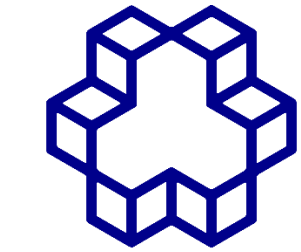
Google drive: https://drive.google.com/drive/folders/1n8uWALN39KWm7Zrq6tpUlCi1efUUQn4D?usp=sharing

Github: https://github.com/AliBagheriNejad/ML-AliYari



دانشگاه صنعتی خواجه نصیرالدین طوسی

مینی پروژه ۱ درس یادیگیری ماشین

علی باقری‌نژاد

۴۰۲۰۲۸۵۴

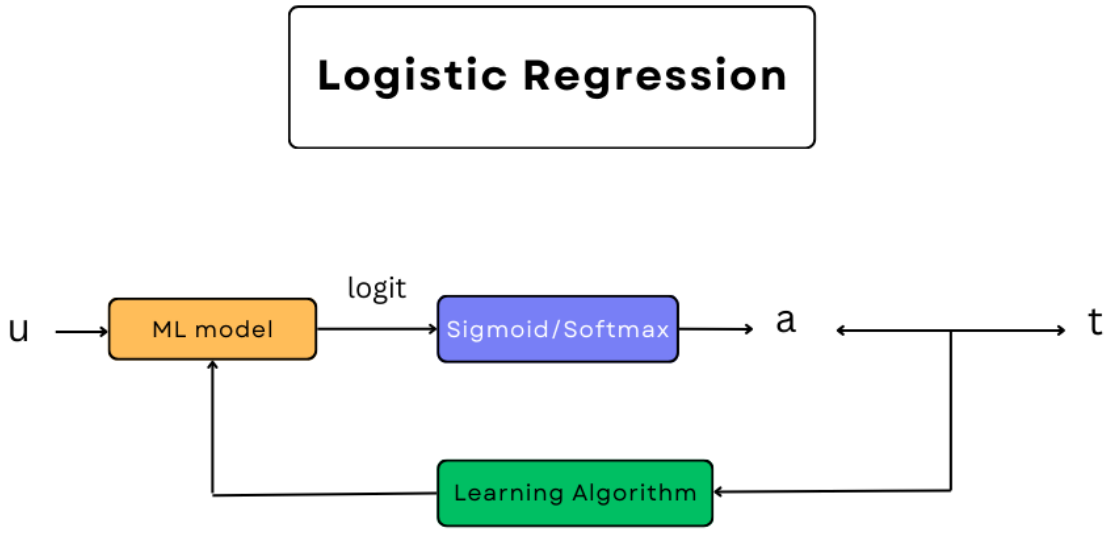
فهرست

فهرست تصاویر

فهرست جداول

سؤال ۱

۱-۱



شکل ۱بلوک دیاگرام مدل طبقه بندی خطی (Logistic Regression)

شکل ۱ بلوک دیاگرام مدل طبقه بندی خطی[۱] را به نمایش در آورده است. این مدل برای هر دو حالت دسته‌بندی[۲] دو کلاسه و چند کلاسه صادق است. در ادامه به توضیح مختصر عملکرد هر یک اجزاء این بلوک دیاگرام می‌پردازیم.

عملکرد کلی مدل طبقه‌بندی خطی:

- در اولین مرحله، ورودی به‌صورت یک بردار (**u**) وارد بلوک یادگیری ماشین می‌شود.
- در بلوک نارنجی یک عملیات ضرب داخلی ماتریسی روی ورودی انجام می‌شود. عملیات ذکر شده به‌صورت زیر است:

$$logit =< \boldsymbol{w}^t.\boldsymbol{u} >$$

---

[۱] Logistic Regression

[۲] Classification

در این مرحله، یک بردار وزن در u، ضرب می‌شود و یک عدد تولید می‌کند. مقادیر موجود در بردار وزن مشخص می‌کنند که هر کدام از ویژگی‌ها چه مقدار اهمیت دارند. خروجی بلوک یادگیری ماشین، یک عدد است (در صورت حالت چند کلاسه یک بردار به‌اندازه تعداد کلاس‌ها).

- خروجی بلوک یادگیری ماشین (logit) وارد بلوک تابع فعال‌سازی[۱] می‌شود. دلیل این که از خروجی بلوک ML به‌صورت خام استفاده نمی‌کنیم این است که خروجی بلوک ذکر شده یک عدد در بازه $[-\infty, +\infty]$ می باشد ولی ما نیاز به این داریم که خروجی نهایی عدد ۰ یا ۱ (در صورت دوکلاسه بودن طبقه بندی) باشد. در این مرحله از توابعی مثل sigmoid استفاده می کنیم که خروجی logit را گرفته و به عنوان خروجی نهایی مدل، یک عدد بین ۰و۱ تحویل می دهد. بر اساس مقدار آستانه‌ای[۲] که به دلخواه تعیین می کنیم، بر اساس خروجی a، یک کلاس به ورودی اختصاص می دهیم.
- خروجی a با برچسب[۳] t مقایسه می‌شود تا عملکرد مدل ماشین لرنیگ مقایسه شود. با مقایسه a و t و بر اساس روش‌هایی همچون MSE یا MAE یا... یک خطایی برای مدل در نظر می‌گیریم و در ادامه با استفاده از الگوریتم یادگیری[۴] (یا الگوریتم بهینه‌سازی[۵]) وزن‌های مدل را بهبود می‌دهیم.
- اتفاقی که در بلوک یادگیری می‌افتد این که است که با استفاده از خطا به‌دست‌آمده و روش‌هایی همچون SGD یا Adam یا... که عموماً الگوریتم‌های گرادیان بیس هستند، وزن‌های موجود را به صورتی تغییر می‌دهیم که مقدار تابع هزینه[۶] به کمینه خود نزدیک شود. به‌عنوان‌مثال فرمول ریاضی الگوریتم گرادیان نزولی[۷] را نمایش می‌دهیم:

---

[۱] Activation Function
[۲] Threshold
[۳] Label
[۴] Learning Algorithm
[۵] Optimization Algorithm
[۶] Cost function
[۷] Gradient Descend

$$w := w - \alpha \frac{\partial J}{\partial w}$$

تفاوت طبقه‌بندی دو کلاسه و چند کلاسه:

هنگامی طبقه‌بندی دو کلاسه، اگر ماشین، خروجی را بیشتر از ۰.۵(در صورتی که مقدار آستانه ۰.۵ باشد) قرار داد، ورودی متناظر را به کلاس ۱ متعلق می‌دانیم ولی اگر خروجی را کمتر از ۰.۵ قرار داد، ورودی متناظر را به کلاس ۰ اختصاص می‌دهیم. یعنی در حالت طبقه بندی دو کلاسه، صرفاً با داشتن یک عدد به عنوان خروجی می‌توانیم طبقه بندی را انجام دهیم. اما در حالتی که بیشتر از ۲ کلاس را طبقه بندی می‌کنیم، دیگر نمی‌توانیم صرفاً با یک عدد طبقه بندی را انجام دهیم. در این حالت باید درصد تعلق ورودی را به هر یک از کلاس‌های خروجی را محاسبه کنیم. مثلاً اگر ۵ کلاس به عنوان خروجی داشته باشیم، ماشین باید توانایی این را داشته باشد که ۵ عدد به عنوان احتمال تعلق برای هر کلاس تحویل بدهد. یعنی می‌توان گفت که در حالت چند کلاسه، ماشین باید چندین مسئله را حل کند. هر مسئله نیز در واقع احتمال تعلق ورودی به هر یک از کلاس‌ها است. یعنی ماشین باید بررسی کند که احتمال تعلق ورودی به هر کلاس خروجی چقدر است.

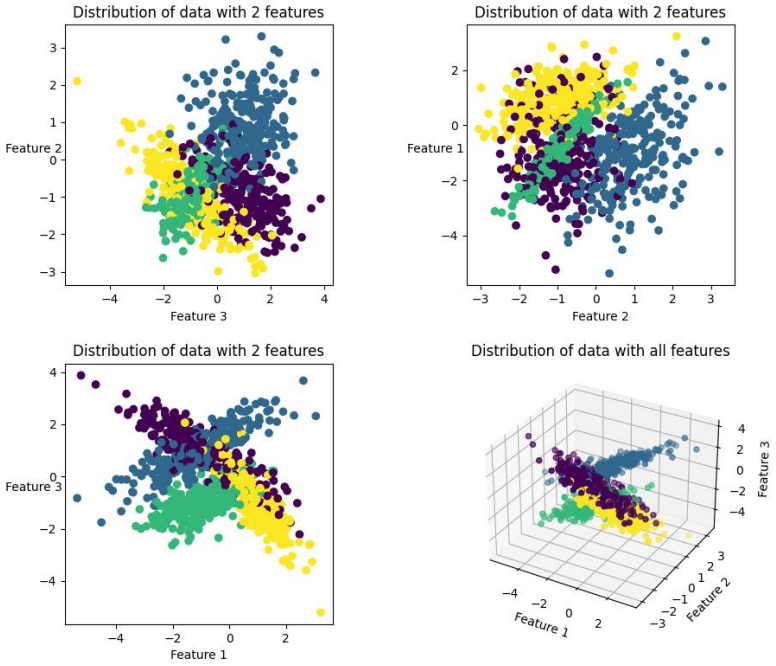
در حالت طبقه‌بندی دو کلاسه، خروجی ماشین تنها یک عدد بود؛ ولی در طبقه‌بندی چند کلاسه، ماشین باید به تعداد کلاس‌ها خروجی داشته باشد. یعنی باید بلوک نارنجی (ML model) را به‌گونه‌ای تغییر دهیم که در خروجی تعداد عدد بیشتری را نمایش دهد. البته باید بلوک آبی (Activation function)، برای تمامی خروجی‌ها عمل کند؛ یعنی خروجی این بلوک هم به تعداد خروجی‌های بلوک نارنجی است.

۲-۱ ایجاد دیتاست

در اولین قدم، با استفاده از دستور `make_classification` و ویژگی‌های زیر، دیتاست موردنظر را ایجاد می‌کنیم.

```
n_samples = 1000
n_features = 3
n_classes = 4
n_informative = 3
n_redundant = 0
n_repeated = 0
n_clusters_per_class = 1
clss_sep = 3
```

```
random_state = 69
```



شکل ۲ دیتاست ساخته شده

باتوجه‌به شکل ۲، می‌توان گفت که دیتاست ساخته شده برای طبقه بندی، مناسب نیست؛ زیرا با توجه به شکل، نقاط به خوبی از هم جدا نشده اند و نقاط داده‌ی[۱] کلاس‌های مختلف به صورت خطی قابل تفکیک[۲] نیستند.

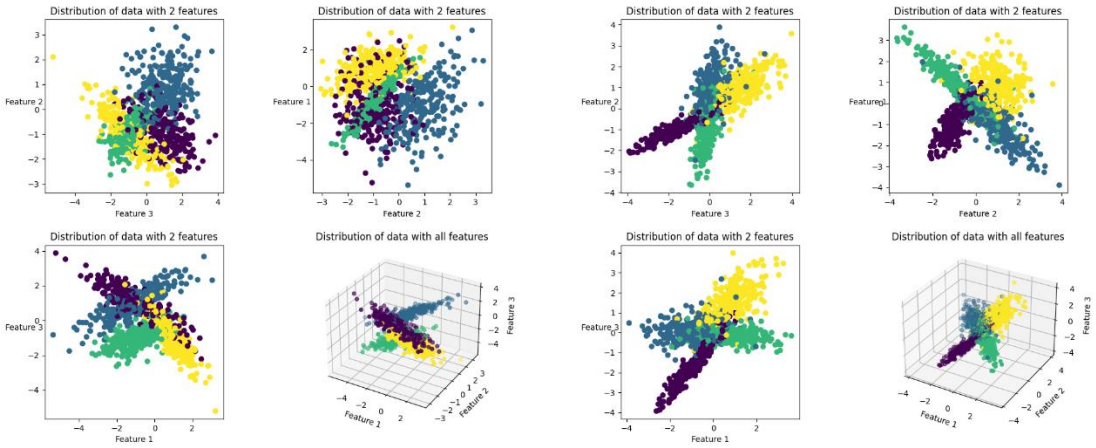
برای دشواری بیشتر می‌توان از راهکارهای زیر استفاده کرد:

- کاهش ویژگی `class_sep`: این ویژگی نماینده میزان جداسازی و فاصله دسته داده‌ها است. هر چه مقدار این ویژگی کمتر باشد، داده‌ها از هم کمتر جدا بوده و جداسازی آن‌ها دشوارتر است.

[۱] Data points

[۲] Linearly separable

- ایجاد عدم تعادل[۱] در داده: برای انجام این کار می‌توان نسبت تعداد داده‌های هر کلاس به هم را تغییر داد تا تعداد آن‌ها یکسان نباشد. این عمل با استفاده از ویژگی `weight` انجام می شود. با تغییر تعادل در داده، امکان یادگیری اشتباه برای ماشین وجود خواهد داشت.
- اعمال نویز[۲]: با اعمال نویز به داده، می‌توان در روند یادگیری ماشین اخلال ایجاد کرد و دقت آن را کاهش داد.
- افزایش استقلال خطی بین ویژگی‌ها: ویژگی n_informative تعیین می‌کند که چه تعداد از ویژگی‌ها دارای استقلال خطی باشند. در دیتاست ایجاد شده، n_informative برابر ۳ در نظر گرفته شده است. اگر این مقدار را کاهش دهیم و مثلاً n_redundant را افزایش دهیم، دیتاست راحتر قابل تفکیک خواهد بود.



شکل ۳دیتاست با n_redundant های مختلف

شکل ۳، دو تصویر از دیتاست‌هایی با استقلال خطی مختلف بین ویژگی‌هایشان را به نمایش گذاشته است. شکل سمت راست دارای استقلال خطی کمتری نسبت به شکل سمت چپ می باشد. همانطور که مشاهده می‌شود، شکل با استقلال خطی کمتر، قابلیت تفکیک پذیری خطی بهتری دارد.

---

[۱] Imbalance

[۲] Noise

۳-۱ ایجاد مدل طبقه بندی خطی

در اولین قدم، داده‌های ایجاد شده در بخش قبل را به دسته‌های آموزش[۱] و اعتبارسنجی[۲] تقسیم می‌کنیم. این تقسیم با استفاده از دستور `train_test_split` از کتابخانه scikit-learn انجام شد و نسبت تقسیم داده‌ها نیز ۸۰ به ۲۰ بود.

پس از تقسیم‌کردن داده‌ها به دسته‌های آموزش و صحه‌سنجی، با استفاده از تابع `StandardScaler` از کتابخانه scikit-learn، یک تبدیل‌گر برای داده‌های آموزش ایجاد کرده و بر اساس آن داده‌های آموزش و صحه‌سنجی را استاندار سازی کردیم.

LogisticRegression

اولین مدلی که برای ایجاد ماشین یادگیری از آن استفاده کردیم، مدل رگرسیون لجستیک بود. این مدل با استفاده از تابع `LogisticRegression` از کتابخانه scikit-learn ایجاد شد که پارامترهای استفاده شده برای آموزش در ادامه آورده شده‌اند:

```
penalty = 'l2'
fit_intercept = True
random_state = 69
```

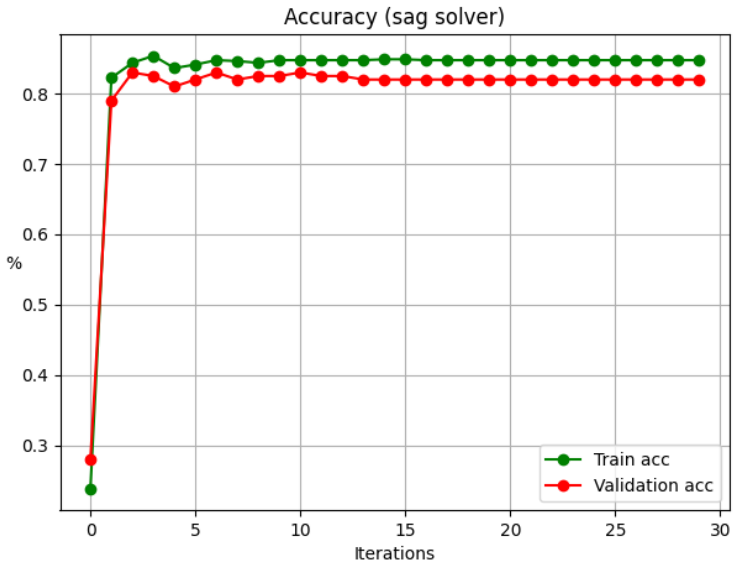
البته این فراپارامترها[۳]، فراپارامترهایی بودند که برای مقایسه تمامی مدل‌های رگرسیون لجستیک ثابت در نظر گرفته شدند. دو فراپارامتر تعداد تکرار[۴] و حلگر[۵] به‌عنوان فراپارامترهایی در نظر گرفته شدند که عملکرد مدل به‌ازای مقادیر مختلف آن‌ها مورد آزمایش قرار گرفت.

---

۱ Train
۲ Validation
۳ Hyperparameters
۴ Iteration
۵ Solver

البته لازم به ذکر است که نرخ یادگیری[1] برای تابع رگرسیون خطی کتابخانه scikit learn قابل‌تعریف نبود.

