

وزارت علوم، تحقیقات و فناوری
دانشگاه تحصیلات تکمیلی علوم پایه
دانشکده علوم کامپیوتر و فناوری اطلاعات



پیاده سازی الگوریتم های، لجستیک رگرسیون، لجستیک رگرسیون وزنی (Naïve Bayes (با فرض و بدون فرض Generative رویکرد و لجستیک رگرسیون بیزی و SVM

موضوع پژوهش درس یادگیری ماشین

فصل زمستان ۱۳۹۸

استاد: دکتر پروین رزاقی

دانشجویان:

بهنام مردانی ۹۸۴۱۱۴

کامران عبدالی ۹۸۴۱۰۱

فهرست مطالب

۴	فصل اول - مقدمه
۴	شرح پژوهه
۵	تعاریف اولیه
۶	شرح مختصر الگوریتمها
۶	الگوریتم Logistic Regression
۶	الگوریتم Locally Weighted Regression (LWR)
۷	الگوریتم رویکرد Generative با/بدون فرض Naïve Bayes
۷	الگوریتم Bayesian Logistic Regression
۱۰	الگوریتم ماشین بردارهای پشتیبان (SVM)
۱۱	معیارهای ارزیابی
۱۱	صحت (Precision)
۱۱	حساسیت (Recall)
۱۱	معیار ارزیابی F-Measure
۱۲	معیار ROC
۱۲	معیار AUC
۱۳	فصل دوم - ابزارهای استفاده شده برای پیاده سازی
۱۴	فصل سوم - نحوه اجرای برنامه
۱۶	فصل چهارم - توضیح کد
۱۶	لجستیک رگرسیون ساده و وزنی
۱۷	تابع پیاده سازی شده در سوپر کلاس LogisticRegression
۱۸	زیرکلاس SimpleLogisticRegression
۱۹	زیرکلاس WeightedLogisticRegression
۱۹	مجموعه داده MSRC
۲۰	مجموعه داده VOC
۲۰	مجموعه داده PIE
۲۲	بیزین لجستیک رگرسیون
۲۲	کلاس BayesianLogisticRegression

۲۴	مجموعه داده MSRC
۲۴	مجموعه داده VOC
۲۵	مجموعه داده PIE
۲۶	ماشین‌های بردار پشتیبان (SVM)
۲۶	کلاس SVM
۲۸	کلاس PIESVM
۲۹	مجموعه داده MSRC
۲۹	مجموعه داده VOC
۲۹	مجموعه داده PIE
۳۰	رویکرد Generative (با/بدون فرض Naïve Bayes)
۳۴	مجموعه داده MSRC
۳۴	مجموعه داده VOC
۳۵	مجموعه داده PIE
۳۶	فصل پنجم - نتایج ارزیابی
۳۷	فصل ششم - گام دوم پژوهه

فصل اول - مقدمه

شرح پژوهش

- در گام اول بایستی الگوریتم‌های:

Logistic Regression .a

Weighted Logistic Regression .b

Generative (With and without Naïve Bayes Assumption) .c

Bayesian Logistic Regression .d

SVM .e

را بر روی سه مجموعه داده استانداردی که همراه با تمرین فرستاده می‌شود، اعمال نمایید و نتایج را گزارش نمایید. توضیح مختصری در مورد مجموعه داده‌ها در انتهای این داکیومنت آورده شده است. برای گزارش نتایج ارزیابی از AUPR، AUC-ROC، Precision-Recall، F1-measure استفاده نمایید. در گزارش تمرین بایستی هر کدام از معیارها را تعریف نمایید و بیان کنید که هر کدام از این معیارها در چه شرایطی بیشتر مورد استفاده قرار می‌گیرند. دقت کنید که در گزارش پژوهش بایستی تمامی الگوریتم بر روی هر مجموعه داده از دیدگاه معیارهای ارزیابی متقاوت مورد بررسی و تحلیل قرار بگیرند. برخی از الگوریتم‌ها نیازمند تنظیم هایپر پارامتر هستند که بایستی با استفاده از تکنیک‌هایی که در کلاس بحث شد مقدار آنها تنظیم گردد.

- در گام دوم بایستی راه حلی برای ترکیب الگوریتم‌های استفاده شده در مرحله قبل ارائه دهید به گونه‌ای که باعث افزایش دقت گردد. بدین منظور در حوزه یادگیری ماشین حوزه‌ی Ensemble Learning ارائه شده است که می‌توانید از این ایده‌ها نیز بهره بگیرید. برخی از لینکهایی که می‌تواند اطلاعات مفیدی در اختیار شما بگذارد در ادامه آورده شده است:

<http://tjzhifei.github.io/links/EMFA.pdf>

<http://people.csail.mit.edu/dsontag/courses/ml12/slides/lecture12.pdf>

همانطور که در کلاس نیز اشاره شد در صورتی که دانشجویان بتوانند در این مرحله ایده و پیشنهاد جدیدی ارائه دهند تا سقف ۲ نمره، نمره اضافه تری دریافت خواهند کرد.

در طی مراحل انجام پژوهش نکات زیر را در نظر بگیرید:

- دانشجویان باید در گروههای ۲ نفره پژوهش را انجام دهند. تمام اعضای گروه در تمامی مراحل پژوهش بایستی مشارکت داشته باشند.

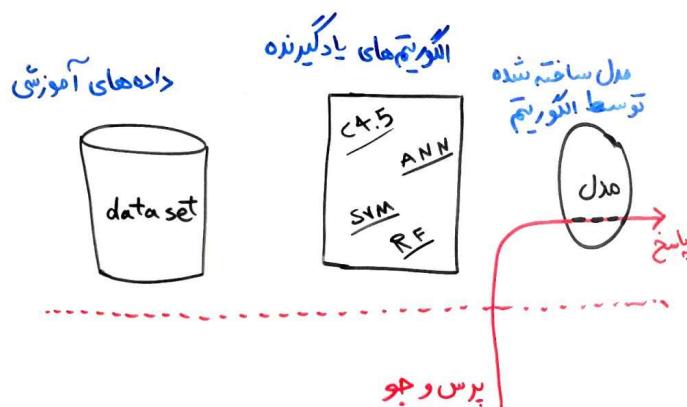
- ۲- پژوهه بایستی کامل توسط دانشجو پیاده سازی شده باشد و در هر مرحله در صورتی که مشاهده گردد که قسمتی از پژوهه کپی شده است نمره صفر برای دانشجو در نظر گرفته خواهد شد.
- ۳- دانشجو می‌تواند برای محاسبه معیارهای ارزیابی از کتابخانه‌های آماده استفاده نماید. همچنین برای الگوریتم SVM نیز می‌تواند از پیاده‌سازی‌های استاندارد استفاده نماید. بقیه موارد باید توسط دانشجو پیاده‌سازی شده باشد.

توضیحات مختصری در مورد مجموعه داده‌ها در ادامه آورده شده است.

PIE dataset. The CMU PIE (PIE) dataset contains 41 368 face images where these images are taken from 68 individuals under 13 different poses, 43 different illumination conditions, and with 4 different expressions .Moreover, each image has a resolution of 32×32 pixels. PIE dataset is divided into five subsets such that each subset corresponds to a distinct pose. These subsets are PIE1 (left pose), PIE2 (upward pose), PIE3 (downward pose), PIE4 (front pose), and PIE 5 (right pose) .(MSRC dataset includes 4323 images with 18 semantic class labels and VOC 2007 dataset contains 5011 images with 20 semantic classes.

