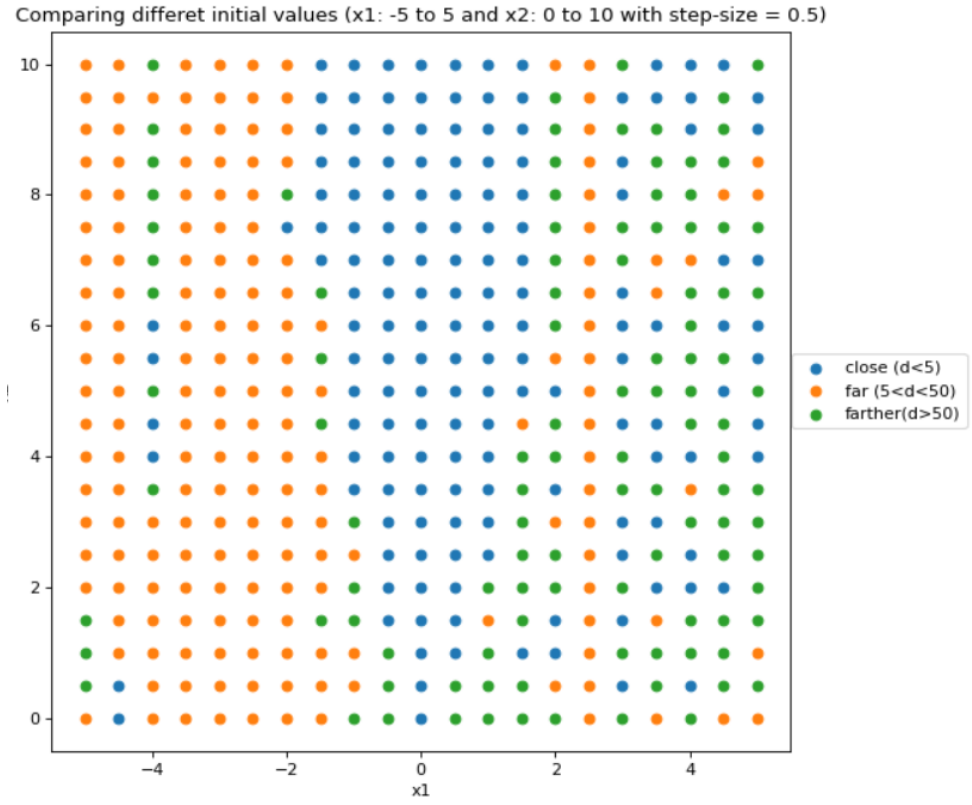
تصویر ۲-۱: نقطه و مقدار بهینه $f(x_1, x_2)$ با روش نیوتن

حال مطابق با توضیحات صورت تمرین، به ازای نقاط اولیه مختلف ($x_1 = -5: 0.5: 5$ و $x_2 = 0: 0.5: 10$) نقطه بهینه را بدست آورده و سپس فاصله آن‌را با نقطه بهینه اصلی (-36.4) محاسبه می‌کنیم. در نهایت نیز سه گروه نزدیک، دور و دورتر را به صورت زیر مرزبندی می‌کنیم.

نزدیک: اگر فاصله کمتر از ۵ باشد.

دور: اگر فاصله بین ۵ و ۵۰ باشد.

دورتر: اگر فاصله بیشتر از ۵۰ باشد.



تصویر ۲-۲: فاصله نقطه بهینه بدست آمده با نقطه بهینه اصلی به ازای نقاط اولیه مختلف

تحلیل نتایج: همانطور که در تصویر ۲-۲ مشاهده می‌شود، به ازای نقاط اولیه دور از نقطه بهینه، الگوریتم در نقاط کمینه محلی گیر می‌کند و نقطه کمینه اصلی را تشخیص نمی‌دهد.

قسمت ج: روش فراابتکاری – الگوریتم ژنتیک

در این قسمت جمعیت را تعداد دلخواه N نقطه در نظر می‌گیریم؛ سپس در مرحله selection بهترین نمونه (با مینیمم مقدار تابع هدف) را انتخاب می‌کنیم و در جمعیت جدید وارد می‌کنیم. $N-1$ نمونه دیگر را به این صورت انتخاب می‌کنیم که از بین دو نقطه رندم انتخاب شده از N نقطه قبلی هر کدام مقدار تابع کمتری داشت را وارد جمعیت جدید می‌کنیم. در مرحله crossover، اعداد انتخاب شده را به یک عدد باینری ۸ بیتی تصویر می‌کنیم. برای این کار با توجه به توضیحات صورت سوال، هرکدام از x_1 و x_2 را به صورت یک آرایه از 15- تا 15+ با تعداد عناصر ۲۵۵ در نظر می‌گیریم. سپس ترتیب جمعیت را تغییر می‌دهیم و هر دو عضو کنار هم را یک pair در نظر می‌کنیم و یک عدد رندوم از ۱ تا ۷ انتخاب می‌کنیم و از سمت چپ به همان تعداد بیت دو pair را جفت می‌کنیم و مجدداً این کار را انجام می‌دهیم تا یک child دیگر ایجاد شود و به همین ترتیب جمعیت جدید را با همان تعداد N تولید می‌کنیم. در مرحله mutation با احتمال mutation_rate تعداد mutation_select از جمعیت انتخاب می‌کنیم. سپس تعداد mutation_window از بیت های آن را گروه انتخاب شده را به صورت رندوم انتخاب کرده و قرینه می‌کنیم. (اگر 0 بود 1 و اگر 1 بود صفر می‌کنیم.) در نهایت مجدداً اعداد را از حالت باینری خارج کرده و به مرحله اول برمی‌گردیم. تمام این مراحل را به تعداد ITR بار تکرار می‌کنیم.