تعداد تکرار: تعداد تکرار مدل از ۱ بار تکرار تا ۳۰ بار تکرار تست شد.



شکل ۴نمودار دقت مدل نسبت به تعداد تکرار

شکل ۴ دقت مدل را برای مجموعه داده آموزش و صحت سنجی به نمایش گذاشته است.

```
Iterations: 100%|          | 30/30 [00:00<00:00, 101.23it/s]
The best outcome on training data is for iteration 3 with accuracy of 85.38%
The best outcome on validation data is for iteration 2 with accuracy of 83.00%
```

شکل ۵ دقت و بهترین تکرار برای مجوعه داده ها

شکل ۵ مقدار بهترین دقت هر مجموعه داده و تکرار متناظر با آن را به نمایش گذاشته است.

حلگر: عملکرد مدل به‌ازای تعداد تکرار ۱۵ برای حلگرهای مختلف به صورت زیر می‌باشد:

[1] Learnin rate

```
Solver: lbfgs
Train Accuracy: 0.85
Validatoin Accuracy: 0.82

Solver: liblinear
Train Accuracy: 0.83
Validatoin Accuracy: 0.81

Solver: newton-cg
Train Accuracy: 0.85
Validatoin Accuracy: 0.82

Solver: newton-cholesky
Train Accuracy: 0.83
Validatoin Accuracy: 0.81

Solver: sag
Train Accuracy: 0.85
Validatoin Accuracy: 0.82

Solver: saga
Train Accuracy: 0.85
Validatoin Accuracy: 0.82
```

شکل ۶ عملکرد مدل به ازای حلگر ها

باتوجه‌به شکل ۶، حلگرهای lbfgs، newton-cg، sagو saga بهترین عملکرد را داشتند.

باتوجه‌به تابع LogisticRegression ای که در کتابخانه scikit-learn تعریف شده است، نمی‌توانستیم که خلاقیت‌های زیادی برای بهبود عملکرد مدل را اعمال کنیم. مثلاً این که در این تابع امکان تعریف نرخ یادگیری وجود نداشت و یا این که ضریب جریمه[۱] را نمی‌توانستیم تغییر دهیم و تعیین کنیم. تمامی کارهایی که می‌توان انجام داد، در تغییر تعداد تکرار و روش حلگر خلاصه می‌شود.

SGD(Stochastic Gradiant Descent)

دومین الگوریتم استفاده شده، SGD بود. برای انجام طبقه‌بندی با استفاده از الگوریتم SGD، از تابع `SGDClassifier` از کتابخانه scikit-learn استفاده کردیم. موارد زیر برای تمامی مدل‌های SGD یکسان بودند:

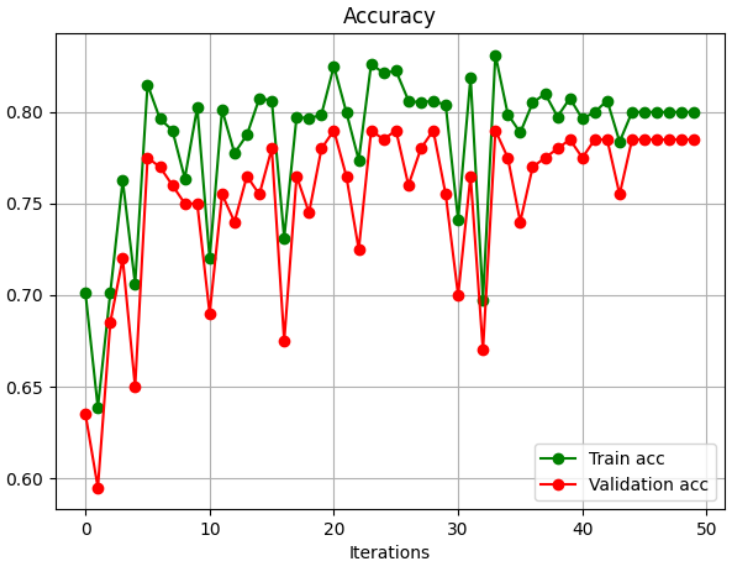
```
penalty = 'l2'
fit_intercept = True
random_state = 69
```

فراپارامترهایی که برای این الگوریتم بهینه‌سازی شدند به‌صورت مفصل در ادامه توضیح داده خواهند شد.

تعداد تکرار: برای آموزش این مدل، از ۱ تا ۵۰ بار تکرار استفاده کردیم که روند آن در شکل ۷ نشان داده شده است.

---

[۱] Penalty (Regularization)



شکل ۷نمودار دقت مدل نسبت به تعداد تکرار ها

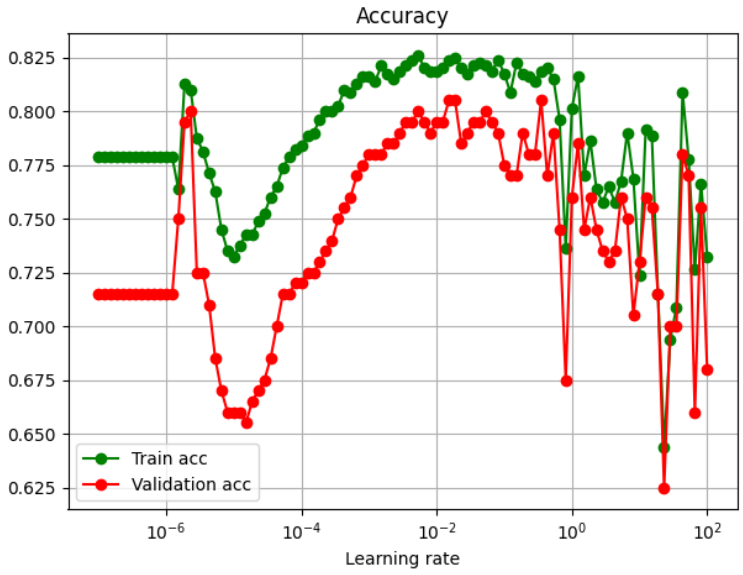
در شکل ۷ نمودار دقت بر حسب تعداد تکرار، برای مجموعه داده آموزش و صحه‌سنجی به نمایش درآمده است.

```
Iterations: 100%|          | 50/50 [00:00<00:00, 68.10it/s]
The best outcome on training data is for iteration 33 with accuracy of 83.12%
The best outcome on validation data is for iteration 20 with accuracy of 79.00%
```

شکل ۸ بهترین دقت و تکرار متناظر با آن

در شکل ۸ نیز بهترین دقت بدست آمده و شماره تکرار متناظر با آن، برای هر دو مجموعه داده آموزش و صحه‌سنجی به نمایش در آمده است.

نرخ یادگیری: در این حالت تعداد تکرار را ثابت در نظر گرفتیم و برای نرخ‌های یادگیری مختلف، عملکرد مدل را بررسی نمودیم.



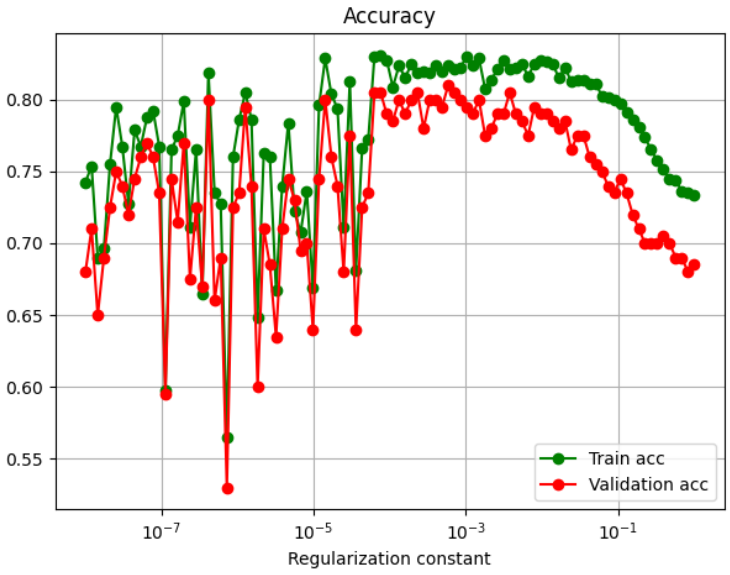
شکل ۹نمودار دقت نسبت به نرخ یادگیری

```
Iterations: 100%|          | 100/100 [00:01<00:00, 58.12it/s]
The best outcome on training data is for learning rate 0.0053 with accuracy of 82.62%
The best outcome on validation data is for learning rate 0.015 with accuracy of 80.50%
```

شکل ۱۰ بهترین دقت بدست آمده و نرخ یادگیری متناظر با آن

شکل ۱۰ بهترین نرخ یادگیری برای مجموعه‌های داده آموزش و صحه‌سنجی و دقت متناظر با آن را به نمایش گذاشته است.

ضریب تنظیم‌سازی α: این ضریبی است که مقدار penalty را کنترل می‌کند. حال برای ضرایب از 10^{-9} تا ۱ ، عملکرد سیستم را بررسی می کنیم.



شکل ۱۱نمودار تغییر دقت بر حسب ضریب regularization

```
Iterations: 100%|          | 100/100 [00:00<00:00, 121.00it/s]
The best outcome on training data is for regularization constant 7.6e-05 with accuracy of 83.12%
The best outcome on validation data is for regularization constant 0.00059 with accuracy of 81.00%
```

شکل ۱۲بهترین دقت بدست آمده و ضریب regularization متناسب با آن

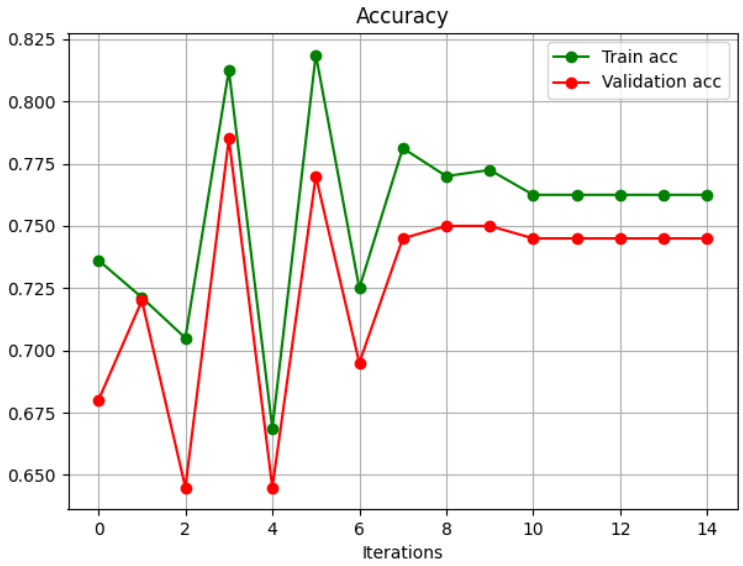
Perceptron

سـومین الگوریتم اسـتفاده شـده، الگوریتم perceptron بود که با اسـتفاده از تابع `Perceptron` از کتابخانه scikit-learn به دست آمد. پارامترهای ثابت برای تمامی مدل‌ها عبارت‌اند از:

```
penalty = 'l2'
fit_intercept = True
random_state = 69
```

فراپارامترهایی که برای این الگوریتم بهینه‌سازی شدند به‌صورت مفصل در ادامه توضیح داده خواهند شد.

تعداد تکرار: برای آموزش این مدل، از ۱ ۱۵تا بار تکرار استفاده کردیم که روند آن در شکل ۱۳ نشان داده شده است.

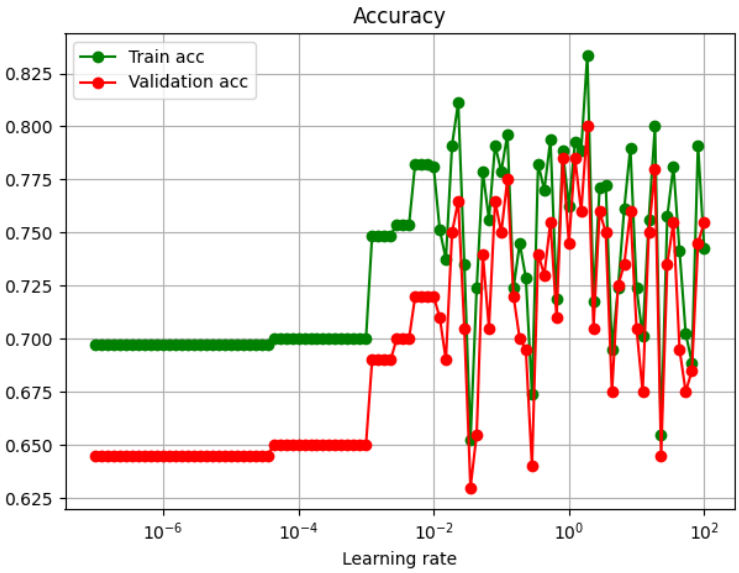


شکل ۱۳ نمودار دقت بر حسب تعداد تکرار

```
Iterations: 100%|          | 15/15 [00:00<00:00, 86.80it/s]
The best outcome on training data is for iteration 5 with accuracy of 81.88%
The best outcome on validation data is for iteration 3 with accuracy of 78.50%
```

شکل ۱۴ بهترین دقت بدست آمده و تعداد تکرار متناظر با آن

نرخ یادگیری: در این حالت تعداد تکرار را ثابت در نظر گرفتیم و برای نرخ‌های یادگیری مختلف، عملکرد مدل را بررسی نمودیم.

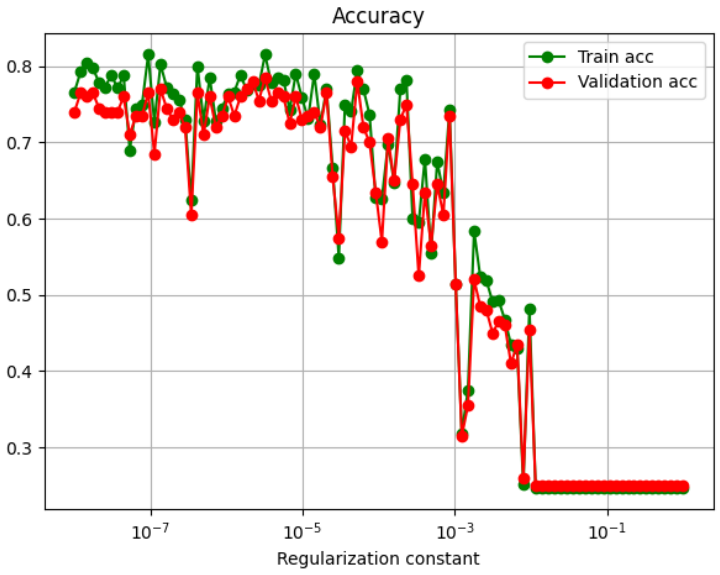


شکل ۱۵نمودار دقت بر حسب نرخ یادگیری

```
Iterations: 100%|██████████| 100/100 [00:00<00:00, 170.99it/s]
The best outcome on training data is for learning rate 1.9 with accuracy of 83.38%
The best outcome on validation data is for learning rate 1.9 with accuracy of 80.00%
```

شکل ۱۶بهترین دقت و نرخ یادگیری متناظر با آن

ضـریب تنظیم‌سـازی α: این ضـریبی اسـت که مقدار penalty را کنترل می‌کند. حال برای ضـرایب از 10^{-9} تا ۱ ، عملکرد سیستم را بررسی می کنیم.



شکل ۱۷نمودار دقت برحسب ضریب regularization

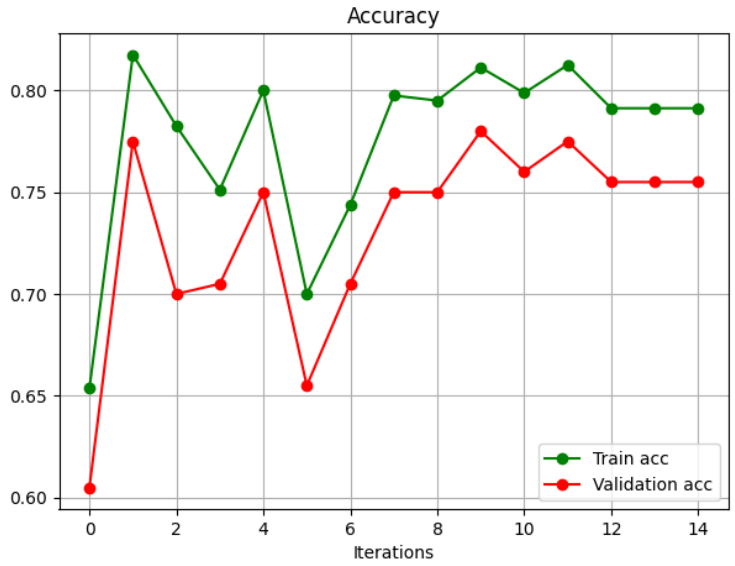
```
Iterations: 100%|          | 100/100 [00:00<00:00, 132.31it/s]
The best outcome on training data is for regularization constant 9.3e-08 with accuracy of 81.50%
The best outcome on validation data is for regularization constant 3.2e-06 with accuracy of 78.50%
```

شکل ۱۸ بیشترین دقت بدست آمده و ضریب regularization متناظر با آن

Passive Aggressive Classifier

برای اســـتفاده از این الگوریتم از تابع `PassiveAgressiveClassifier` موجود در کتابخانه scikit-learn اســـتفاده کردیم.