تعاریف اولیه

یادگیری ماشین زیرمجموعه‌ای از هوش مصنوعی است. با استفاده از تکنیک‌های یادگیری ماشین، کامپیوتر، الگوهای موجود در داده‌ها (اطلاعات پردازش شده) را یادگرفته و می‌تواند از آن استفاده کند. فرآیند کلی یادگیری ماشین را می‌توان بصورت زیر مدل کرد. داده‌های آموزشی به الگوریتم‌های یادگیری ماشین تزریق می‌شوند. این الگوریتم‌ها وظیفه یادگیری و واکنشی الگوهای (Pattern) مختلف را دارند. بعد از بدست آوردن الگوها توسط الگوریتم‌ها، معمولاً یکی از الگوریتم‌ها مورد استفاده قرار می‌گیرد، و یک مدل (Model) ساخته می‌شود. این مدل می‌تواند در حافظه ذخیره شود. بعد از ذخیره مدل سیستم توانایی پیش‌بینی رفتار را دارد.



شرح مختصر الگوریتم‌ها

الگوریتم Logistic Regression

رگرسیون لجستیک یک مدل آماری رگرسیون برای متغیرهای وابسته دودویی مانند ابتلا به یک بیماری خاص یا عدم ابتلا به آن بیماری است. این مدل را می‌توان به عنوان مدل خطی تعمیم‌یافته‌ای که ازتابع لوچیت به عنوان تابع پیوند استفاده می‌کند و خطایش از توزیع چندجمله‌ای پیروی می‌کند، به حساب آورد. منظور از دودویی بودن، رخ داد یک واقعه تصادفی در دو موقعیت ممکن است. به عنوان مثال خرید یا عدم خرید، ثبت نام یا عدم ثبت نام، ورشکسته شدن یا ورشکسته نشدن و ... متغیرهایی هستند که فقط دارای دو موقعیت هستند و مجموع احتمال هر یک آن‌ها در نهایت یک خواهد شد. کاربرد این روش عمده‌ای در ابتدای ظهور در مورد کاربردهای پزشکی برای احتمال وقوع یک بیماری مورد استفاده قرار می‌گرفت. لیکن امروزه در تمام زمینه‌های علمی کاربرد وسیعی یافته‌است. به عنوان مثال مدیر سازمانی می‌خواهد بداند در مشارکت یا عدم مشارکت کارمندان کدام متغیرها نقش پیش‌بینی دارند؟ مدیر تبلیغاتی می‌خواهد بداند در خرید یا عدم خرید یک محصول یا برنز چه متغیرهایی مهم هستند؟ یک مرکز تحقیقات پزشکی می‌خواهد بداند در مبتلا شدن به بیماری عروق کرنری قلب چه متغیرهایی نقش پیش‌بینی‌کننده دارند؟ تا با اطلاع‌رسانی از احتمال وقوع کاسته شود.

رگرسیون لجستیک می‌تواند یک مورد خاص از مدل خطی عمومی و رگرسیون خطی دیده شود. مدل رگرسیون لجستیک، بر اساس فرض‌های کاملاً متفاوتی (درباره رابطه متغیرهای وابسته و مستقل) از رگرسیون خطی است. تفاوت مهم این دو مدل در دو ویژگی رگرسیون لجستیک می‌تواند دیده شود. اول توزیع شرطی $x|y$ یک توزیع برنولی به جای یک توزیع گوسی است چون‌که متغیر وابسته دودویی است. دوم مقادیر پیش‌بینی احتمالاتی است و محدود بین بازه صفر و یک و به کمک تابع توزیع لجستیک بدست می‌آید رگرسیون لجستیک احتمال خروجی را پیش‌بینی می‌کند.

الگوریتم Locally Weighted Regression (LWR)

نوعی از الگوریتم رگرسیون است که به هر نمونه آموزشی در تابع هزینه آن بوسیله $(x^{(i)})^w$ وزن اختصاص می‌دهد. که پارامتر $R \in \tau$ بصورت زیر تعریف شده است:

$$w^{(i)}(x) = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

نکته: در فرمول فوق برای کاهش حجم محاسبات مقدار پهنای باند را برابر $\frac{1}{\sqrt{2}}$ قرار داده ایم لذا مخرج کسر برابر با یک شده است.

الگوریتم رویکرد Generative با/بدون فرض

یکی از تقسیم بندی‌های متداول در یادگیری ماشین، تقسیم بندی براساس نوع داده‌ها است.

یادگیری نظارت شده (Supervised) و یادگیری بدون نظارت (Unsupervised). در یادگیری نظارت شده که یک روش عمومی در یادگیری ماشین است و در آن به یک سیستم، مجموعه‌ای از جفت‌های ورودی - خروجی ارائه شده و سیستم تلاش می‌کند تا تابعی از ورودی به خروجی را فرا بگیرد. یادگیری Supervised نیازمند تعدادی داده ورودی به منظور آموزش سیستم است. یادگیری تحت نظارت خود به دو دسته تقسیم می‌شود: رگرسیون و کلاس‌بندی. خروجی رگرسیون یک عدد پیوسته است مثل پیش‌بینی قیمت خانه بر اساس اطلاعاتی مانند مساحت، تعداد اتاق و غیره. در مسائل از نوع کلاس‌بندی، خروجی یک عضو از یک کلاس یا یک مجموعه است مثل تشخیص اینکه یک ایمیل هرزنامه است یا خیر، یا کلاس‌بندی بوهایی که توسط سنسورهای الکترونیکی جمع‌آوری می‌شوند.

طبقه بندی دیگری در الگوریتم‌های یادگیری ماشین وجود دارد: یادگیری با نظارت آماری که در این روش احتمال خروجی بر اساس ورودی محاسبه می‌شود. اگر ورودی x باشد و خروجی y ، $p(y|x)$ از داده‌ها یادگرفته می‌شود. عبارت دیگر یادگیری در واقع پیدا کردن تابع p است. دو روش برای پیدا کردن تابع p وجود دارد: روش Discriminative و روش Generative. در روش Discriminative تابع $p(y|x)$ بطور مستقیم یادگرفته می‌شود ولی در روش Generative ابتدا $p(y)$ و $p(x|y)$ از داده‌ها برآورد می‌شوند و بعد با استفاده از قانون بیز $p(y|x)$ محاسبه می‌شود.

در صورت فرض Naïve Bayes برای سادگی محاسبات متغیرها از یکدیگر مستقل در نظر گرفته می‌شوند هر چند که در دنیای واقعی چنین چیزی کم اتفاق می‌افتد و اغلب در مسائل ویژگی‌های آن مساله به یکدیگر وابستگی دارند. ولی با این وجود پس از مدل کردن این گونه مسائل با فرض استقلال متغیرها مشاهده شد که پاسخ بسیار خوبی هم بعنوان خروجی این نوع الگوریتم‌ها بدست آمد.

الگوریتم Generative بدون فرض Naïve Bayes فرض استقلال ویژگی‌ها از یکدیگر را ندارد و محاسبات با تعداد بسیار زیاد ویژگی‌ها جزو مسائل NP-Hard محسوب می‌شود.

الگوریتم Bayesian Logistic Regression

ایده اصلی روش بهینه بیزی، استفاده از یک توزیع پیشین برای پارامترهای مجھول مدل بجای مقادیر اولیه برای پارامترها و یا بکار بردن یک فاصله برای هر یک از پارامترها است، لذا روش بیزی کلی تراز روش‌های دیگر رگرسیون لجستیک است.