توضیحات توابع نوشته شده:

۱- **تابع Genetic()** : این تابع شامل تمامی مراحل الگوریتم است. ورودی، پارامتر های جمعیت و الگوریتم و خروجی آن عدد بهینه به‌دست آمده است.

۲- **تابع init_Pop()** : این تابع جمعیت اولیه را به صورت رندوم با تعداد ورودی داده شده انتخاب می‌کند.

۳- **تابع selection()** : این تابع مرحله selection را مطابق با توضیحات بالا انجام می‌دهد.

۴- **تابع crossover()** : این تابع مرحله crossover را مطابق با توضیحات بالا انجام می‌دهد.

۵- **تابع mutation()** : این تابع مرحله mutation را مطابق با توضیحات بالا انجام می‌دهد.

۶- **تابع BtD()** : این تابع اعداد را از حالت باینری خارج کرده و به حالت پیش از مرحله crossover در می‌آورد.

حال پارامتر های مسئله را به صورت زیر تعیین می‌کنیم. خروجی این قسمت (نقطه و مقدار کمینه تابع) در تصویر ۲-۳ نشان داده شده است.

```
ITR = 100
Pop_size = 1000
mutation_rate = 0.1
mutation_window = 1
mutation_select = 5
```

```
minimum of f(x1,x2) using Genetic Algorithm: f(0.11811023622047244,4.251968503937007) = -36.40034296457406
```

تصویر ۲-۳: نقطه و مقدار بهینه $f(x_1, x_2)$ با استفاده از الگوریتم ژنتیک

سوال ۳: ماشین بردار پشتیبان

قسمت الف: تحلیلی

سوال ۱. از صفر شدن ξ_i یک نمونه نمی‌توان نتیجه گرفت که آن نمونه در کدام کلاس است. اگر $y_i(W^T x_i + b) \geq 1$ باشد، داده ها بیرون از بردار های پشتیبان (در ناحیه خود) قرار می‌گیرند و در حد فاصل بردار پشتیبان و مرز تصمیم گیری قرار نگرفته اند.

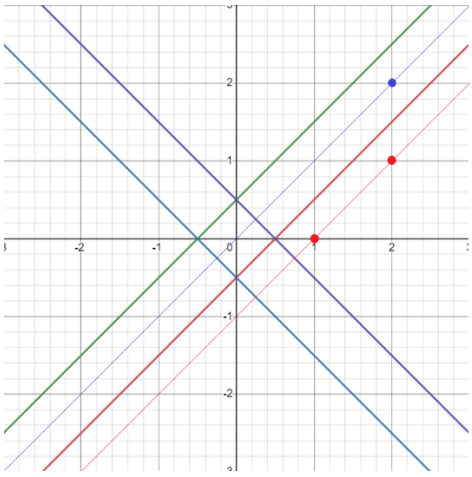
$$\xi_i = \max(0, 1 - y_i(W^T x_i + b))$$

$$if \quad \xi_i = 0 \rightarrow y_i(W^T x_i + b) \geq 1$$

می‌دانیم : $W^T x_i + b \geq +1 \rightarrow y_i = +1 \quad or \quad W^T x_i + b \leq -1 \rightarrow y_i = -1 \Rightarrow y_i(W^T x_i + b) \geq 1$

همچنین با توجه به اینکه فاصله هر بردار پشتیبان تا مرز دو کلاس برابر $d = \frac{1}{\|\omega\|}$ است، در صورت صفر شدن همه ξ_i ها، $\| \omega \|$ نیز صفر می‌شود و در نتیجه فاصله بین داده های دو کلاس بی‌نهایت می‌شود. بنابراین از صفر شدن ξ_i ها می‌توان نتیجه گرفت که فاصله بردار پشتیبان و مرز بین دو کلاس مختلف حداکثر است و ناحیه (فضای) خالی بین دو کلاس بسیار زیاد است. (به‌عبارت دیگر ویژگی استخراج شده از داده ها، به خوبی کلاس ها را از هم تفکیک می‌کند.)