تعداد تکرار: برای بررسی این فراپارامتر از ۱ تا ۱۵ تکرار را بررسی کردیم.



شکل ۱۹ نمودار دقت بر حسب تعداد تکرار

```
Iterations: 100%|██████████| 15/15 [00:00<00:00, 166.76it/s]
The best outcome on training data is for iteration 1 with accuracy of 81.75%
The best outcome on validation data is for iteration 9 with accuracy of 78.00%
```

شکل ۲۰ بیشترین دقت بدست آمده و تعداد تکرار متناظر با آن

تابع استفاده شده برای این الگوریتم از هیچ نرخ یادگیری‌ای استفاده نمی‌کرد و نمی‌توانستیم که عملکرد سیستم را برای نرخ‌های یادگیری متفاوت بررسی کنیم.

تکنیک‌های بهبود مدل

تمرکز بخش‌های قبلی برو روی این بود که بهترین تعداد تکرار برای الگوریتم یا بهترین نرخ یادگیری را بیابند. اما در علم یادگیری ماشین، کارهای بسیار مختلف دیگری برای بهبود عملکرد مدل پیشنهاد می‌شود که بسیاری از آن‌ها صرفاً به تنظیم فراپارامترهای خود مدل نیز محدود نمی‌شوند.

تکنیک‌هایی که به‌صورت کلی می‌توان از آن‌ها برای بهبود عملکرد مدل استفاده کرد:

- اولین و مهم‌ترین تکنیکی که برای بهبود عملکرد مدل پیشنهاد می‌شود، نرمال‌سازی داده ورودی به ماشین است. این بدان معنا است که بازه تغییرات داده را به صفر تا ۱ یا ۱- تا ۱ انتقال دهیم. این کار برای تمامی

ویژگی‌ها انجام می‌شود. این کار به این علت انجام می‌شود که تاثیر تغییرات تمامی ویژگی‌ها به صورت یکسانی باشد.

- در بسیاری از موارد، نسبت تغییرات هر ویژگی را به تغییرات خروجی نهایی را می‌سنجند. با این کار می‌توان متوجه شد که کدام ویژگی‌ها روی خروجی نهایی تأثیر دارند و استفاده از چه ویژگی‌هایی به ایجاد یک مدل بهتر کمک می‌کند. مثلاً اگر با تغییرات متغیر x، خروجی تغییر نکرد، می‌توان آن ویژگی را حذف کرد تا از داده کمتری استفاده شود و سرعت محاسبات افزایش یابد.
- از جمله مهم‌ترین تکنیکی که بر روی داده ورودی انجام می‌شود، اصلاح عدم تعادل درداده است. این تکنیک برای مسائل طبقه‌بندی انجام می‌شود. هنگامی که از کلاس‌های مختلف خروجی، تعداد یکسانی داده در اختیار نداریم، ایجاد یک مدل یادگیری ماشین در این حالت باعث می‌شود که طبقه‌بندی‌ای که مدل انجام می‌دهد، دارای یک سوگیری[1] در طبقه‌بندی به نفع داده آموزش با تعداد بیشتر خواهد بود. برای جلوگیری از این مشکل، از روش‌های اصلاح عدم تعادل استفاده می‌شود.
- استفاده از توابع هزینه مناسب می‌توان بسیار به رسیدن به نتیجه مطلوب کمک کند. اگر تابع هزینه به صورتی تعریف شود که کمینه آن به معنای نزدیک‌تر بودن پاسخ مدل به نتیجه مطلوب باشد، قطعاً عملکرد مدل افزایش پیدا خواهد کرد.
- می‌توان گفت که مهم‌ترین بخش یادگیری ماشین، الگوریتمی است که برای بهینه‌سازی وزن‌های مدل استفاده می‌کنیم. پس با انتخاب الگوریتم بهینه‌سازی مناسب، هم به مدل بهتری دست پیدا خواهیم کرد و زمان کمتری برای آموزش مدل می‌گذاریم.
- ما باید باتوجه‌به تعداد داده‌های آموزش، و تعداد ویژگی‌های موجود، یک پیچیدگی مناسب برای مدل انتخاب کنیم. هر چه ویژگی‌های موجود درداده بیشتر باشد، باید مدل بزرگ‌تری انتخاب شود تا قادر باشد که همه آن‌ها را بررسی کرده و نتیجه مناسب را تحویل دهد. اگر تعداد داده آموزش بیشتری در اختیار داشته باشیم، مدل‌های

[1] Bias

کوچک قابلیت این را نخواهند داشت که از ویژگی‌های موجود در تمامی داده‌ها را برای ساخت ماشین استفاده کنند و هر مدلی می‌تواند تعداد محدودی داده را، متناسب با پیچیدگی خود مدل، برای ساخت مدل استفاده کند. داده کم باعث می‌شود که مدل به‌صورت ناقص آموزش داده شود و داده زیاد هم بیشتر از ظرفیت مدل بوده و تمامی ویژگی‌های موجود درداده استخراج نمی‌شود

- Regularization یک از مهم‌ترین مباحثی است که برای بهبود عملکرد مدل یادگیری ماشین استفاده می‌شود. این روش عموماً برای کاهش واریانس[1] اعمال می‌شوند. Regularization تکنیک‌های بسیار متنوعی را شامل می‌شود. مثل اعمال یک مقدار خطا اضافه به تابع هزینه برای کاهش وزن‌ها.

تکنیک‌های استفاده شده در این مسئله:

۱. نرمالایز کردن داده ورودی
۲. آموزش داده روی مدل‌های مختلف
۳. اعمال عبارت خطا اضافی (regularization) به تابع هزینه (در مدل‌هایی که قابلیت را داشتند)
۴. استفاده از حلگرهای مختلف و مقایسه نتایج

به دلیل کم‌بودن داده یا متعادل بودن کلاس‌ها و... نمی‌توانستیم از دیگر تکنیک‌های بهبود مدل استفاده کنیم.

۴-۱ نمایش مرز و نواحی تصمیم گیری

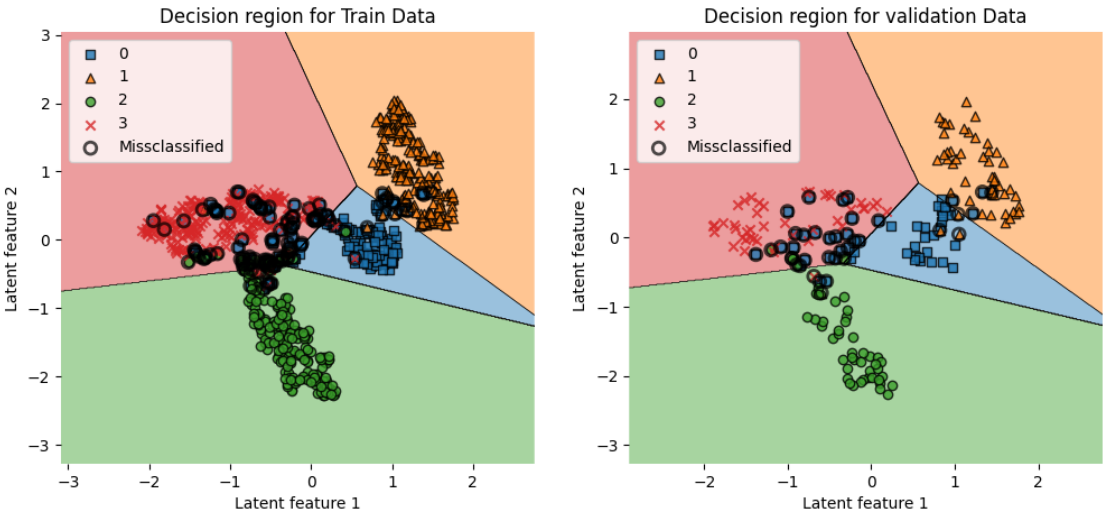
داده‌های تولید شده دارای ۳ ویژگی هستند. برای نمایش مرزها و نواحی تصمیم‌گیری می‌بایست که تعداد ویژگی داده‌ها را به ۲ بعد کاهش دهیم. برای کاهش ابعاد داده‌ها از الگوریتم t-SNE استفاده کردیم. کاهش ابعاد با استفاده از تابع `TSNE` از کتابخانه scikit-learn انجام شد.

[1] Variance

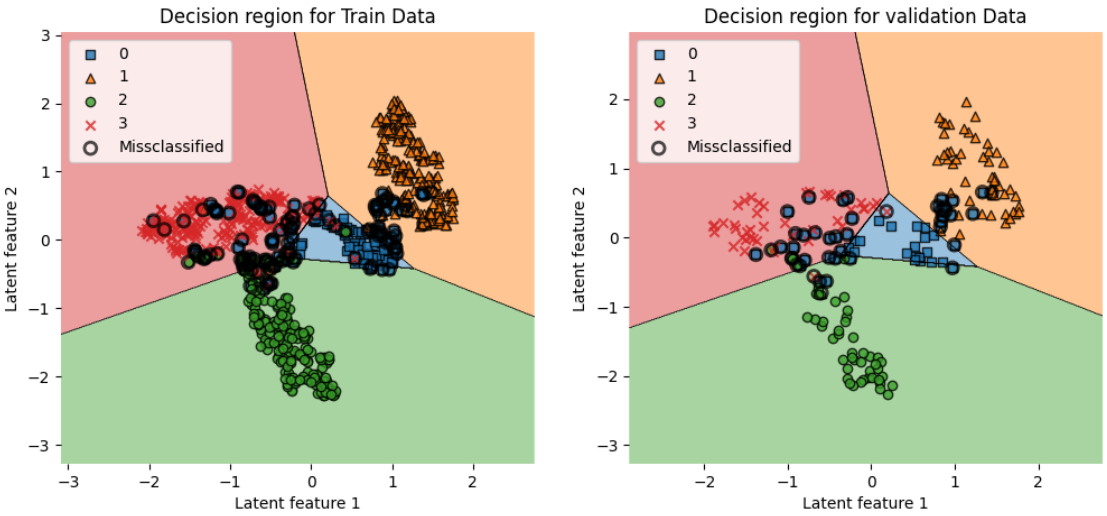
پس از کاهش ابعاد داده‌ها، مدل‌های یادگیری ماشـــین بار دیگر روی داده‌های جدید که دارای ۲ ویژگی بودند، آموزش داده شدند. لازم به ذکر است که پس از کاهش ابعاد داده، دوباره با استفاده از دستور `train_test_split`، بار دیگر داده‌ها را به دسته‌های آموزش و صحه‌سنجی تقسیم کردیم.

پس از آموزش دوباره مدل‌ها، با اسـتفاده از دسـتور plot_decision_regions از کتابخانه mlxtend، مرزها و نواحی تصمیم‌گیری به نمایش درآمدند.

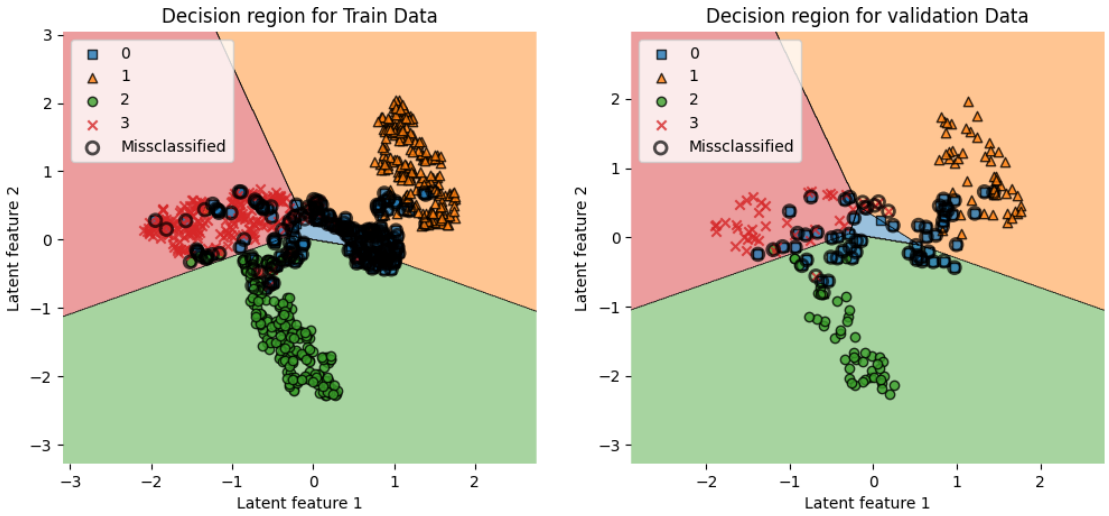
در ادامه نمودارهای به وجود آمده را برای هر مدل به نمایش می‌گذاریم.



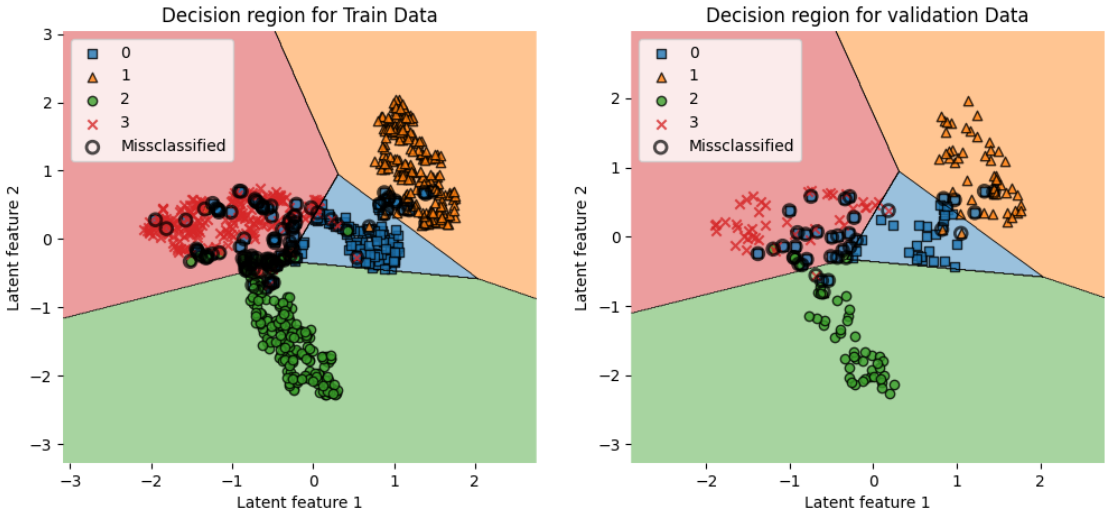
شکل ۲۱نواحی تصمیم گیری برای مدل Logisic Regression



شکل ۲۲نواحی تصمیم‌گیری برای مدل SDG



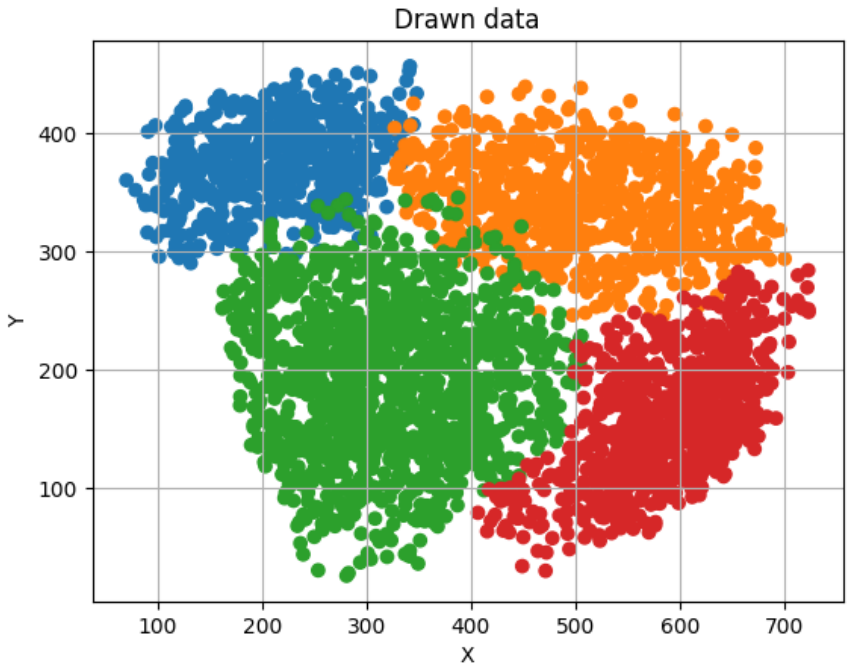
شکل ۲۳ نواحی تصمیم‌گیری برای مدل *Perceptron*



شکل ۲۴ نواحی تصمیم‌گیری برای مدل *Passive Aggressive Classifier*

۴-۱ استفاده از ابزار draw data

در اولین قدم با اسـتفاده از کتابخانه rawdata و با اسـتفاده از دسـتور ScatterWidget، یک مجموعه داده دلخواه که شامل ۴ کلاس می باشند را ایجاد می کنیم.



شکل ۲۵ پخش نقاط داده ایجاد شده

شکل ۲۵ پخش داده ایجاد شده را بر روی صفحه دوبعدی به نمایش گذاشته است. داده ایجاد شده را در یک دیتافریم پاندا[۱] ذخیره می کنیم. شکل ۲۶ دیتافریم ایجاد شده را به نمایش گذاشته است. در اولین قدم باید مقادیر رشته[۲] موجود در ستون برچسب[۳] را به مقادیر عدد تبدیل کنیم.