همانطور که در بخش قبلی هم گفته شد، رگرسیون لجستیک یک مدل آماری برای متغیرهای با پاسخ دودویی است. این مدل که اولین بار در حدود دهه ۵۰ میلادی در مورد داروهای مصرفی مورد استفاده قرار گرفت، در خانواده مدل‌های خطی تعمیم یافته قرار می‌گیرد که از توابع لوجیت بعنوانتابع پیوند استفاده می‌کند و با توجه به دو مدل برنولی و دو جمله‌ای ساخته می‌شود. اگر تابع مطلوبیت به صورت زیر باشد:

$$u_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_{p-1} x_{(p-1)i} + \varepsilon_i, \quad i=1,2,\dots,n$$

و متغیر تصادفی y_i دارای توزیع برنولی $B(1, \pi_i)$ باشد، دراینصورت $\pi_i = E(y_i|x_i) = \frac{1}{1 + \exp(-\mathbf{f}^T(x_i)\boldsymbol{\beta})}$ نشان‌دهنده‌ی میانگین متغیر پاسخ، یعنی احتمال پیروزی در آامین نقطه توزیع برنولی و x_i بردار متغیرهای توصیفی در آامین نقطه می‌باشد، در این حالت:

$$\pi_i = \frac{\exp(\mathbf{f}^T(x_i)\boldsymbol{\beta})}{1 + \exp(\mathbf{f}^T(x_i)\boldsymbol{\beta})} = \frac{1}{1 + \exp(-\mathbf{f}^T(x_i)\boldsymbol{\beta})}$$

و تابع پیوند لوجیت به صورت

$$g(\pi_i) = \ln \frac{\pi_i}{1 - \pi_i} = \mathbf{f}^T(x_i)\boldsymbol{\beta}$$

می‌باشد.

برآورد پارامترها با استفاده از روش درستنمایی ماکزیمم
اگر نمونه‌ای از n مشاهده مستقل

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

در دسترس باشد که x_i مقدار آامین متغیر توصیفی و y_i متغیر دوحالتی می‌باشد، یعنی:

$$y_i \sim Ber(1, \pi_i) \quad , \quad i = 1, 2, \dots, n$$

تابع درستنمایی بصورت زیر تعریف می‌شود:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^n \pi_i^{y_i} (1 - \pi_i)^{1-y_i} = \prod_{i=1}^n \left(\frac{\pi_i}{1 - \pi_i} \right)^{y_i} (1 - \pi_i)$$

بنابراین لگاریتم تابع درستنمایی بصورت زیر محاسبه می‌شود:

$$l(\boldsymbol{\beta}) = \ln L(\boldsymbol{\beta}) = \sum_{i=1}^n y_i \ln\left(\frac{\pi_i}{1-\pi_i}\right) + \ln(1-\pi_i)$$

با توجه به اینکه

$$1 - \pi_i = \frac{1}{1 + \exp(\mathbf{f}^T(x_i)\boldsymbol{\beta})}$$

۹

$$\ln\left(\frac{\pi_i}{1-\pi_i}\right) = \mathbf{f}^T(x_i)\boldsymbol{\beta}$$

مشتق تابع لگاریتم درستنامی نسبت به $\boldsymbol{\beta}$ بصورت زیر خواهد بود:

$$\begin{aligned} \frac{\partial l(\boldsymbol{\beta})}{\partial \beta_j} &= \sum_{i=1}^n y_i f_j(x_i) - \sum_{i=1}^n \frac{\exp(\mathbf{f}^T(x_i)\boldsymbol{\beta})}{1 + \exp(\mathbf{f}^T(x_i)\boldsymbol{\beta})} f_j(x_i) \\ &= \sum_{i=1}^n (y_i - \mu_i) f_j(x_i) \end{aligned}$$

حال با فرض اینکه

$$\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_n)^T \text{ و } \mathbf{F} = (\mathbf{f}(x_1), \mathbf{f}(x_2), \dots, \mathbf{f}(x_n))^T \text{ و } \mathbf{y} = (y_1, y_2, \dots, y_n)^T$$

که y و $\boldsymbol{\mu}$ بردارهای n بعدی و \mathbf{F} یک ماتریس $n \times p$ است که سطر i ام آن را بردار رگرسور $f^T(x_i)$ تشکیل می‌دهد، مشتق تابع لگاریتم درستنامی را به شکل ماتریس زیر می‌توان نوشت:

$$\frac{\partial l(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \mathbf{F}^T(\mathbf{y} - \boldsymbol{\mu})$$

حال برای بدست آوردن برآورد ماکزیمم درستنامی باید معادله زیر را حل کرد:

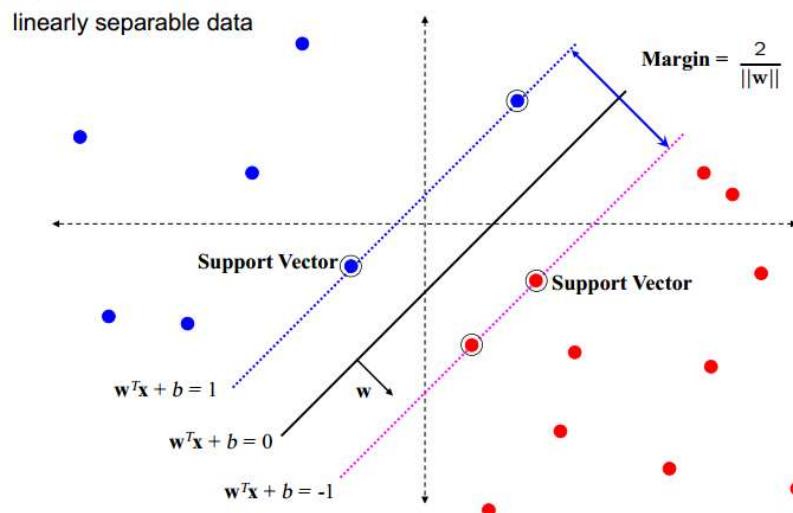
$$\mathbf{F}^T(\mathbf{y} - \boldsymbol{\mu}) = \mathbf{0}$$

ولی با توجه به اینکه معادله فوق از طریق μ به β بستگی دارد و عناصر بردار μ یعنی $\frac{1}{1+\exp(-f^T(x_i)\beta)}$ توابعی غیر خطی از β هستند. بنابراین از روش‌های عددی برای بدست آوردن جواب معادله استفاده می‌شود.

الگوریتم ماشین بردارهای پشتیبان (SVM)

یکی از الگوریتم‌ها و روش‌های بسیار رایج در حوزه کلاس‌بندی داده‌ها، الگوریتم SVM یا ماشین بردار پشتیبان است. بردارهای پشتیبان به زبان ساده، مجموعه‌ای از نقاط در فضای n بعدی داده‌ها هستند که مرز کلاس‌ها را مشخص می‌کنند. و مرزبندی داده‌ها بر اساس آنها انجام می‌شود و با جابجایی یکی از آنها، خروجی کلاس‌بندی ممکن است تغییر کند. هدف ماشین‌های بردار پشتیبان پیدا کردن خطی است که حداقل فاصله تا خط را بیشینه کند.

Support Vector Machine



ماشینهای بردار پشتیبان، الگوریتم‌های بسیار قدرتمندی در دسته بندی و تفکیک داده‌ها هستند بخصوص زمانی که با سایر روش‌های یادگیری ماشین مانند روش جنگل تصادفی تلفیق شوند. این روش برای جاهایی که با دقت بسیار بالا نیاز به ماشینی داده‌ها داریم، به شرط اینکه توابع نگاشت را به درستی انتخاب کنیم، بسیار خوب عمل می‌کند.

معیارهای ارزیابی

صحت (Precision)

زمانی که خروجی پیش‌بینی یک مدل مثبت (Positive) باشد، چگونه پی ببریم که این نتیجه تا چه اندازه‌ای قابل اطمینان است؟ جاهایی که ارزش false positive بالا باشد، مثل مدل‌های تشخیص بیماری سرطان، که تعداد منفی‌های صحیح بالاست، معیار صحت (Precision) معیار مناسبی برای سنجش صحت مدل خواهد بود.