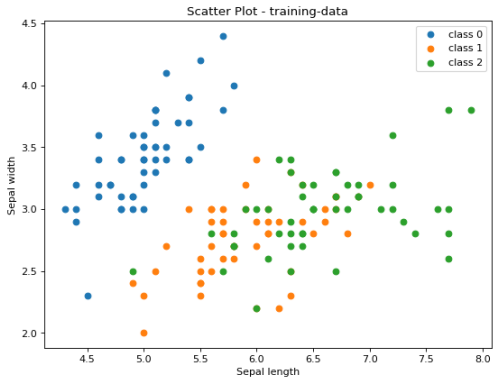
سوال ۲. مرز تصمیم بهینه، باید بین دو بردار پشتیبان طوری قرار گیرد که فاصله آن از هردو بردار پشتیبان به یک اندازه باشد و همچنین داده های دو کلاس رو تا حد امکان از هم متمایز کند و حداکثر تعداد داده های هر کلاس را از کلاس دیگر جداسازی کند. همانطور که در تصویر ۳-۱ مشاهده می‌شود، مرز $y = x - \frac{1}{2}$ دقیقاً بین دو بردار پشتیبان قرار گرفته است و در نتیجه مرز تصمیم بهینه خواهد بود.



تصویر ۳-۱: مرز تصمیم بهینه : $y = x - \frac{1}{2}$

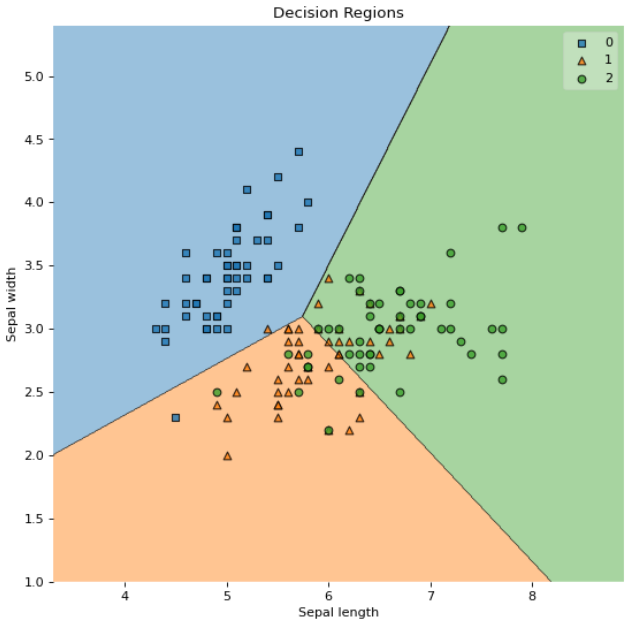
قسمت ب: پیاده سازی

نمودار عرض کاسبرگ بر حسب طول کاسبرگ داده های مختلف از هر کلاس در تصویر ۳-۲ نشان داده شده است.



تصویر ۳-۲: عرض کاسبرگ بر حسب طول کاسبرگ داده های مختلف هر کلاس

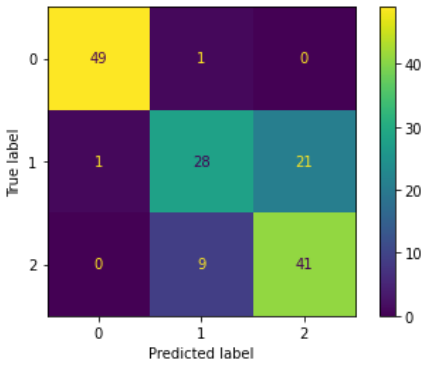
حال با استفاده از کتابخانه sklearn سعی می‌کنیم ماشین بردار پشتیبان (SVM) را پیاده سازی کرده و نواحی کلاس ها مختلف را با روش One vs Rest طبقه بندی می‌کنیم. تصویر ۳-۳ ناحیه های مختلف طبقه بندی را نشان می‌دهد.



تصویر ۳-۳: نواحی طبقه بندی کلاس ها با SVM و استفاده از One vs Rest

همچنین با استفاده از کتابخانه sklearn، دقت دادهٔ آموزش و ماتریس آشفتگی را بدست می‌آوریم. تصویر ۳-۴ دقت دادهٔ آموزش و ماتریس آشفتگی (confusion) را نشان می‌دهد.

```
Accuracy (using SVM with One vs Rest Method): 78.66666666666666%
```



تصویر ۳-۴: دقت دادهٔ آموزش و ماتریس آشفتگی (confusion)

همچنین ماتریس اطمینان (confidence) را با نرمالیزه کردن ماتریس آشفتگی (confusion) به صورت ستونی (روی Predicted Label) به دست می‌آوریم.

تصویر ۳-۵ ماتریس اطمینان (confidence) را برای ماشین طراحی شده نشان می‌دهد.

```
Confusion Matrix: (rows: True Labels, columns: Predicted Labels)
 [[49  1  0]
 [ 1 28 21]
 [ 0  9 41]]

Confidence Matrix:
 [[0.98       0.02631579 0.        ]
 [0.02       0.73684211 0.33870968]
 [0.         0.23684211 0.66129032]]
```

تصویر ۳-۵: ماتریس اطمینان (confidence)