Shape of drawn data is (3504, 4)

	x	y	color	label
0	90.538044	316.306034	#1f77b4	a
1	88.802248	339.499024	#1f77b4	a
2	98.982651	312.549852	#1f77b4	a
3	104.882094	317.394244	#1f77b4	a
4	116.697923	327.698127	#1f77b4	a

شکل ۲۶ دیتافریم ایجاد شده از rawdata

۱ Pandas dataframe
۲ String
۳ Label

```
lable values before correction are: ['a' 'b' 'c' 'd']
label values after correction are: [0 1 2 3]
```

شکل ۲۷تبدیل مقادیر رشته‌ای به مقادیر عددی

شکل ۲۷ نمایش می دهد که مقادیر رشته‌ای را به مقادیر عددی تبدیل کردیم و می توانیم محاسبات را ادامه دهیم.

همانطور که از شکل ۲۵ و ۲۸ مشخص می باشد، داده ها دارای عدم تعادل می‌باشند. پیشتر درباره ضرر عدم تعادل در داده صحبت شد. برای رفع این عیب در دیتافریم، از روش SMOTE[1] استفاده می کنیم.

```
Number of data in each class before balancing:
2    1323
3     866
1     724
0     591
Name: label, dtype: int64
Number of data in each class after balancing:
0    1323
1    1323
2    1323
3    1323
Name: label, dtype: int64
```

شکل ۲۸تعداد داده ها قبل و بعد از متعادل کردن مجموعه داده

شکل ۲۸ نمایش می دهد که تعداد داده ها در همه کلاس ها برابر شده است.

بعد از نرمالایز کردن داده و تقسیم آن به مجموعه های آموزش و ارزیابی[۲]، مدل هایی بخش ۳ را دوباره اجرا می کنیم و نتایج بدست آمده را نمایش می دهیم.

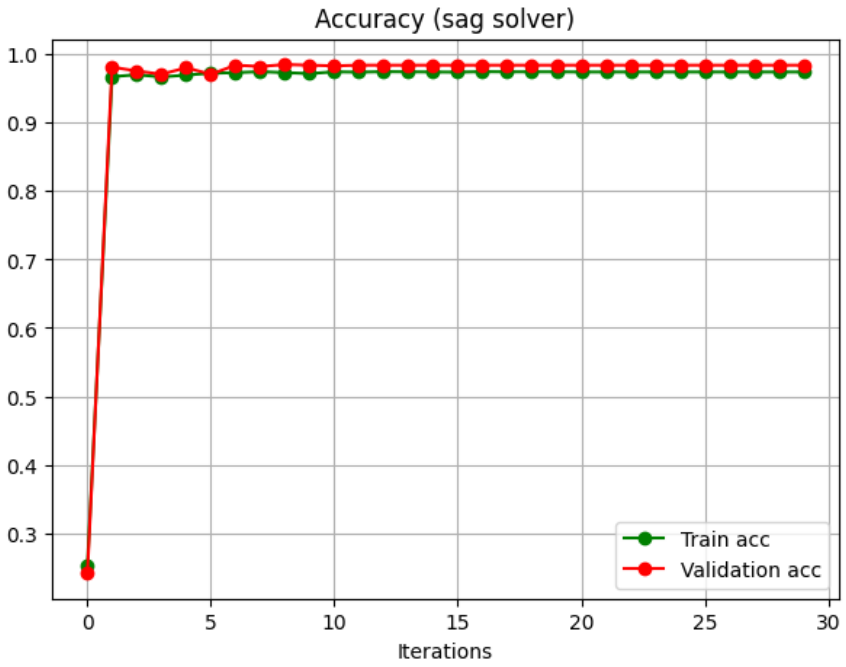
```
Mean value of train data:
BEFORE normalizaing -> 336.8340242533887
AFTER normalizaing -> 3.5834526673396145e-15
Mean value of validation data:
BEFORE normalizaing -> 336.38553746867785
AFTER normalizaing -> -0.006061250435698513
```

شکل ۲۹ میانگین مجموعه داده های آموزش و ارزیابی قبل و بعد از نرمال کردن

---

[۱] Synthetic Minority Over-sampling Technique

[۲] Test



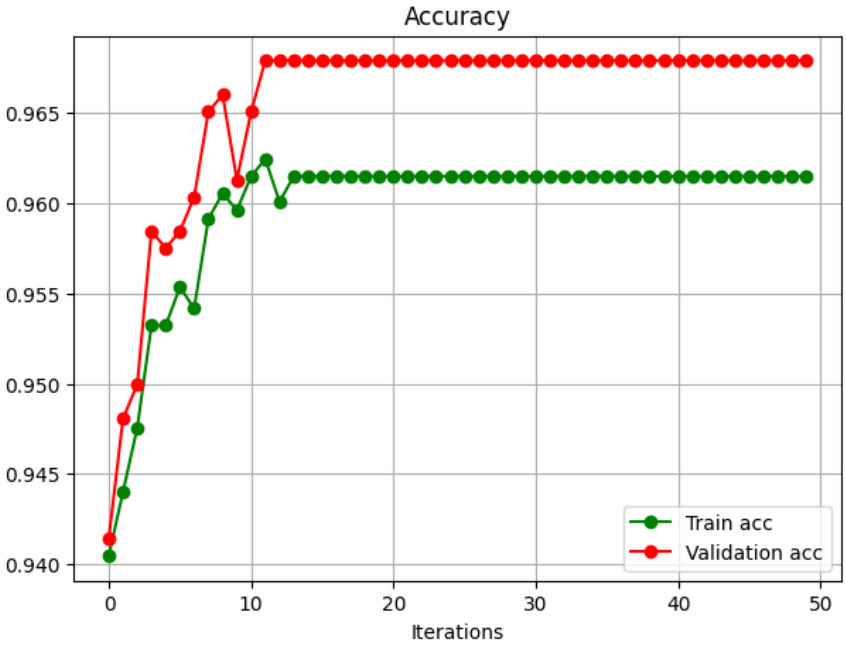
شکل ۳۰ نمودار دقت بر حسب تکرار

```
Iterations: 100%|██████████| 30/30 [00:00<00:00, 54.05it/s]
The best outcome on training data is for iteration 7 with accuracy of 97.45%
The best outcome on validation data is for iteration 8 with accuracy of 98.49%
```

شکل ۳۱ بهترین دقت و تکرار متناظر با آن

```
Solver: lbfgs
Train Accuracy: 0.97
Validatoin Accuracy: 0.98

Solver: liblinear
Train Accuracy: 0.96
Validatoin Accuracy: 0.97

Solver: newton-cg
Train Accuracy: 0.97
Validatoin Accuracy: 0.98

Solver: newton-cholesky
Train Accuracy: 0.97
Validatoin Accuracy: 0.97

Solver: sag
Train Accuracy: 0.97
Validatoin Accuracy: 0.98

Solver: saga
Train Accuracy: 0.97
Validatoin Accuracy: 0.98
```

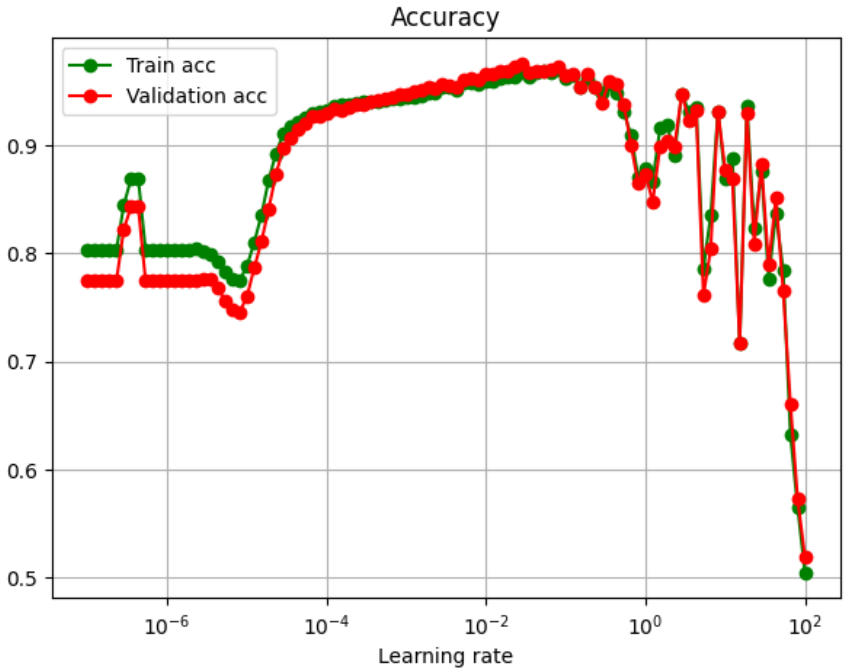
شکل ۳۲ دقت های بدست آمده برای حلگر های مختلف



شکل ۳۳نمودار دقت بر حسب تکرار

```
Iterations: 100%|██████████| 50/50 [00:01<00:00, 48.04it/s]
The best outcome on training data is for iteration 11 with accuracy of 96.24%
The best outcome on validation data is for iteration 11 with accuracy of 96.79%
```

شکل ۳۴ بهترین دقت بدست آمده و تکرار متناظر با آن

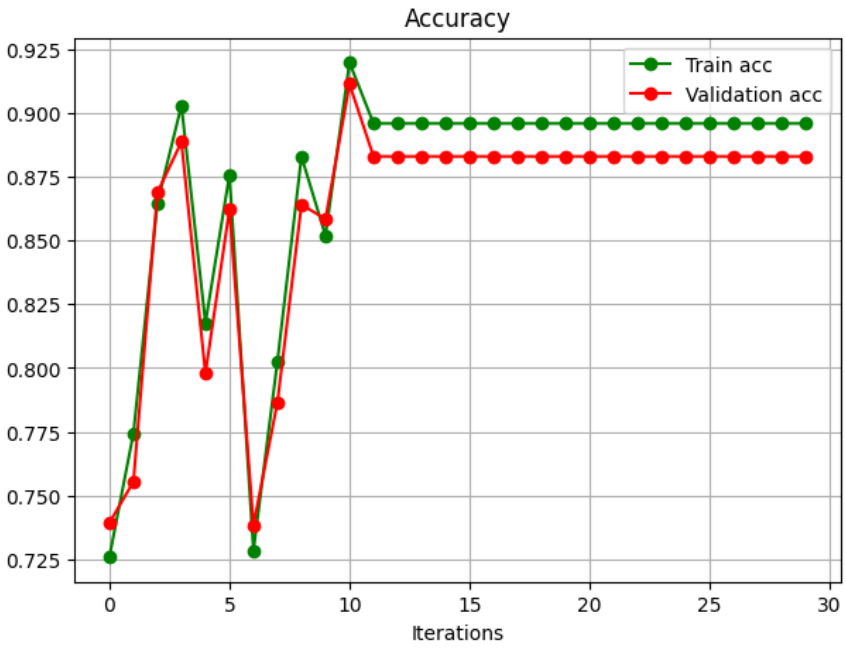


شکل ۳۵ نمودار دقت برحسب نرخ یادگیری

```
Iterations: 100%|          | 100/100 [00:02<00:00, 42.95it/s]
The best outcome on training data is for learning rate 0.081 with accuracy of 97.02%
The best outcome on validation data is for learning rate 0.028 with accuracy of 97.64%
```

شکل ۳۶ بهترین دقت بدست آمده و نرخ یادگیری متناظر با آن

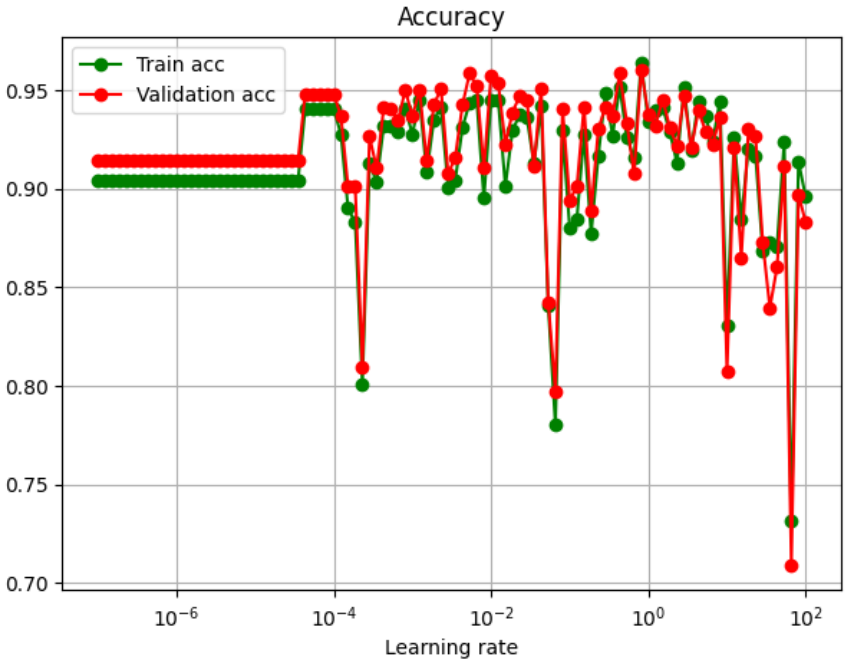
Perceptron



شکل ۳۷نمودار دقت بدست آمده برحسب تکرار

```
Iterations: 100%|████████| 30/30 [00:00<00:00, 33.93it/s]
The best outcome on training data is for iteration 10 with accuracy of 91.97%
The best outcome on validation data is for iteration 10 with accuracy of 91.12%
```

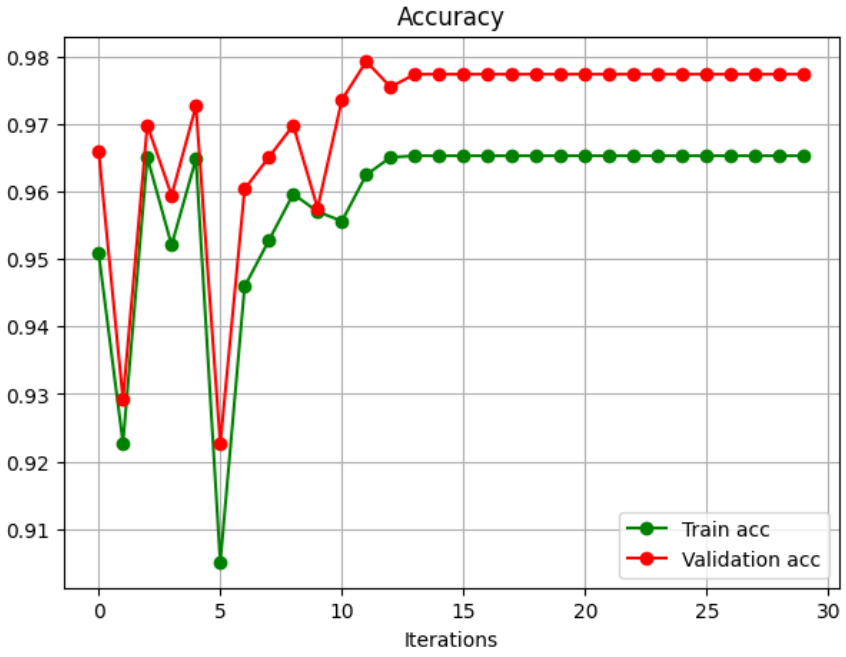
شکل ۳۸ بهترین دقت و تکرار متناظر با آن



شکل ۳۹نمودار دقت برحسب نرخ یادگیری

```
Iterations: 100%|████████| 100/100 [00:00<00:00, 102.04it/s]
The best outcome on training data is for learning rate 0.81 with accuracy of 96.41%
The best outcome on validation data is for learning rate 0.81 with accuracy of 96.03%
```

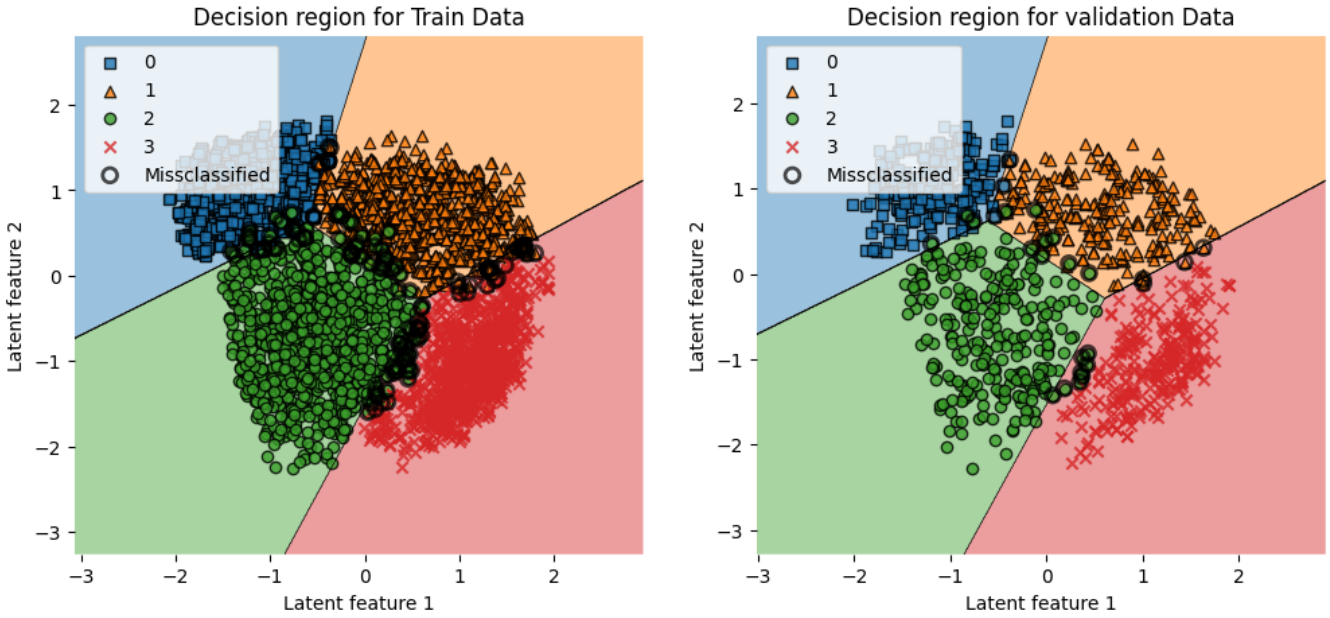
شکل ۴۰ بهترین دقت بدست آمده و نرخ یادگیری متناظر با آن