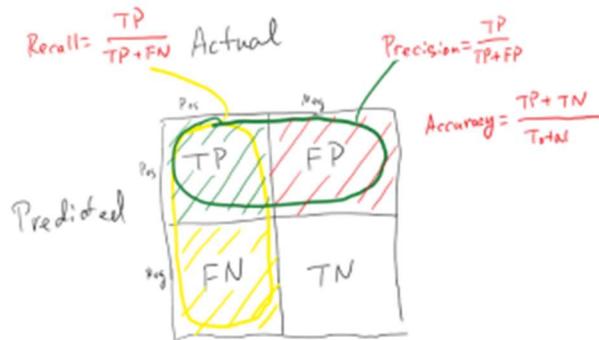
معیار صحت از رابطه زیر محاسبه می‌شود:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

حساسیت (Recall)

بعضی جاها ممکن است دقت تشخیص کلاس منفی حائز اهمیت باشد، مثل مدل مورد استفاده در تشخیص بیماری کرونا، اگر این مدل Recall پایینی داشته باشد، چه اتفاقی خواهد افتاد؟ این مدل افراد زیادی را که آلوده به این ویروس خطرناک هستند، سالم تشخیص خواهد داد و این فاجعه است. این معیار ارزیابی از طریق رابطه زیر محاسبه می‌شود:

$$\text{Recall} = \text{Sensitivity} = (\text{TPR}) = \frac{\text{TP}}{\text{TP} + \text{FN}}$$



F-Measure

معیار F1 معیار مناسبی برای ارزیابی دقت یک مدل است. این مدل یک Trade off بین معیارهای Precision و Recall برقرار می‌کند و هر دو پارامتر را بصورت همزمان استفاده می‌کند. معیار F1 در بهترین حالت ۱ و در بدترین حالت ۰ است و از رابطه زیر محاسبه می‌شود:

$$\text{F-measure} = 2 \times (\text{Recall} \times \text{Precision}) \div (\text{Recall} + \text{Precision})$$

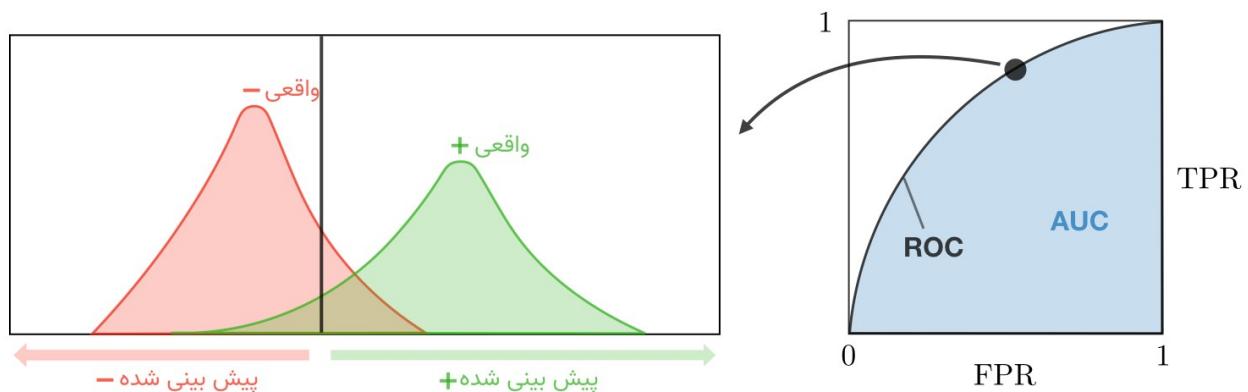
ROC معیار

منحنی عملیاتی گیرنده که تحت عنوان ROC نیز شناخته می‌شود، تصویر TPR به ازای FPR و با تغییر آستانه است. این معیارها بصورت خلاصه در جدول زیر آورده شده است:

معادل	فرمول	معیار
Recall	$TP \div (TP + FN)$	True Positive Rate (TPR)
F1 measure	$FP \div (TN + FP)$	False Positive Rate (FPR)

AUC معیار

ناحیه‌ی زیر منحنی عملیاتی گیرنده، که با AUROC یا AUC نیز شناخته می‌شود، مساحت زیر منحنی ROC که در شکل زیر نشان داده شده است:



فصل دوم - ابزارهای استفاده شده برای پیاده سازی

زبان برنامه نویسی: Python 3.7

محیط توسعه: Visual Studio 2019 و Notepad++

کتابخانه‌های استفاده شده:

- numpy
- pandas
- logging
- matplotlib.pyplot
- builtins -> range, input
- seaborn
- scipy -> interp
- scipy.io
- scipy.stats -> multivariate_normal
- itertools -> cycle
- sklearn.metrics -> roc_auc_score, auc, roc_curve
- sklearn.model_selection -> StratifiedKFold
- sklearn.preprocessing -> label_binarize
- sklearn.metrics -> precision_recall_fscore_support
- sklearn.preprocessing -> label_binarize
- sklearn.metrics -> classification_report
- sklearn.preprocessing -> LabelEncoder, OrdinalEncoder
- sklearn.metrics -> classification_report
- sklearn.linear_model.logistic -> _logistic_loss_and_grad, _logistic_loss, _logistic_grad_hess
- sklearn.linear_model.base -> LinearClassifierMixin, BaseEstimator

فصل سوم - نحوه اجرای برنامه

هر الگوریتم در یک پوشه با همان نام پیاده سازی شده است. الگوریتم‌ها بصورت کلاس پیاده سازی شده و اعمال هر الگوریتم روی هر کدام از مجموعه داده‌ها بصورت جداگانه پیاده سازی شده است. بطور مثلا برای اجرای الگوریتم لجستیک ساده برای مجموعه داده VOC باید فایلی بنام logistic_regression_VOC.py که در پوشه LR قرار دارد اجرا شود. در ادامه هر کدام از فایل‌ها توضیح داده شده است.

فایل‌های داخل پوشه LR مربوط به پیاده سازی لجستیک ساده و وزن‌دار هستند:

کلاس مربوط به پیاده سازی الگوریتم Logistic Regression ساده و وزن‌دار: logistic_regression.py

لجستیک رگرسیون ساده برای مجموعه داده MSRC: logistic_regression_MSRC.py

لجستیک رگرسیون ساده برای مجموعه داده PIE: logistic_regression_PIE.py

لجستیک رگرسیون ساده برای مجموعه داده VOC: logistic_regression_VOC.py

لجستیک رگرسیون وزن‌دار برای مجموعه داده MSRC: weighted_logistic_regression_MSRC.py

لجستیک رگرسیون وزن‌دار برای مجموعه داده PIE: weighted_logistic_regression_PIE.py

لجستیک رگرسیون وزن‌دار برای مجموعه داده VOC: weighted_logistic_regression_VOC.py

فایل‌های داخل پوشه BayesLog مربوط به پیاده سازی بیزین لجستیک رگرسیون است:

پیاده سازی الگوریتم فوق روی مجموعه داده MSRC: b_logistic_regression_MSRC.py

پیاده سازی الگوریتم فوق روی مجموعه داده VOC: b_logistic_regression_VOC.py

پیاده سازی الگوریتم فوق روی مجموعه داده PIE: b_logistic_regression_PIE.py

فایل‌های داخل پوشه Generative: مربوط به پیاده سازی رویکرد Generative با/بدون فرض نایو بیز:

مربوط به پیاده سازی الگوریتم‌های Generative با فرض نایو بیز و بدون فرض آن: generative.py

پیاده سازی الگوریتم فوق بدون فرض نایو بیز روی مجموعه داده MSRC: gen_MSRC.py

پیاده سازی الگوریتم فوق بدون فرض نایو بیز روی مجموعه داده VOC: gen_VOC.py

PIE: پیاده سازی الگوریتم فوق بدون فرض نایو بیز روی مجموعه داده gen_PIE.py

MSRC: پیاده سازی الگوریتم فوق با فرض نایو بیز روی مجموعه داده NB_gen_MSRC.py

VOC: پیاده سازی الگوریتم فوق با فرض نایو بیز روی مجموعه داده NB_gen_VOC.py

PIE: پیاده سازی الگوریتم فوق با فرض نایو بیز روی مجموعه داده NB_gen_PIE.py

فایل‌های داخل پوشه SVM: مربوط به پیاده سازی الگوریتم ماشین‌های بردار ماشین:

svm.py: پیاده سازی الگوریتم SVM البته همانطور که در شرح پژوهه گفته شده است می‌توان از کتابخانه‌های آماده برای پیاده سازی این بخش استفاده کرد.