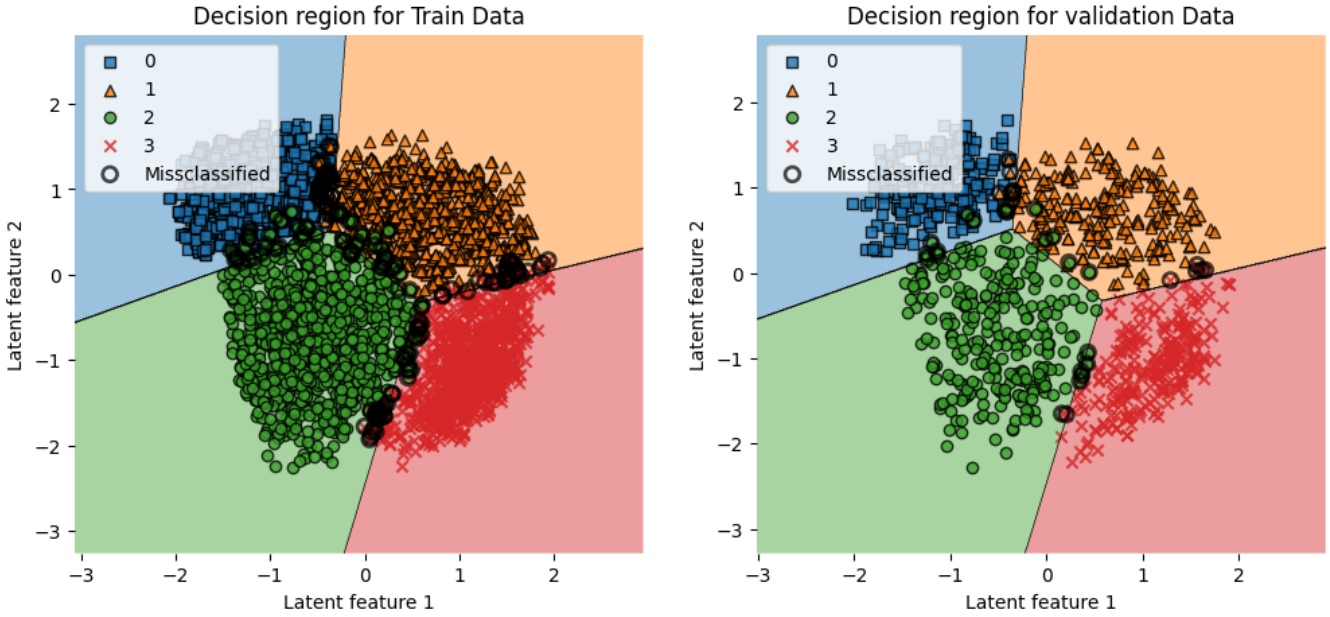
شکل ۴۱ نمودار دقت برحسب تکرار

```
Iterations: 100%|██████████| 30/30 [00:00<00:00, 58.63it/s]

The best outcome on training data is for iteration 13 with accuracy of 96.53%
The best outcome on validation data is for iteration 11 with accuracy of 97.92%
```

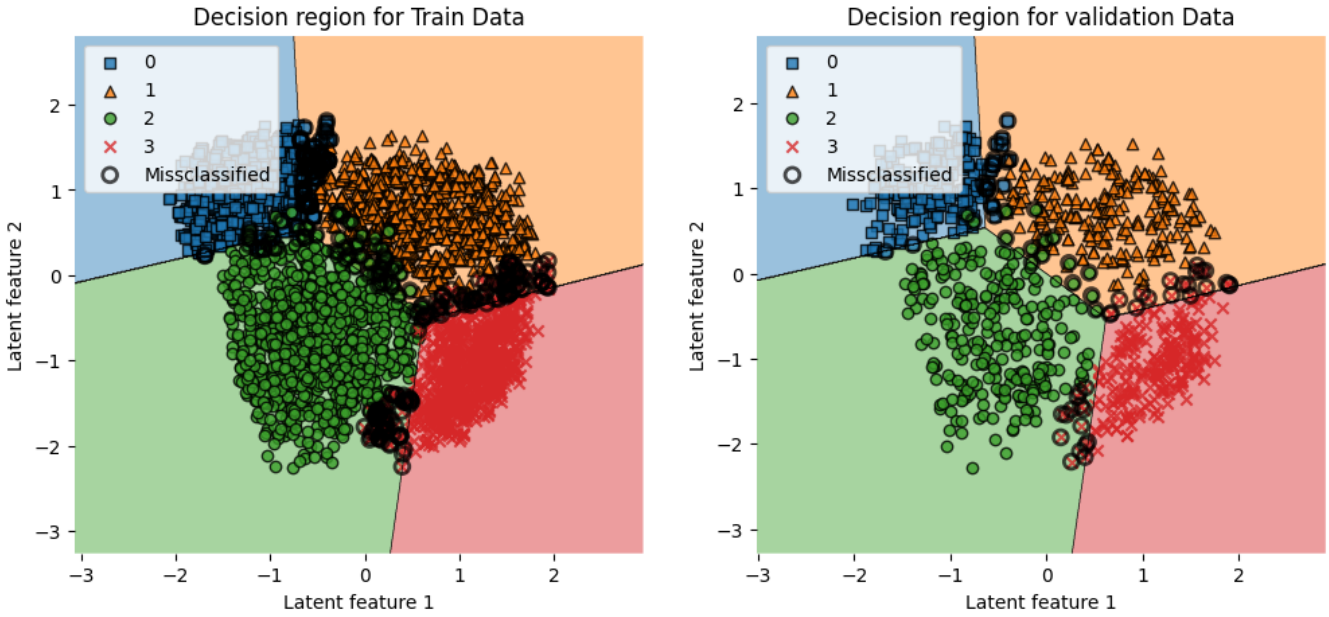
شکل ۴۲ بهترین دقت بدست آمده و تکرار متناظر با آن



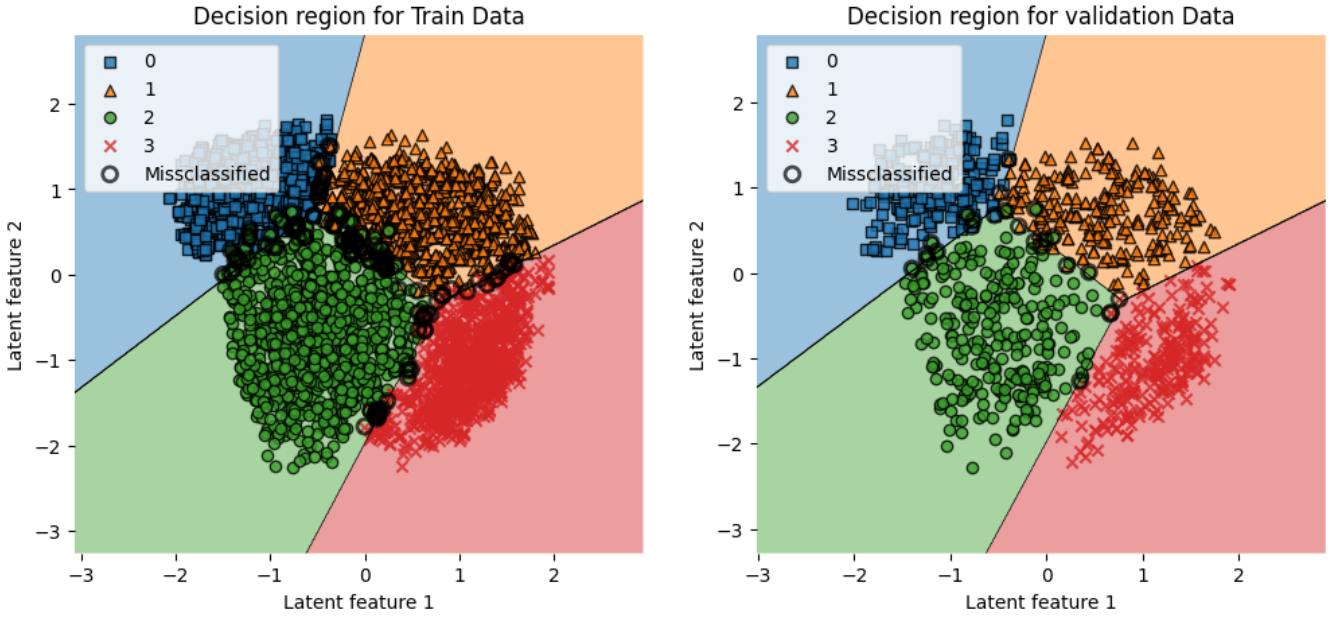
شکل ۴۳نواحی تصمیم‌گیری برای مدل Linear Rgression



شکل ۴۴ نواحی تصمیم‌گیری برای مدل SGD



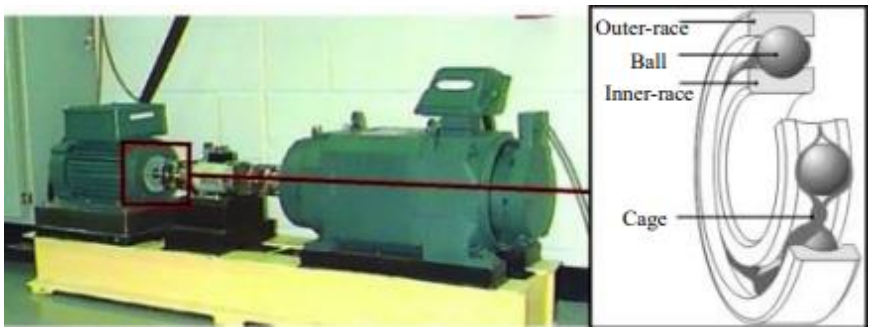
شکل ۴۵ نواحی تصمیم‌گیری برای مدل Perceptropn



شکل ۴۶ نواحی تصمیم‌گیری برای مدل Passive Agressive Classifier

سوال ۲

۲-۱ CWRT dataset



شکل ۴۷ پلتفرم آزمایش یاتاقان ها برای استخراج داده [1]

اهداف

این مجموعه داده توسط دانشگاه Case Western Reserve ایجاد شده است. این مجموعه داده، یک دیتاست متن-باز است که به عنوان اساس و مرجع اصلی بسیاری از مدل های یادگیری ماشین، که در حوزه تشخیص عیوب یاتاقان کار می کنند، است[1]. هدف از ایجاد این مجموعه داده، کمک به ایجاد مدل های تشخیص و پیشبینی عیب در یاتاقان ها می باشد.

ویژگی ها و حالات مختلف

این مجموعه متشکل از ۱۶۱ رکورد است که در چهار کلاس طبقه‌بندی شده اند:

۱. Normal-baseline با نرخ داده برداری ۴۸۰۰۰ نمونه در ثانیه
۲. عیب drive-end با نرخ داده برداری ۴۸۰۰۰ نمونه بر ثانیه
۳. عیب drive-end با نرخ نمونه برداری ۱۲۰۰۰ نمونه بر ثانیه
۴. عیب fan-end با نرخ نمونه برداری ۱۲۰۰۰ نمونه بر ثانیه

هر یک از کلاس های بیان شده در ادامه به دیتاست هایی با عیوب مختلف تقسیم می شود. این عیب ها می توانند شامل عیوب زیر باشند:

- عیب ساچمه
- عیب حلقه داخلی
- عیب حلقه خارجی؛ تقسیم‌بندی براساس موقعیت بار وارد شده:

- متمرکز[1]
- قائم
- متقابل[2][1]

دو رقم آخر شماره دانشجویی ۵۴ می باشد که باقی مانده آن بر ۴ برابر ۲ است. بنابراین فایل های Normal-2 و IR007-2 را دریافت کردیم.

۲-۲ کار با دیتاست

فایل های دانلود شده به فرمت mat. هستند. برای این که این فایل ها توسط پایتون خوانده و ذخیره شوند از دستور `loadmat` موجود در کتابخانه scipy استفاده کردیم.

```
Data in normal dataset is:
['X098_DE_time', 'X098_FE_time', 'X099_DE_time', 'X099_FE_time']

Data in fault dataset is:
['X107_DE_time', 'X107_FE_time', 'X107_BA_time']
```

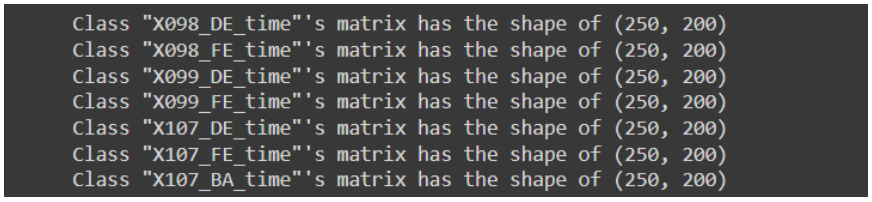
شکل ۴۸داده موجود در دیتاست ها

۲-۲-۱ استخراج داده

شکل ۴۸ تمامی حالت‌های موجود در دیتاست را به نمایش گذاشته است. در کل ۷ حالت داریم که می بایست برای هر کدام یک ماتریس MxN تشکیل دهیم. برای این کار از دو حلقه for تودرتو استفاده کردیم و تمامی ماتریس ها را در یک دیکشنری به نام matrices ذخیره نمودیم. M و N به صورت زیر در نظر گرفته شدند.

$$M = 250$$
$$N = 200$$

[1] Centered
[2] Opposite



شکل ۴۹ شکل ماتریس های ایجاد شده برای هر کلاس

اما از آنجایی که بای برای هر یک از دیتاست های نرمال و معیوب، یک کلاس داشته باشیم، یکی از کلاس های موجود را انتخاب کرده و محاسبات را ادامه می دهیم.(X**-DE-time را انتخاب می کنیم)

۲-۲-۲ *استخراج ویژگی*

تشخیص عیب یا هر کار دیگر مدل های یادگیری ماشین، با استفاده از داده انجام می شود. اما داده خام که مثلا توسط سنسور یا دوربین یا ... دارای اطلاعات اضافی و به‌درد نخور زیادی است. در واقع داده خام دارای نویز می باشد. نویز داده باعث بوجود آمدن اختلال در تصمیم‌گیری ماشین می شود؛ زیرا نمی تواند با کنار هم گذاشتن تمام اطلاعات موجود تصمیم درستی بگیری. برای این که بهبود عملکرد مدل، سعی می کنیم که اطلاعات اضافی و نویز را از داده ورودی حذف کنیم. یعنی از میان تمامی اطلاعات موجود، باید یک سری ویژگی استخراج کنیم که ماشین با استفاده از آن ها عملیات تصمیم گیری را انجام دهد.

تمامی ویژگی های موجود را محاسبه کردیم. برای استخراج ویژگی ها، بعضی از ویژگی ها را که در کتابخانه های numpy یا scipy، توابع از پیش تعریف شده داشتند، از آن توابع استفاده کردیم و برای محاسبه دیگر ویژگی ها، از فرمول های آنان استفاده کردیم. سپس برای هر کدام از ماتریس های نرمال و عیب، یک دیتاست جداگانه ایجاد کردیم. برای هر یک از دیتاست ها یک ویژگی جدید به نام "label"[۱] ایجاد کردیم و برای داده های نرمال عدد ۰ و برای داده های عیب عدد ۱ بود. سپس دو دیتاست را در راستای محور ۰ یا x به هم اضافه کردیم (یعنی سطر ها را به هم اضافه کردیم).

۲-۲-۳ *برزدن*

وظیفه یک مدل یادگیری ماشین در بحث طبقه بندی این است که یک کلاس مناسب برای ورودی خود اختصاص دهد. با استفاده از داده های آموزش، مدل رابطه ای بین ورودی و خروجی پیدا میکند. اما تمامی این یادگیری بر مبنای داده های

[۱] برچسب

آمورش است. یعنی توانایی طبقه‌بندی برای داده هایی است که مثل داده های آموزش باشند. به عبارت دیگر برای انجام آموزش و ارزیابی مدل ایجاد شده، نیاز داریم که داده ها پخش یکسانی داشته باشند.

تشکیل دیتاست جدید بدین صورت بود که ابتدا داده های نرمال قرار داده شدند و سپس داده های عیب. اگر تقسیم بندی آموزش و ارزیابی را در این حالت انجام دهیم، مثلا ۸۰ درصد ابتدایی را به آموزش و بقیه داده ها را به ارزیابی اختصاص دهیم، داده های ازریابی فقط دارای کلاس عیب خواهند بود. از طرفی داده های آموزش با این که هم داده های نرمال و هم داده های عیب را دارند ولی تعداد داده های هر کلاس یکسان نیست و عدم تعادل در داده آموزش وجود خواهد داشت. یعنی اگر برزدن را انجام ندهیم و داده های آموزش و ارزیابی را از هم جدا کنیم، پخش داده ها یکسان نبوده و نتایج مدل غیرقابل اعتماد خواهند بود.

برای برزدن و تقسیم کردن داده می‌توان دو روش را در پیش گرفت:

۱. ابتدا دیتاست را بر بزنیم و سپس داده را بدون برزدن جدا کنیم
۲. دیتاست برنخورد ولی هنگام تقسیم به دیتاست های آموزش و ارزیابی، `shuffle=True` قرار دهیم.

برای برزدن از روش دوم استفاده می کنیم. برای برزدن از تابع `train_test_split` از کتابخانه scikit-learn استفاده می کنیم.

```
The split rate is: Train 0.8 & Test 0.2
Size of Train data is:
X --> (400, 14)
y --> (400,)
Size of Test data is:
X --> (100, 14)
y --> (100,)
```

شکل ۵۰ تقسیم بندی داده ها

همانطور که در شکل ۵۰ مشاهده می شود، داده ها با نسبت ۰.۸ و ۰.۲ تقسیم شده اند و اندازه تمامی متغییر های ایجاد شده به نمایش درآمده است.

	standard deviation	peak	skewness	mean	absolute mean	root mean square	square root mean	kurtosis	crest factor	clearance factor	peak to peak	shape factor	impact factor	impulse factor	label
0	0.065162	0.179826	-0.102434	0.016275	0.051197	0.067005	0.041341	0.205579	2.683775	4.349826	0.355481	1.308760	1.308760	0.317890	0
1	0.065124	0.179826	-0.101089	0.016218	0.051140	0.066955	0.041293	0.211600	2.685784	4.354910	0.355481	1.309248	1.309248	0.317125	0
2	0.065046	0.179826	-0.095520	0.016050	0.050972	0.066839	0.041131	0.224659	2.690446	4.371997	0.355481	1.311286	1.311286	0.314875	0
3	0.065087	0.179826	-0.092039	0.015959	0.051063	0.066857	0.041302	0.216531	2.689711	4.353963	0.355481	1.309313	1.309313	0.312538	0
4	0.065158	0.179826	-0.091184	0.015882	0.051140	0.066907	0.041377	0.205829	2.687688	4.346023	0.355481	1.308321	1.308321	0.310557	0

شکل ۵۱ پنج سطر اول دیتاست

شکل ۵۱ پنج سطر اول دیتاست را به نمایش درآورده است. اگر به مقادیر موجود در ویژگی های Root Mean Square و Clearance Factor توجه کنیم، متوجه می شویم که این مقادیر، هم از نظر بازه خود مقادیر و از نظر بازه تغییراتشان با هم متفاوت هستند. در نتیجه ممکن است که تغییرات یکی، از تغییرات دیگری تاثیر بیشتری داشته باشد(اگر وزن ها برای هر دو ویژگی یکسان باشند). برای این که تأثیر همه ویژگی ها برای محاسبات ماشین یکسان باشد، بازه تغییرات تمامی ویژگی ها را یکسان می کنند؛ به طور معمول بازه همه متغییر ها به [0,1] یا [-1,1] انتقال داده می شود.

دو روش استفاده شده برای نرمال سازی داده ورودی:

- Standardization: $x_{new} = \frac{x-\mu}{\sigma}$; $\mu: mean,\ \ \sigma: standard\ deviation$

در روش بالا، مقدار میانگین از تمامی مقادیر کم شده و سپس بر انحراف معیار تقسیم می شوند. با این روش اعداد بین -۱ و ۱ قرار می گیرند.

- MinMaxScaler: $x_{new} = \frac{x-x_{min}}{x_{max}-x_{min}}$

در این روش با یک تبدیل، کمترین مقدار ۰ میشود و با تقسیم تمامی اعداد بر بازه اعداد، همه مقادیر بین ۰ و ۱ قرار می گیرند.

در این بخش از روش Standardization استفاده شد. با استفاده از scaler.fit(x_train)، پارمترهای مقیاسگر[۱] با توجه به داده های آموزش تنظیم می شوند و سپس از آن مقادیر استفاده شده تا داده های آموزش و ارزیابی نرمالسازی شوند.

[۱] Scaler

نرمال سازی کردن داده های ارزیابی به این دلیل است که پخش داده های ارزیابی هم مثل داده های آموزش شود. همانند برزدن داده ها، هدف این بود که پخش داده های ارزیابی و آموزش یکسان باشد، در نرمال سازی هم برای یکسان کردن پخش داده ها استفاده می شود.

۳-۲ ایجاد مدل طبقه‌بندی از صفر

از آنجایی که باید تمامی محاسبات را به صورت از ۰ تا صد توسط کد انجام دهیم و ریاضیات را پیاده‌سازی کنیم، از کتابخانه numpy استفاده می کنیم. مدل های استفاده شده به صورت زیر می باشند:

- **مدل یادگیری ماشین:** Logistic Regression
- **تابع هزینه:** $J = \frac{1}{N}\sum_{i=1}^{N} \underbrace{(\hat{y} - y)^2}_{e_i}$
- **تابع یادگیری:** $w := w \ - \eta \sum_{i=1}^{N} e_i x_i$

در ادامه به توضیح نحوه پیاده‌سازی مدل یادگیری ماشین می پردازیم.

به صورت خلاصه، مدل خام به صورت یک کلاس در پایتون تعریف شد و با استفاده از آن آموزش صورت گرفت.

تعریف کلاس و مقادیر اولیه

```
class LogisticRegression:

    def __init__(self, n_iter=10, learning_rate=0.01, random_state=None):
        self.n_iter = n_iter
        self.eta = learning_rate
        np.random.seed(random_state)
```

متد برای مقداردهی اولیه وزن ها

```
    def _weight_init(self):
        self.w = np.random.rand(14)*0
        self.b = np.random.rand()
```

متد پیاده‌سازی الگوریتم یادگیری (بروزرسانی وزن ها)

```
    def _update(self,x,E):
```

```
        dj = np.dot(x.T,E)
        self.w += self.eta*dj
        self.b += self.eta*np.sum(E)
```

متد برای تبدیل ورودی مدل به مقدار پیش‌بینی (اختصاص برچسب به ورودی)

```
    def forward(self,x):
        z = np.dot(x,self.w) + self.b
        a = 1/(1+np.e**(-z))
        y_hat = np.array([1 if hat>.5 else 0 for hat in a])
        return y_hat
```

متد محاسبه خطا و هزینه

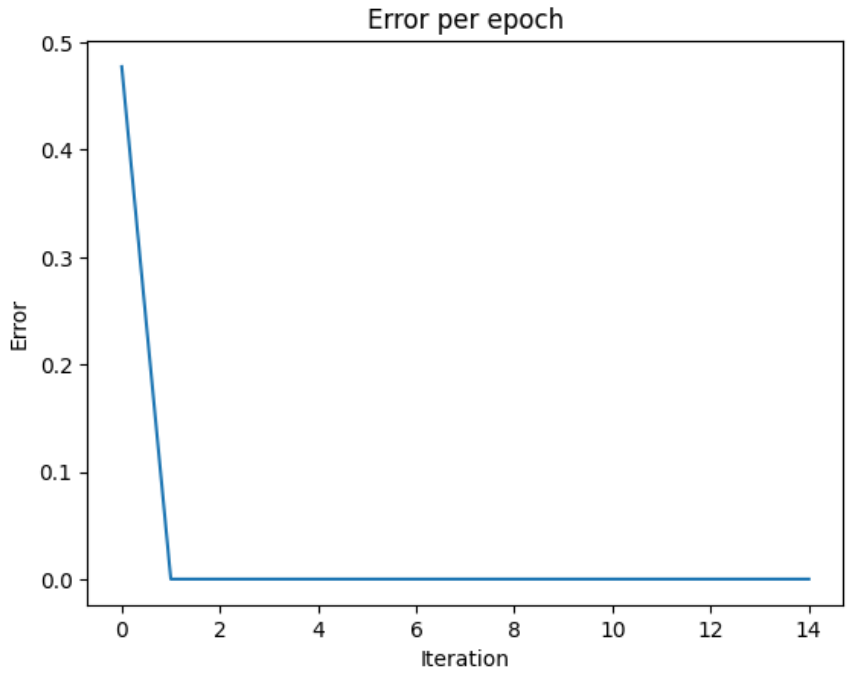
```
    def error(self,predict,true):
        E = true - predict
        e = 1/len(true) * np.dot(E,E)
        return E,e
```

متد آموزش، در این بخش با بهره گیری از متدهای قبل، عمل آموزش انجام می شـــود و خطا هر مرحله نیز ذخیره می شود

```
    def train(self,x,y):
        self._weight_init()
        self.loss = []
        for iter in range(self.n_iter):

            y_hat = self.forward(x)
            E,e = self.error(y_hat,y)
            self.loss.append(e)
            self._update(x,E)
```

ایجاد مدل و اعمال داده های آموزش به آن

```
# Train model with data
model = LogisticRegression(
    n_iter = 15,
    learning_rate =0.0001,
    random_state = 25
)
model.train(x_train_scaled,y_train)
```



شکل ۵۲ نمودار خطا

شـکل ۵۲ نمودار خطا مدل به هنگام آموزش را به نمایش می گذارد. همانطور که مشـاهده می شـود بعد از اولین تکرار مدل، خطا به صفر رسید و مدل توانست بهترین وزن های ممکن را محاسبه کند.

برای ارزیابی عملکرد مدل روی داده تست، از دو معیار دقت[۱] و f1-score استفاده کردیم.

```
Accuracy on test data is 100.0%
F1-score on test data is 100.0%
```

شکل ۵۳ معیار های محاسبه شده روی داده تست

شکل ۵۳ عمکلرد مدل را روی داده ارزیابی به نمایش گذاشته است.

برای نظر دادن درباره عملکرد مدل نمی توان تنها با اتکا به نمودار خطا آموزش عمل کرد. باید حتما از یک دسـته دیگر داده برای ارزیابی عملکرد مدل استفاده کنیم. مدل با هر بار تکرار و بروزرسانی وزن ها، ماشینی ایجاد می کند که بهتر بتواند داده های آموزش را طبقه‌بندی کند. پس همیشـه و با هر بار تکرار، مدل تنیجه بهتری را روی داده آموزش خواهد داشـت. اما هدف

[۱] Accuracy

یادگیری ماشین این نیست که مدلی ایجاد کنیم تا داده های آموزش را به خوبی تشخیص دهد. هدف این است که هر داده ای را به خوبی طبقه‌بندی کند. یعنی به دنبال خاصیب عمومیت‌بخشی[1] هستیم. آموزش بیش‌ازحد مدل باعث می شود که مدل نویز های موجود در داده آموزش را یاد بگیرد. مدل با آموزش زیاد، داده آموزش را حفظ می کند درحالی که باید رابطه مناسبی بین ورودی و خروجی آموزش پیدا کند. در کل باید علاوه بر کمینه کردن خطا آموزش، خطا ارزیابی را هم کمینه کنیم تا از پدیده over-fitting جلوگیری نماییم.

برای جلوگیری از پدیده over-fitting می توانیم از روش های زیر استفاده کنیم:

- آموزش مدل با تکرار مناسب (تکرار زیاد باعث ایجاد over-fitting و تکرار کم باعث بروز under-fitting می شود.)
- اعمال روش های regularization

۴-۲ استفاده از مدل آماده scikit-learn

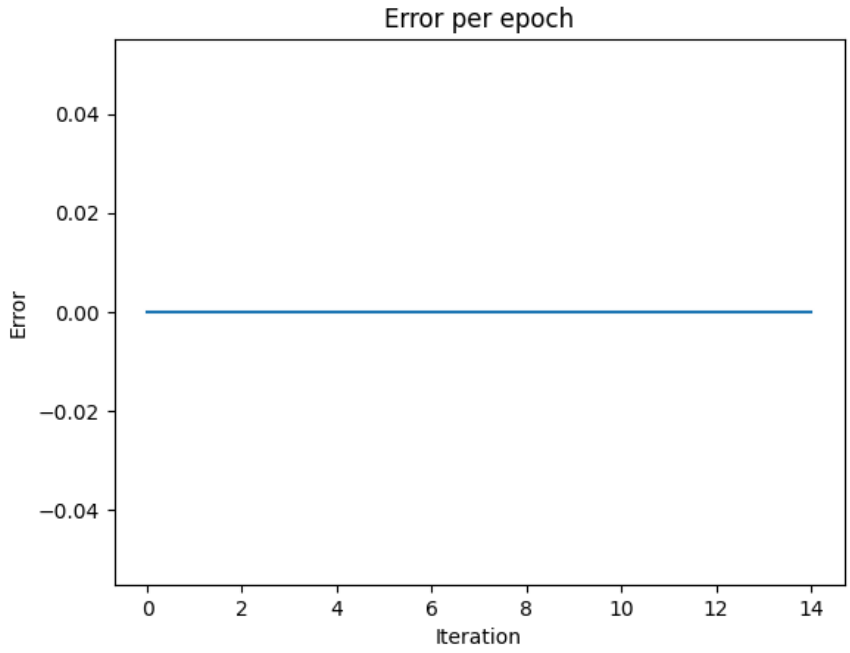
برای این بخش، از الگوریتم SGD استفاده می کنیم.

```
Accuracy on test data is 100.0%
F1-score on test data is 100.0%
```

شکل ۵۴ نتایج ارزیابی مدل SGD Classifier

همانطور که از شکل ۵۴ مشخص است، تنایج ارزیابی مدل SGD هم همانند مدل خودمان بود.

[1] Generalization



شکل ۵۵ نمودار خطا بر حسب تعداد تکرار

با توجه به شکل ۵۵، می بینم که خطا در تمامی تکرار ها برابر ۰ بوده است.

کلاس SGDClassifier به صـــورت خودبه‌خود، آرایه ای برای ذخیره ســـازی خطا در تکرار ها به ما نمی داد. ولی با استفاده از متد `partial_fit()`، در هر تکرار خطا را محاسبه کرده و در یک آرایه ذخیره نمودیم. برای پیاده‌سازی این ایده، یک زیرکلاس[۱] از زیرمجموعه کلاس `sklearn.linear_model.SGDClassifier` ایجاد کرده و ویژگی loss_history را در آن تعریف کردیم. این ویژگی با استفاده از متد `partial_fit()` به همراه محاسبات خطا، خطا را در تکرار در خود ذخیره می کرد.

سوال ۳

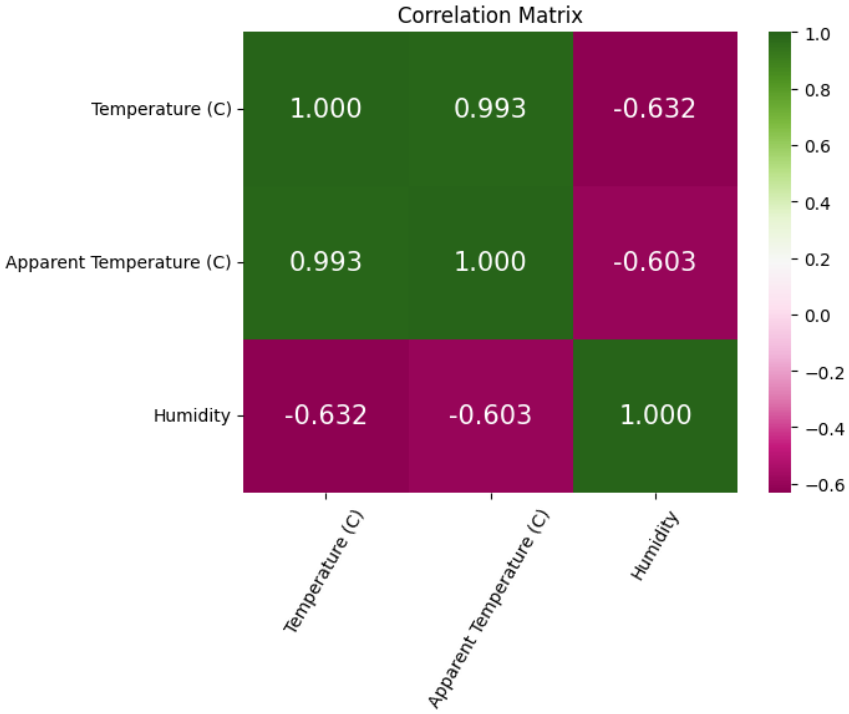
ابتدا داده را از مرجع دریافت کردیم و فایل را با استفاده از تابع `read_csv` از کتابخانه pandas بارگذاری می کنیم.