MSRC: پیاده سازی الگوریتم فوق روی مجموعه داده svm_MSRC.py

VOC: پیاده سازی الگوریتم فوق روی مجموعه داده svm_VOC.py

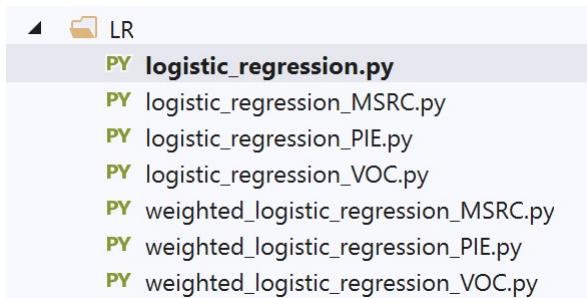
PIE: پیاده سازی الگوریتم فوق روی مجموعه داده svm_PIE.py

فصل چهارم - توضیح کد

کدها بصورت Object-Oriented پیاده سازی شده است. که در ادامه کلاس های تعریف شده توضیح داده می شود.

لجهستیک رگرسیون ساده و وزنی

یک سوپر کلاس بنام `LogisticRegression` در فایل `logistic_regression.py` و در پوشش LR تعریف شده است که توابع مشترک رگرسیون ساده و وزن دار در این کلاس تعریف شده است. دو زیر کلاس بنام های `WeightedLogisticRegression` و `SimpleLogisticRegression` توابعی که در دو زیر کلاس مورد اشاره با یکدیگر تفاوت دارند بصورت جداگانه در این دو زیر کلاس پیاده سازی شده اند.



برای هر کدام از مجموعه داده های MSRC, PIE و VOC نیز یک فایل py تعریف شده است که عملیات خواندن مجموعه داده و فراخوانی توابع پیاده سازی شده در فایل `logistic_regression.py` در آنها انجام می شود. که در ادامه هر کدام از مجموعه داده ها جداگانه توضیح داده شده اند.

ساختار توضیح به این شکل است که ابتدا کلاس های پیاده سازی شده مربوط به هر الگوریتم و توابع داخلی آن کلاس ها توضیح داده می شود سپس کد پایتون پیاده سازی شده مربوط به هر مجموعه داده تشریح می شود.

کلاس های اشاره شده، شامل چندین تابع مختلف برای انجام مراحل مختلف الگوریتم می باشد هر کدام به صورت مختصر توضیح داده می شود و همچنین در کد هم توضیحات به صورت کامنت وجود دارد.

تابع پیاده سازی شده در سوپر کلاس LogisticRegression

تابع $\text{sigmoid}(x)$: این تابع یک پارامتر دارد و فرمول تابع زیگموید را به پارامتر اعمال می‌کند و نتیجه را بر می‌گرداند.

```
class LogisticRegression(LinearClassifierMixin, BaseEstimator):
    """
    Superclass for two implementations of Logistic Regression: sigmoid
    """

    def __init__(self, iterations = 1500, learning_rate = 200):
        self.iterations = iterations
        self.learning_rate = learning_rate

    def sigmoid(self, x):
        """
        sigmoid function
        """
        return 1 / (1 + np.exp(-x))
```

تابع predict(X_eval, params) : برای پیش‌بینی برچسب نمونه‌های ارزیابی با استفاده از مقدار بهینه بردار params می‌باشد.

```
#@jit(target ="cuda")
def predict(self, X_eval, params):
    """
    return sigmoid function
    """
    return self.sigmoid(X_eval @ params)
```

تابع evaluation(Y_eval, result) : مقادیر برچسب‌های پیش‌بینی شده برای نمونه‌تست و برچسب‌های واقعی را می‌گیرد و بر اساس آن f1-score, recall, precision را محاسبه می‌کند.

```
#@jit(nopython=True, target='cuda', boundscheck=False)
def evaluation(self, Y_eval, result):
    """
    evaluation function
    """
    result = np.array(result)
    result = result.T
    result = result[0]

    label = np.argmax(result, 1)+1
    print(classification_report(Y_eval, label))
    return result
```

تابع eval_roc_auc(y_test, y_score) : برچسب‌ها واقعی و برچسب‌های پیش‌بینی شده را دریافت می‌کند و سپس AUC-ROC را محاسبه می‌کند.

```

@@jit(nopython=True, target='cuda', boundscheck=False)
def eval_roc_auc(self, y_test, y_score):
    """
    eval ROC_AUC
    """
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    # n_classes = y_bin.shape[1]
    n_classes = y_test.shape[1]
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    plt.figure()
    lw = 2
    plt.plot(fpr[2], tpr[2], color='darkorange',
              lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[2])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')

```

زیرکلاس SimpleLogisticRegression

توابع زیر در دو روش رگرسیون لجستیک ساده و وزن دار متفاوت از یکدیگر هستند به همین دلیل در سوپر کلاس پیاده سازی نشده و در هر زیر کلاس جداگانه پیاده سازی شده اند.

تابع gradient_descent(X, y, params) : این تابع هم برای اعمال روش گرادیان کاهشی پیاده سازی شده است و پارامترهای ورودی آن عبارتند از نمونه ها و برچسب ها و بردار پارامتر ها (مجھول) که در ابتداء با صفر مقداردهی شده است و نرخ یادگیری و تعداد تکرارها دیگر پارامتر های تابع هستند در این تابع با استفاده از یک حلقه تکرار به تعداد iterations هر بار بردار params را تا رسیدن به یک مقدار بهینه به روز سانی می کند و در نهایت بردار params را بر میگرداند.

```

class SimpleLogisticRegression(LogisticRegression):
    """Linear Logistic Regression"""
    def __init__(self, iterations = 1500, learning_rate = 200):...
        ...
    #@jit(nopython=True, target='cuda', boundscheck=False)
    def gradient_descent(self, X, y, params):
        m = len(y)

        for i in range(self.iterations):
            params = params - (self.learning_rate/m) * (X.T @ (self.sigmoid(X @ params) - y))
        return params

```

زیرکلاس WeightedLogisticRegression

تابع w gradient_descent(X, y, params , w) این تابع دقیقاً مثلاً تابع پیاده سازی شده در کلاس SimpleLogisticRegression است با این تفاوت که یک پارامتر اضافه تری بنام w که همان وزن است به تابع پاس می‌شود.

```

class WeightedLogisticRegression(LogisticRegression):
    """
    Weighted Logistic Regression
    """
    def __init__(self, iterations = 1500, learning_rate = 200):...
        ...

    def gradient_descent(self, X, y, params, w):
        m = len(y)

        vec = []
        '''for j in range(len(w)):
            vec.append(w[j][j])
        vec = np.array(vec)'''

        for i in range(self.iterations):
            params = params - (self.learning_rate/m) * (w * (X.T @ (self.sigmoid(X @ params) - y)))
        return params

```

مجموعه داده MSRC

اعمال الگوریتم لجستیک رگرسیون ساده بر روی مجموعه داده MSRC در فایل logistic_regression_MSRC.py پیاده سازی شده است. و همینطور اعمال الگوریتم لجستیک رگرسیون وزن دار بر روی مجموعه داده MSRC در فایل weighted_logistic_regression_MSRC.py پیاده سازی شده است. که در ادامه بطور خلاصه توضیح داده شده است.