۳-۱ هیتمپ و هیستوگرام

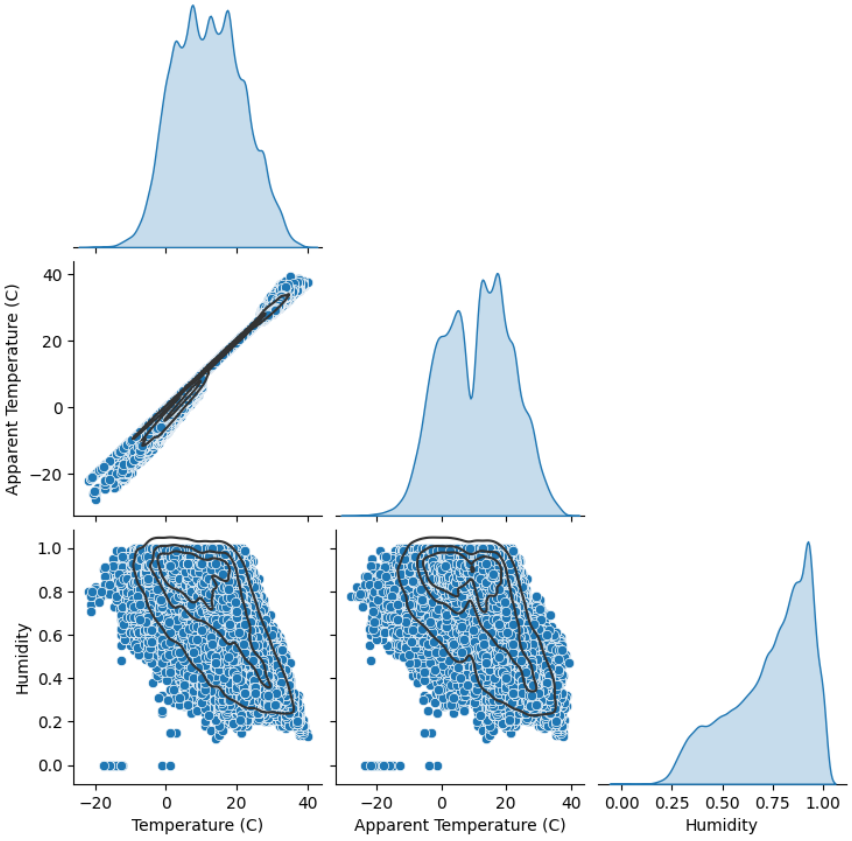
برای انجام محاسبات از ویژگی ها Apperant temperature و Temprature و Humidity استفاده میکنیم.

---

[۱] Subclass



شکل ۵۶ هیتمپ ماتریس همبستگی



شکل ۵۷ هیستوگرام و پراکندگی داده ها

شکل ۵۷ هیستوگرام و ارتباط بین متغییر های مختلف را نمایش می دهد. در ادامه برای هر یک از سه حالت موجود یک مدل regressionایجاد می کنیم ولی با توجه به حالاتی که در شکل ۵۷ مشاهده می شود، مدل فقط برای حللت بین Apparent temperature و Temperature قابل اعتماد بوده و خطا کمی دارد.

۳-۲ اعمال تخمین روی داده ها

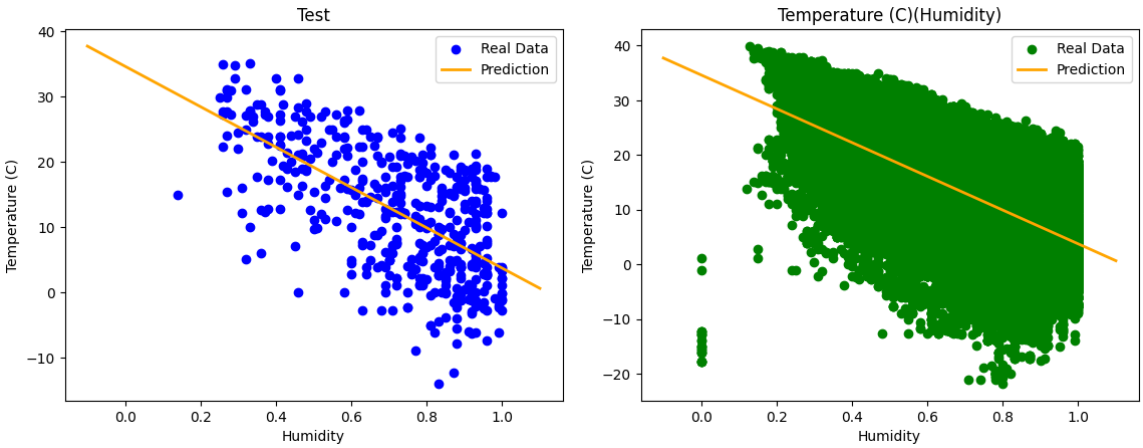
۳-۲-۱ LS

با استفاده از کد زیر، یک کلاس برای مدل LS طراحی می کنیم تا با استفاده از آن، محاسبات را انجام دهیم.

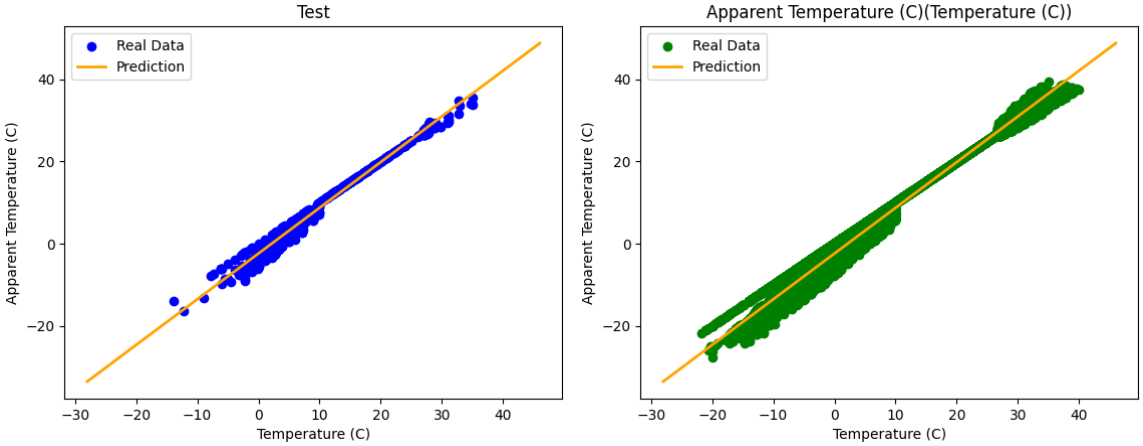
```
class LinearRegressionLS:
    def __init__(self):
        self.coef = None

    def fit(self,x,y):
        x = np.column_stack((np.ones(len(x)),x))
        self.coef = np.linalg.inv(np.dot(x.T,x)).dot(x.T).dot(y)

    def predict(self,x):
        x = np.column_stack((np.ones(len(x)),x))
        return np.dot(x,self.coef)
```

با استفاده از کلاس تعریف شده، برای هر سه ویژگی، به صورت دوبه‌دو، یک مدل ایجاد کرده و خروجی هر کدام را به نمایش می گذاریم.

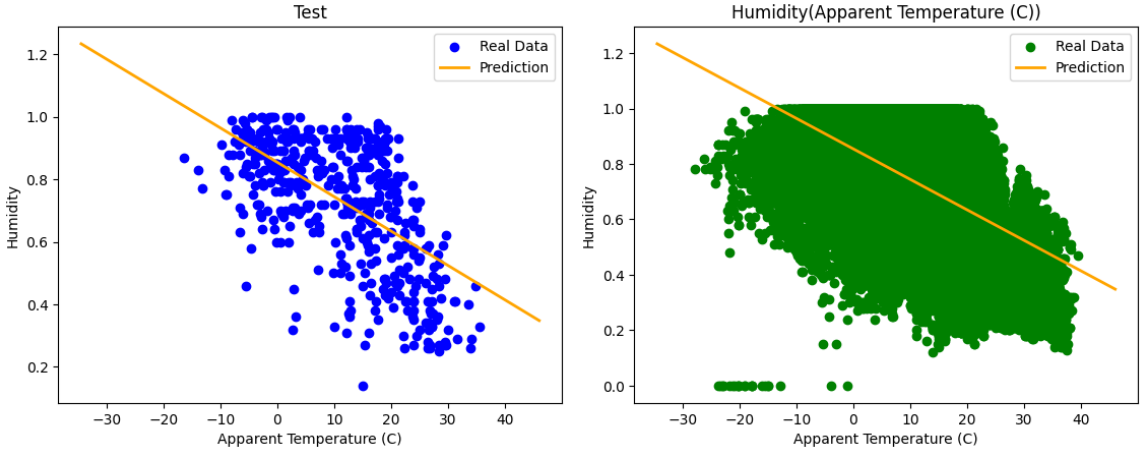
برای ارزیابی بهتری مدل سیستم، داده ها به دو قسمت آموزش و ارزیابی تقسیم شدند. علاوه بر تقسیم دیتا، داده ها نرمالایز هم شدند.



شکل ۵۸ مدل *Temperature* برحسب *Humidity*



شکل ۵۹ مدل *Apparent Temperature* برحسب *Temperature*



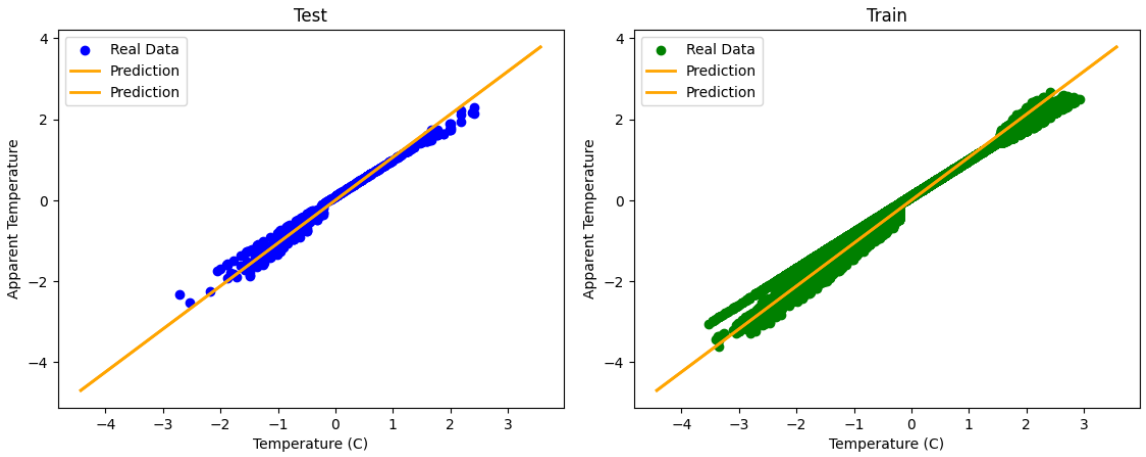
شکل ۶۰ مدل *Humidity* برحسب *Apparent Temperature*

جدول۱ خطا مدل LS

	MSE Train	MSE Test	MAE Train	MAE Test
Temperature – Humidity	0.600226	0.620935	0.631732	0.647557
Apparent Temperature – Temperature	0.014688	0.015153	0.092799	0.094164
Humidity – Apparent Temperature	0.636896	0.691355	0.646953	0.669909

با توجه به جدول ۱، کمترین خطا مربوط به مدل Apparent Temperature و Temperature می باشد. همانطور که قبلا هم بیان شد، با توجه به پراکندگی داده ها در شکل ۵۷، به دلیلی پراکندگی کم داده های Apparent Temperature و Temperature نسبت به هم در مقایسه با دیگر حالات، انتظار داشتیم که خطا این حالت کمتر از دیگر حالات باشد و بقیه حالات خطا بسیار زیادی داشته و مدل عملکرد خوبی نخواهد داشت.

درحالت قبل، روش LS را برای هر سه ویژگی و به صورت دوبه‌دو اعمال کردیم. یعنی ورودی یک ویژگی و خروجی نیز یک ویژگی بود. حال بدین صورت مدل را ایجاد می کنیم که ورودی ها Teperature و Humidity باشند و خروجی نیز ویژگی Apparent Temperature باشد.



شکل ۶۱ نمودار Apparent Temperature برحسب Temperature

جدول۲ خطا LS (حالت دوم)

	MSE Train	MSE Test	MAE Train	MAE Test
Apparent Temperature – Temperature	0.01364	0.014259	0.088310	0.089807

با مقایسه جداول ۱ و ۲، مشاهده می کنیم که خطا بدست آوردن Apparent Temperature در حالتی که ورودی مدل دو وِیژگی می باشد، کمتر از حالتی است که Apparent Temperature را صرفا یا استفاده از یک ویژگی محاسبه می کنیم.

RLS ۳-۲-۲

کلاس تعریف شده مدل به صورت زیر می باشد.

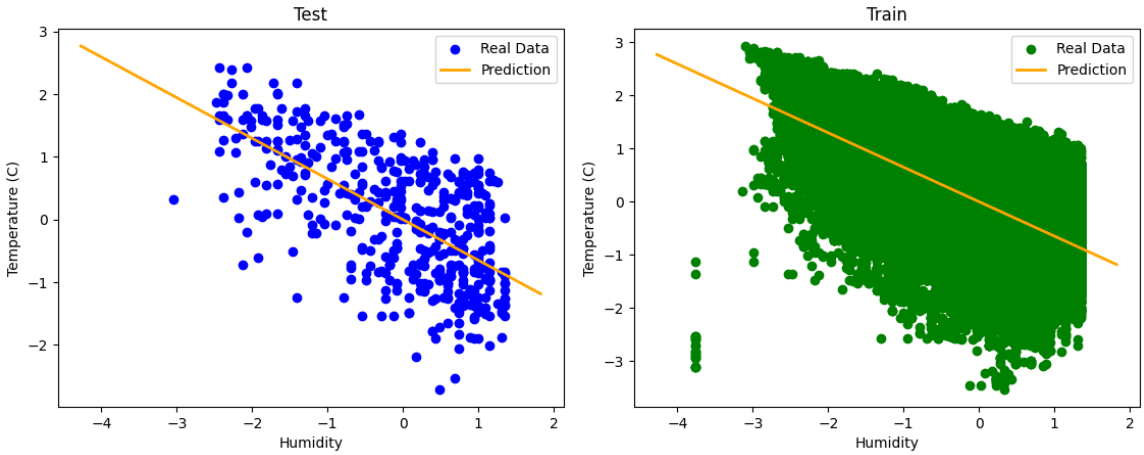
```
class RecursiveLeastSquares:
    def __init__(self, n_features, forgetting_factor=0.99):
        self.n_features = n_features
        self.forgetting_factor = forgetting_factor
        self.theta = np.zeros((n_features, 1))  # Initialize model parameters
        self.P = np.eye(n_features)  # Initialize covariance matrix

    def fit(self, X, y):
        errors = []
        for i in range(len(X)):
            x_i = X[i].reshape(-1, 1)
            y_i = y[i]

            # Predict
            y_pred = np.dot(x_i.T, self.theta)

            # Update
            error = y_i - y_pred
            errors.append(error)
            K = np.dot(self.P, x_i) / (self.forgetting_factor + np.dot(np.dot(x_i.T,
self.P), x_i))
            self.theta = self.theta + np.dot(K, error)
            self.P = (1 / self.forgetting_factor) * (self.P - np.dot(K, np.dot(x_i.T,
self.P)))

        return errors
```

```
    def predict(self, X):
        return np.dot(X, self.theta)
```

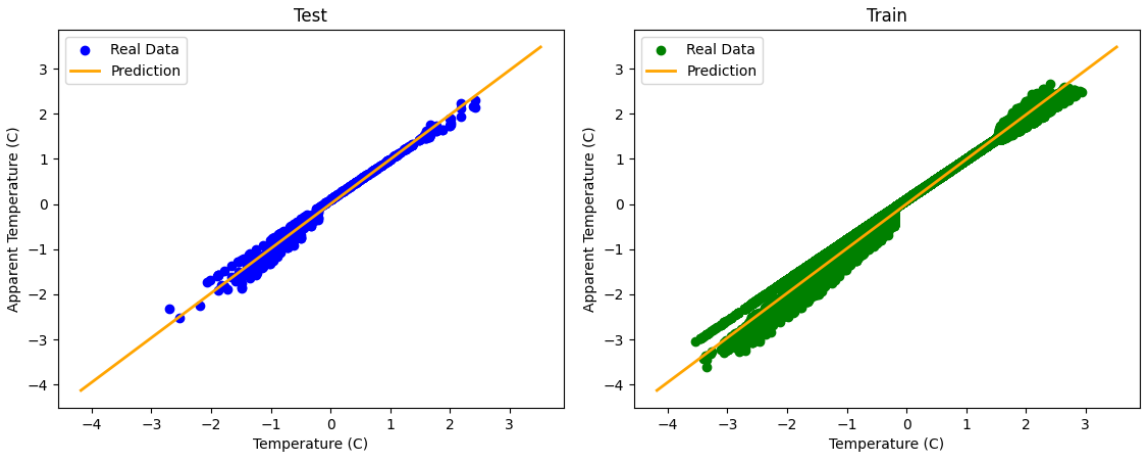
پارامترها:

$$n_{features} = 2, ForgettingFactor = 0.9999$$

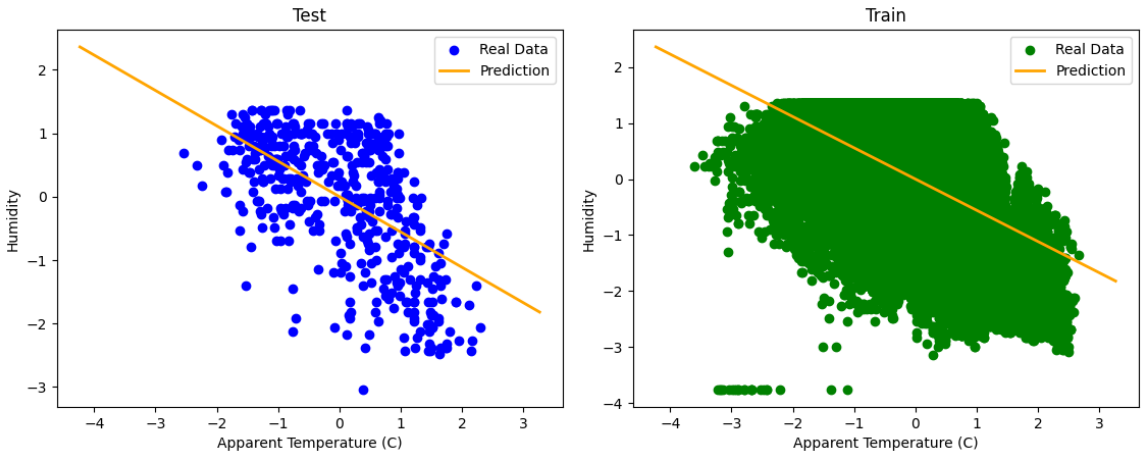
با اســتفاده از تقســیم داده ای که در بخش قبل انجام شــد، برای این قســمت نیز مدل را برای حالات مختلف ارزیابی می کنیم.



شکل ۶۲ نمودار Temperature برحسب Humidity



شکل ۶۳ نمودار Apparent Temperature برحسب Temperature



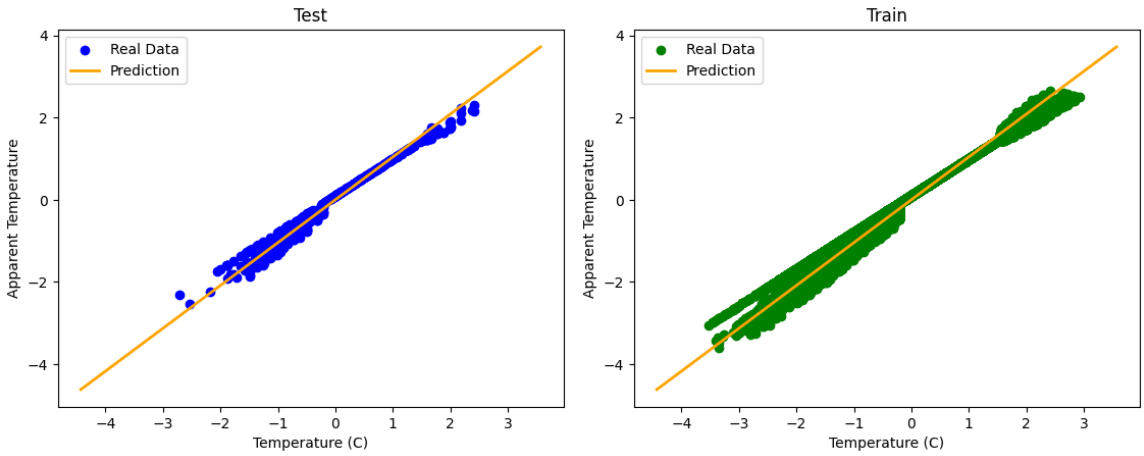
شکل ۶۴ نمودار Humidity برحسب Apparent Temerature

جدول۳ خطا روش RLS

	MSE Train	MSE Test	MAE Train	MAE Test
Temperature – Humidity	0.600467	0.621896	0.630839	0.645957
Apparent Temperature – Temperature	0.014697	0.015151	0.092924	0.094339
Humidity – Apparent Temperature	0.6389	0.694733	0.650073	0.672374

همانند روش قبل، همانطور که انتظار می رفت، حالت Apparent Temperature-Temperature به دلیلی حالت خطی ای که دارد، دارای کمترین خطا می باشد.

حال Apparent temperature را با استفاده از دو متغیر Humidity و Temperature محاسبه می نماییم.



شکل ۶۵ نمودار Apparent Teperature برحسب Teperature

جدول۴ خطا روش RLS (حالت دوم)

	MSE Train	MSE Test	MAE Train	MAE Test
Apparent Temperature – Temperature	0.013643	0.014259	0.088310	0.089807

۳-۳ Weighted Least Squares

در روش WLS فرض می کنیم که یک واریانس ثابتی در خطا وجود دارد.

$$\boldsymbol{Y} = \boldsymbol{X\theta} + \boldsymbol{\epsilon}^*$$

فرض می شود که $\boldsymbol{\epsilon}^*$ دارای پخشی نرمال و میانگین صفر می باشد. از طرفی ماتریس واریانس-کواریانس آن غیر ثابت می باشد.

ماتریس کواریانس :
$$S = \begin{pmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{pmatrix}$$

بر این اساس، ماتریس وزن به صورت زیر تعریف می شود:

$$W = \begin{pmatrix} \frac{1}{\sigma_1^2} & 0 & \dots & 0 \\ 0 & \frac{1}{\sigma_2^2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{1}{\sigma_n^2} \end{pmatrix}$$

در این حالت تقریب حداقل مربعات وزن دار به صورت زیر می باشد:

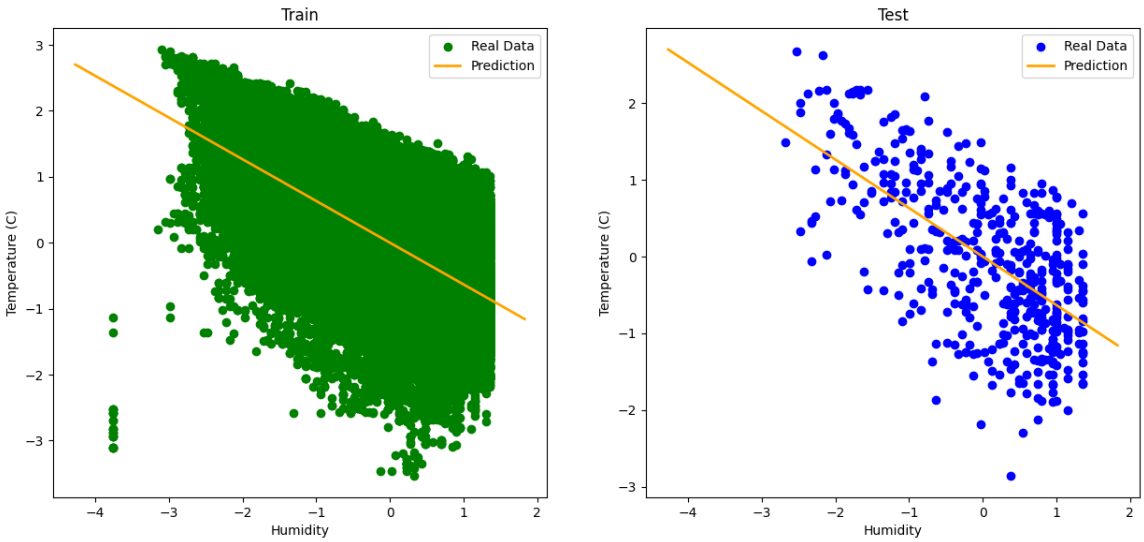
$$\hat{\theta} = \underset{\theta}{argmin} \sum_{i=1}^{N} \epsilon^* = (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{Y}$$

برخلاف بخش قبلی، در این بخش از کتابخانه statsmodels اســـتفاده کردیم که مدلی آماده برای الگوریتم WLP در اختیار ما قرار می گذاشت.

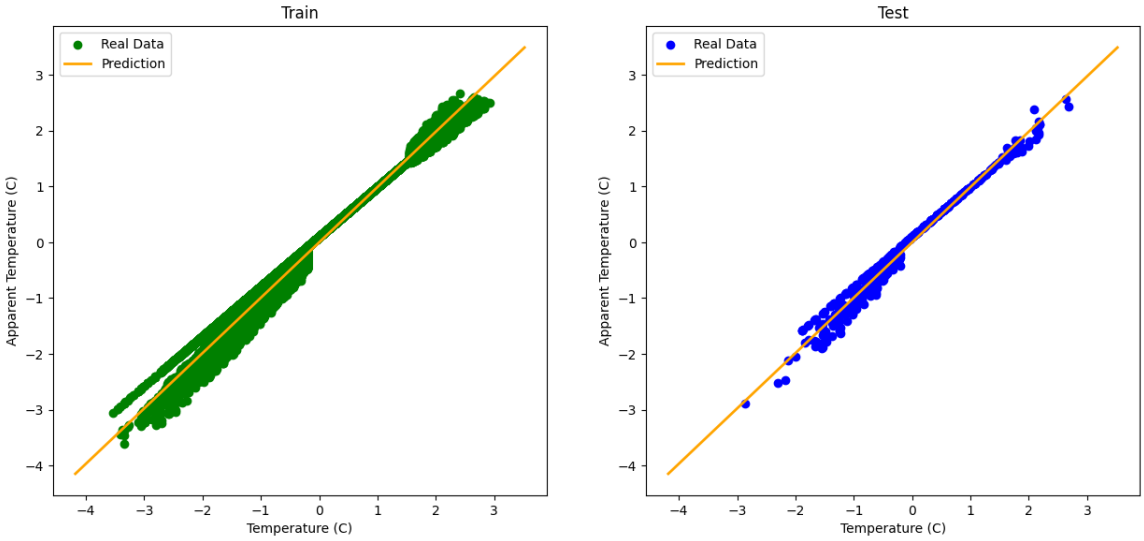
کلاســـی تعریف شـــده که به صـــورت خودکار ورودی ها را بگیرد و تمامی مراحل اولیه را روی داده انجام داده و پس از آموزش مدل، وزن ها را ذخیره کند.

```
import statsmodels.api as sm

class WeightedLeastSquares:
    def __init__(self):
        self.weights = None
        self.model = None

    def fit(self, X, y):
        # Calculate the weights as 1/var(y)
        self.weights = 1 / np.var(y)

        # Fit the WLS model
        X_with_intercept = sm.add_constant(X)  # Add constant term for intercept
        self.model = sm.WLS(y, X_with_intercept, weights=self.weights)
        self.results = self.model.fit()

    def predict(self, X):
        # Add constant term for intercept
        X_with_intercept = sm.add_constant(X)
        return self.results.predict(X_with_intercept)
```

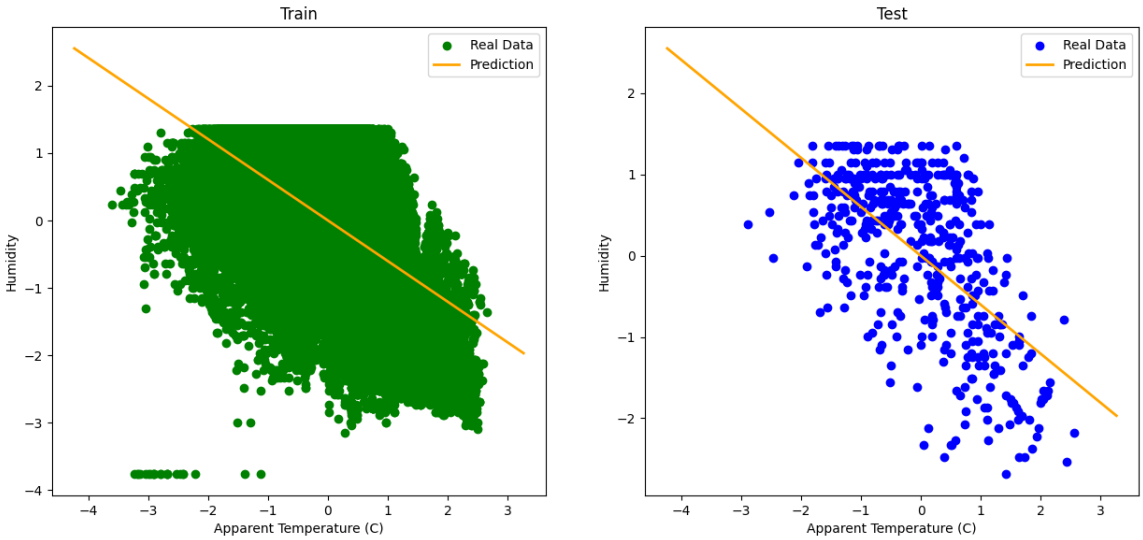
نتایج مدل بعد از آموزش روی ویژگی ها به صورت دوبه‌دو



شکل ۶۶ نمودار Temerature برحسب Humidiy



شکل ۶۷ نمودار Apparent Temperature برحسب Temperature

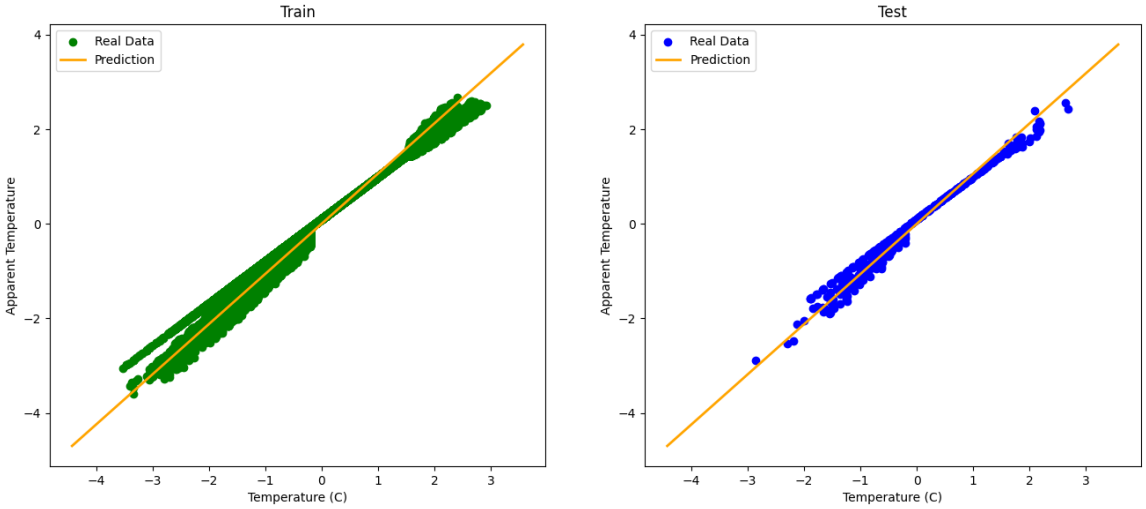


شکل ۶۸ نمودار Humidity برحسب Apparent Temperature

جدول۵ خطا روش WLS

	MSE Train	MSE Test	MAE Train	MAE Test
Temperature – Humidity	0.600675	0.527115	0.632006	0.598671
Apparent Temperature – Temperature	0.014701	0.012222	0.092843	0.084752
Humidity – Apparent Temperature	0.637333	0.570876	0.647182	0.61513

حال مدل را برای حالتی که آموزش می دهیم که ورودی Temperature و Humidity باشـــد و خروجی نیز Apparent Teperature.



شکل ۶۹ نمودار Apparent Teperature بر حسب Temperature

جدول ۶ خطا روش WLS (حالت دوم)

	MSE Train	MSE Test	MAE Train	MAE Test
Apparent Temperature – Temperature	0.013641	0.014379	0.088304	0.090560

در حالت دوم، با این که خطا روی داده آموزش کاهش پیدا کرده است، اما خطا ارزیابی بیشتره شده است.

۳-۴ QR-decomposition-based RLS

RLS مدلی تطبیقی از نوع رگرسیون خطی می باشد. این مدل این قابلیت را دارد که با اضافه شدن مشاهده جدید، وزن ها را تغییر دهد؛ یعنی با هر مشاهده جدید، مدل عوض می شود. بروزرسانی وزن ها به صورت زیر می باشد:

$$\theta(t) = \theta(t-1) - \alpha < x(t-1), \theta + tz > z + tz$$

$$z = Ax(t)$$

$$\alpha = \left(1 + < x(t), (X^T X)^{-1} x(t) > x(t)\right)^{-1}$$

$$A = (X^T X)^{-1}$$

که در عبارت بالا، $x(t)$ نمایانگر مشاهده جدید، $\theta(t-1)$وزن قبلی و $\theta(t)$ وزن جدید می باشند.

از طرفی طبق روش QR-decomposition یک ماتریس مربعی مثل A را میتوان به صورت حاصل ضرب دو ماتریس درنظر گرفت:

$$A = QR$$

که ماتریس Q یک ماتریس مربعی و متعامد و ماتریس R یک ماتریس بالا مثلثی می باشند.

میتوان از خاصیت ماتریس های QR-decomposition استفاده کرد تا به هنگام بدست آوردن معکوس ماتریس ها، محاسبات کمتری انجام شود. برای محاسبه $(X^T X)^{-1}$ خواهیم داشت:

$$X = QR \Rightarrow (X^T X)^{-1} = \left(R^T \underbrace{Q^T Q}_{I} R \right)^{-1} = (R^T R)^{-1}$$

ماتریس R یک ماتریس بالامثلثی می باشد. $R^T R$ هم یک ماتریس بالا مثلثی است و معکوس آن هم یک ماتریس بالا مثلثی خواهد بود. به همین دلیل هم محاسبات کمتری نیاز خواهیم داشت.