طبق خواسته پژوهه نمونه‌ها و برچسب‌ها با استفاده از روش StratifiedKFold به پنج قسمت تقسیم می‌شود و در مرحله بعدی به کمک یک حلقه تکرار هر بار یک قسمت برای ارزیابی و چهار قسمت دیگر برای آموزش استفاده می‌شوند.

چون در اینجا بیش دو کلاس داریم مشابه روش one vs all هر بار برچسب یک کلاس را یک گرفته ایم و بقیه کلاس‌ها دارای برچسب صفر هستند و این روند برای همه کلاس‌ها تکرار شده است.

مجموعه داده VOC

اعمال الگوریتم لجستیک رگرسیون ساده بر روی مجموعه داده VOC در فایل logistic_regression_VOC.py پیاده سازی شده است. و همینطور اعمال الگوریتم لجستیک رگرسیون وزن دار بر روی مجموعه داده VOC در فایل weighted_logistic_regression_VOC.py پیاده سازی شده است. که در ادامه بطور خلاصه توضیح داده شده است.

طبق خواسته پژوهه نمونه ها و برچسب ها با استفاده از روش StratifiedKFold به پنج قسمت تقسیم می شود و در مرحله بعدی به کمک یک حلقه تکرار هر بار یک قسمت برای ارزیابی و چهار قسمت دیگر برای آموزش استفاده می شوند.

چون در اینجا بیش دو کلاس داریم مشابه روش one vs all هر بار برچسب یک کلاس را یک گرفته ایم و بقیه کلاس ها دارای برچسب صفر هستند و این روند برای همه کلاس ها تکرار شده است.

مجموعه داده PIE

اعمال الگوریتم لجستیک رگرسیون ساده بر روی مجموعه داده PIE در فایل logistic_regression_PIE.py پیاده سازی شده است. و همینطور اعمال الگوریتم لجستیک رگرسیون وزن دار بر روی مجموعه داده PIE در فایل weighted_logistic_regression_PIE.py پیاده سازی شده است. که در ادامه بطور خلاصه توضیح داده شده است.

این کد شامل چندینتابع مختلف برای انجام مراحل مختلف الگوریتم می باشد هر کدام به صورت مختصر توضیح داده می شود و هم چنین در کد هم توضیحات به صورت کامنت وجود دارد.

حال در این مرحله چون این دیتا است به صورت استاندارد به ۵ قسمت تقسیم شده است به کمک یک حلقه هر بار به کمک چهار قسمت برای آموزش و یک قسمت برای ارزیابی مدل استفاده می شود.

چون در اینجا بیش دو کلاس داریم مشابه روش one vs all هر بار برچسب یک کلاس را یک گرفته ایم و بقیه کلاس ها دارای برچسب صفر هستند و این روند برای همه کلاس ها تکرار شده است.

متاسفانه با توجه به تعداد زیاد نمونه ها و کلاس ها دیتا و ابعاد بالای ویژگی ها هنگام اجرای کد لاجستیک رگرسیون وزن دار رو دیتا PIE به خطر کمبود حافظه ای اصلی (رم سیستم) امکان مشاهده نتیجه کد را نداشتیم.

```
MemoryError: Unable to allocate array with shape (8222, 3332) and data type float64
>>>
```

بیزین لجستیک رگرسیون

کلاس و کدهای مربوط به این الگوریتم در پوشه BayesLog قرار دارد، در ابتدا کلاس Bayesian_logistic_regression.py را شرح می‌دهیم که مربوط به پیاده سازی الگوریتم Logistic Regression می‌باشد. و در ادامه اعمال الگوریتم مورد نظر روی هر کدام از مجموعه داده‌ها بصورت جداگانه توضیح داده خواهد شد.



BayesianLogisticRegression

کلاس مورد نظر در فایل Bayesian_logistic_regression.py قرار دارد. این الگوریتم با روش درست‌نمایی ماکزیمم پیاده سازی شده است که روش کار آن در فصل اول بخش الگوریتم "بیزین لجستیک رگرسیون" و زیر بخش "برآورد پارامترها با استفاده از درست‌نمایی ماکزیمم"، توضیح داده شده است.

این کلاس از دو کلاس BaseEstimator و LinearClassifierMixin از کتابخانه آماده sklearn.linear_model.base جهت ارزیابی مدل و پیش‌بینی، ارت بری کرده است

کلاس مورد نظر شامل پیاده‌سازی توابع زیر است:

تابع `fit(self, X, y)`: که مجموعه داده آموزشی و برچسب‌ها را بعنوان ورودی تابع دریافت می‌کند. قبل از انجام هر کاری با استفاده از تابع `check_X_y` کتابخانه آماده `sklearn.utils` کار پیش پردازش که شامل بررسی هم اندازه بودن داده‌های آموزشی و برچسب‌ها و همینطور `null` نبودن لیست برچسب‌ها و غیره است انجام می‌شود. سپس از طریق یک حلقه کلاس‌بندی داده‌های آموزشی با روش `one-vs-all` را انجام می‌دهد.

```

def fit(self,X,y):
    ...
    Fits Bayesian Logistic Regression
    ...

    # preprocess data
    X,y = check_X_y( X, y , dtype = np.float64)

    self.classes_ = np.unique(y)
    n_classes = len(self.classes_)

    n_samples, n_features = X.shape
    self.coef_, self.sigma_ = [0]*n_classes,[0]*n_classes
    self.intercept_ = [0]*n_classes
    # make classifier for each class (one-vs-all) []
    for i in range(len(self.coef_)):
        pos_class = self.classes_[i]
        mask = (y == pos_class)
        y_bin = np.ones(y.shape, dtype=np.float64)
        y_bin[~mask] = self._mask_val
        coef_, sigma_ = self._fit(X,y_bin)
        self.coef_[i] = coef_
        self.sigma_[i] = sigma_

```

تابع `_posterior(self, X, Y, alpha0, w0)` در اینجا پارامترهای طبقه‌بند با استفاده از بیشینه درست‌نمایی و بیشینه احتمال پسین بدست آورده می‌شوند. که از تابع `_logistic_loss_and_grad` کتابخانه آماده برای اینکار استفاده شده است. ماتریس هسیان همانطور که در کد مشخص است با استفاده از پارامتر `W` بدست آمده از تابع بالا محاسبه می‌شود.

برای پیاده سازی این بخش از کد از توضیحات قسمت دوم تمرین دوم درس یادگیری ماشین (فصل زمستان ۹۸ دکتر رزاقی) استفاده شده است.

سوال دوم- در این مسئله تخمین پارامترهای طبقه‌بند Naïve Bayes را با استفاده از بیشینه درست‌نمایی و بیشینه احتمال پسین ملاحظه خواهیم کرد. ویژگی‌های ورودی با استفاده از x_j , $j=1,2,\dots,n$ نمایش داده می‌شوند که به صورت گستره $x_j \in \{0,1\}$ نمایش داده می‌شوند. هر نمونه آموزشی با استفاده از بردار $x = [x_1 \ x_2 \ \dots \ x_n]^T$ نمایش داده می‌شود و خروجی با استفاده از مقدار پابندی $y \in \{0,1\}$ نمایش داده می‌شود. مدل با استفاده از پارامترهای $\phi_{j|y=1}, \phi_{j|y=0} = p(x_j = 1 | y = 1), \phi_{j|y=0} = p(x_j = 1 | y = 0)$ و $\phi_y = p(y = 1)$ نمایش داده می‌شود.

(الف) احتمال توان درست‌نمایی را $I(\varphi) = \log \prod_{i=1}^m p(x^{(i)}, y^{(i)}; \varphi)$ براساس پارامترهای ذکر شده به دست آورید. پارامتر φ به صورت مجموعه ای از پارامترهای $\{\phi_j, \phi_{j|y=1}, \phi_{j|y=0}, j = 1, \dots, n\}$ نمایش داده می‌شود.

(ب) نشان دهد که پارامترهای مسئله با استفاده از بیشینه درست‌نمایی به صورت زیر به دست می‌آید:

$$\phi_{j|y=0} = \frac{\sum_{i=1}^m \mathbb{1}\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m \mathbb{1}\{y^{(i)} = 0\}}$$

$$\phi_{j|y=1} = \frac{\sum_{i=1}^m \mathbb{1}\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m \mathbb{1}\{y^{(i)} = 1\}}$$

$$\phi_y = \frac{\sum_{i=1}^m \mathbb{1}\{y^{(i)} = 1\}}{m}$$

```

def _posterior(self, X, Y, alpha0, w0):
    """
    posterior function
    """
    n_samples, n_features = X.shape

    f = lambda w: _logistic_loss_and_grad(w, X[:, :-1], Y, alpha0)
    w = fmin_l_bfgs_b(f, x0 = w0, pgtol = self.tol_solver,
                       maxiter = self.n_iter_solver)[0]

    # calculate negative of Hessian at w
    xw      = np.dot(X, w)
    s       = expit(xw)
    R       = s * (1 - s)
    Hess   = np.dot(X.T * R, X)
    Alpha  = np.ones(n_features) * alpha0
    np.fill_diagonal(Hess, np.diag(Hess) + Alpha)
    e      = eigvalsh(Hess)
    return w, 1./e

```

در ادامه توضیح نحوه اعمال الگوریتم فوق بر روی مجموعه داده‌ها بصورت جداگانه توضیح داده شده است:

مجموعه داده MSRC

فایل BayesLog b_logistic_regression_MSRC.py

مجموعه داده MSRC با استفاده از تابع `loadmat` کتابخانه آماده `scipy.io` خوانده می‌شود و برچسبها و مجموعه داده از هم تفکیک می‌شود سپس یک نمونه (Instance) از کلاس `BayesianLogisticRegression` که در بخش بالا توضیح داده شد ایجاد می‌شود. آنگاه تابع `fit` (در بالا توضیح داده شده است) با پارامترهای مجموعه داده آموزشی و برچسبها فراخوانی می‌شود. برای ارزیابی مدل نیز خروجی تابع آماده `predict` از کتابخانه آماده `BestEstimator` که سوپر کلاس، کلاس پیاده سازی شده است، بعنوان پارامتر ورودی تابع `classification_report` از همان کلاس، پاس شده است. و در نهایت خروجی تابع فوق که شامل معیارهای ارزیابی است چاپ می‌شود.

```

mat = sio.loadmat('MSRC/MSRC.mat')

X = mat['fts']
y = mat['labels']

blr = BayesianLogisticRegression()
blr.fit(X, y)

print("\n === MSRC: Bayesian Logistic Regression ===")
print(classification_report(y, blr.predict(X)))

```

مجموعه داده VOC

فایل BayesLog b_logistic_regression_VOC.py

دقیقاً مثل توضیحات بخش مجموعه داده MSRC است با این تفاوت که در این فایل داده‌های مربوط به VOC لود می‌شود.

```
mat = sio.loadmat('VOC/VOC.mat')

X = mat['fts']
y = mat['labels']

blr = BayesianLogisticRegression()
blr.fit(X, y)

print("\n === VOC: Bayesian Logistic Regression ===")
print(classification_report(y, blr.predict(X)))
```

مجموعه داده PIE

فایل BayesLog در پوشه b_logistic_regression_PIE.py

دقیقاً مثل توضیحات بخش مجموعه داده MSRC است با این تفاوت که در این فایل داده‌های مربوط به PIE لود می‌شود.

```
mat1 = sio.loadmat('PIE/P1.mat')
mat2 = sio.loadmat('PIE/P2.mat')
mat3 = sio.loadmat('PIE/P3.mat')
mat4 = sio.loadmat('PIE/P4.mat')
mat5 = sio.loadmat('PIE/P5.mat')
Y1 = mat1['labels']
Y2 = mat2['labels']
Y3 = mat3['labels']
Y4 = mat4['labels']
Y5 = mat5['labels']
X1 = mat1['fts']
X2 = mat2['fts']
X3 = mat3['fts']
X4 = mat4['fts']
X5 = mat5['fts']

X = np.concatenate([X1,X2,X3,X4,X5])
y = np.concatenate([Y1,Y2,Y3,Y4,Y5])

blr = BayesianLogisticRegression()
blr.fit(X, y)

print("\n === PIE: Bayesian Logistic Regression ===")
print(classification_report(y, blr.predict(X)))
```

ماشین‌های بردار پشتیبان (SVM)

کلاس و کدهای مربوط به این الگوریتم در پوشه SVM قرار دارد، در ابتدا کلاس svm.py که شامل دو کلاس PIESVM و SVM است را شرح می‌دهیم که مربوط به پیاده سازی الگوریتم SVM می‌باشد. همانطور که در شرح پژوهه اشاره شده در این قسمت مجاز به استفاده از کتابخانه‌های آماده هستیم که در این بخش از کتابخانه sklearn.svm استفاده شده است. و در ادامه اعمال الگوریتم مورد نظر روی هر کدام از مجموعه داده‌ها بصورت جداگانه توضیح داده خواهد شد.

```
◀ └── SVM
    PY svm.py
    PY svm_MSRC.py
    PY svm_PIE.py
    PY svm_VOC.py
```

کلاس SVM

این کلاس در پیاده سازی مجموعه داده‌های MSRC و VOC استفاده شده است. این کلاس خود سوپر کلاس، کلاس دیگری بنام PIESVM است که در پیاده سازی مجموعه داده PIE استفاده شده است. زیرا برخی توابع پیاده سازی شده در این دو کلاس متفاوت بودند.

```
import logging
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score, auc, roc_curve
from itertools import cycle
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.multiclass import OneVsRestClassifier
from sklearn.preprocessing import label_binarize
from scipy import interp

class SVM(object):
    ...
class PIESVM(SVM):
    ...
```

تابع **evaluation(Y_eval, result)**: مقادیر برچسب‌های پیش‌بینی شده برای نمونه تست و برچسب‌های واقعی را می‌گیرد و بر اساس آن f1-score, recall, precision را محاسبه می‌کند.

```
def evaluation(self, Y_eval, prediction):
    ...
    evaluation function
    ...
    print(classification_report(Y_eval, prediction))
```

تابع eval_roc_auc(y_test, y_score) : برچسب ها واقعی و برچسب های پیش بینی شده را دریافت می کند و سپس، AUC-ROC را محاسبه می کند.

```
def eval_roc_auc(self, y_test, y_score, y_bin):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    n_classes = y_bin.shape[1]
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    plt.figure()
    lw = 2
    plt.plot(fpr[2], tpr[2], color='darkorange',
              lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[2])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

تابع svc: در مراحل بعدی طبق خواسته پژوهه نمونه ها و برچسب ها با استفاده از روش StratifiedKFold به پنج قسمت تقسیم می شود و در مرحله بعدی به کمک یک حلقه تکرار هر بار یک قسمت برای ارزیابی و چهار قسمت دیگر برای آموزش استفاده می شوند.

چون در اینجا بیش دو کلاس داریم مشابه روش one vs all هر بار برچسب یک کلاس را یک گرفته ایم و بقیه کلاس ها دارای برچسب صفر هستند و این روند برای همه کلاس ها تکرار شده است.

```
def svc(self, X, y, y_bin):
    ...
    svc_function
    ...

    logging.basicConfig(filename='svm.log', level=logging.DEBUG, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    logger = logging.getLogger()
    logger.debug(f"svm");

    kfolds = StratifiedKFold(n_splits=5, random_state=1).split(X, y)
    for fold, (train_index, eval_index) in enumerate(kfolds):
        X_cross, X_eval = X[train_index], X[eval_index]
        Y_cross, Y_eval = y[train_index], y[eval_index]
        y_bin_cross, y_bin_eval = y_bin[train_index], y_bin[eval_index]
        clf = OneVsRestClassifier(SVC(gamma='auto', probability=True)).fit(X_cross, Y_cross)
        y_pred = np.zeros((np.size(X_eval, 0)))
        y_true = np.zeros((np.size(X_eval, 0)))
        prediction = clf.predict(X_eval)
        y_score = clf.fit(X_cross, y_bin_cross).decision_function(X_eval)
        logger.info('\n' + "-"*20)
        self.evaluation(Y_eval, prediction)
        self.eval_roc_auc(y_bin_eval, y_score, y_bin)
```

کلاس PIESVM

همانطور که در بالا هم اشاره شد از این کلاس در پیاده سازی مجموعه داده PIE استفاده شده است بدلیل اینکه پیاده سازی توابع SVC و trainEval متفاوت از دو مجموعه داده دیگر بود.

تابع **evaluation(Y_eval, result)**: مقادیر برچسب های پیش بینی شده برای نمونه تست و برچسب های واقعی را می گیرد و بر اساس آن f1-score, recall, precision را محاسبه می کند.

```
def evaluation(self, Y_eval, prediction):
    print(classification_report(Y_eval, prediction))
    print("---"*20)
```

تابع **svc**: در مراحل بعدی طبق خواسته پژوهش نمونه ها و برچسب ها با استفاده از روش StratifiedKFold به پنج قسمت تقسیم می شود و در مرحله بعدی به کمک یک حلقه تکرار هر بار یک قسمت برای ارزیابی و چهار قسمت دیگر برای آموزش استفاده می شوند.

چون در اینجا بیش دو کلاس داریم مشابه روش one vs all هر بار برچسب یک کلاس را یک گرفته ایم و بقیه کلاس ها دارای برچسب صفر هستند و این روند برای همه کلاس ها تکرار شده است.

```
def svc(self, X, y, y_bin):
    """
    svc function
    """
    logging.basicConfig(filename='svm.log', level=logging.DEBUG, format='%(asctime)s - %(name)s'
    logger = logging.getLogger()
    logger.debug(f"svm");

    kfolds = StratifiedKFold(n_splits=5, random_state=1).split(X, y)
    for fold, (train_index, eval_index) in enumerate(kfolds):
        X_cross, X_eval = X[train_index], X[eval_index]
        Y_cross, Y_eval = y[train_index], y[eval_index]
        y_bin_cross, y_bin_eval = y_bin[train_index], y_bin[eval_index]
        clf = OneVsRestClassifier(SVC(gamma='auto', probability=True)).fit(X_cross, Y_cross)
        y_pred = np.zeros((np.size(X_eval, 0)))
        y_true = np.zeros((np.size(X_eval, 0)))
        prediction = clf.predict(X_eval)
        y_score = clf.fit(X_cross, y_bin_cross).decision_function(X_eval)
        logger.info('\n' + "---"*20)
        self.evaluation(Y_eval, prediction)
        self.eval_roc_auc(y_bin_eval, y_score, y_bin)
```

مجموعه داده MSRC

فایل `svm_MSRC.py` در پوشه SVM با استفاده از تابع `loadmat` کتابخانه آماده `scipy.io` خوانده می‌شود و برچسبها و مجموعه داده از هم تفکیک می‌شود سپس یک نمونه (`Instance`) از کلاس SVM که در بخش بالا توضیح داده شد ایجاد می‌شود. آنگاه تابع `SVC` (در بالا توضیح داده شده است) با پارامترهای مجموعه داده آموزشی و برچسبها و همینطور برچسبهایی که بصورت باینری (برای استفاده در روش `one vs all`) تبدیل شده اند، فراخوانی می‌شود.

مجموعه داده VOC

فایل `svm_VOC.py` در پوشه SVM با استفاده از تابع `loadmat` کتابخانه آماده `scipy.io` خوانده می‌شود و برچسبها و مجموعه داده از هم تفکیک می‌شود سپس یک نمونه (`Instance`) از کلاس SVM که در بخش بالا توضیح داده شد ایجاد می‌شود. آنگاه تابع `SVC` (در بالا توضیح داده شده است) با پارامترهای مجموعه داده آموزشی و برچسبها و همینطور برچسبهایی که بصورت باینری (برای استفاده در روش `one vs all`) تبدیل شده اند، فراخوانی می‌شود.

مجموعه داده PIE

فایل `svm_PIE.py` در پوشه SVM با استفاده از تابع `loadmat` کتابخانه آماده `scipy.io` خوانده می‌شود. سپس یک نمونه (`Instance`) از کلاس `PIESVM` که در بخش بالا توضیح داده شد ایجاد می‌شود. آنگاه تابع `SVC` (در بالا توضیح داده شده است) با پارامترهای مجموعه داده آموزشی و برچسبها فراخوانی می‌شوند.

رویکرد Generative (با/بدون فرض Naïve Bayes)

کلاس و کدهای این الگوریتم در پوشه Generative قرار دارد. کدهای مربوط به پیاده سازی الگوریتم در فایل generative.py تحت عنوان کلاس Naïve Bayes برای بدون فرض Generative و کلاس Naïve BayesGenerative برای با فرض Naïve Bayes که خود از کلاس Naïve Bayes بدليل اشتراک کد زیاد به ارث رفته است، پیاده سازی شده است. این دو پیاده سازی در توابع زیر با یکدیگر **تفاوت** دارند:

تابع **NB_covar_cal(X)** : این تابع همه نمونه ها را به عنوان پارامتر می گیرد و برای کل محاسبات یک ماتریس کواریانس را به ما می دهد.

```
def NB_covar_cal(self, X):
    ...
    ...
    return np.cov(X, rowvar=False)
```

تابع **predict_class(X, class_prior, cov, mean, Y)** : این تابع نیز با استفاده از محاسبات انجام شده به وسیله دیگر توابع برچسب نمونه را پیش بینی می کند و پارامتر های آن عبارتند از نمونه ها، احتمال پیشین هر کلاس، کواریانس هر کلاس، میانگین هر کلاس و مقدار واقعی برچسب ها. به عبارت دیگر این تابع برای هر نمونه برای تمام کلاس ها احتمال پیشین کلاس را در تابع چگالی احتمال ضرب میکند و نتیجه را برای ارزیابی ذخیره می کند.

```
def predict_class(self, X, class_prior, cov, mean, Y):
    ...
    predict target class
    ...
    k = len(np.unique(Y))
    prediction = []
    out_come = []
    for i in range(len(X)):
        label = []
        for j in range(1, k):
            res = class_prior[j] * self.prob_dens_func(i, cov, mean[j])
            label.append(res)
        ind = np.argmax(label, 1)
        out_come.append(label)
        prediction.append(ind)
    return prediction, out_come
```

تابع فوق در بخشی که هایلایت شده است با همین تابع در الگوریتم بدون فرض Naïve Bayes متفاوت است. به همین دلیل بصورت تابعی جداگانه در کلاس NaiveBayesGenerative پیاده سازی شده است.

تابع $\text{cal_mean}(X, Y)$: این تابع تمام نمونه ها و برچسب ها را به عنوان ورودی می گیرد و میانگین هر کلاس را محاسبه می کند و میانگین های محاسبه شده را در یک ذخیره می کند و در پایان دیکشنری را بر می گرداند.

```
def cal_mean(self, X, Y):
    """
    calculate mean and covariance of any class
    """
    k = len(np.unique(Y))
    mean = dict()
    cov = dict()
    for s in range(1, k+1):
        l = []
        for i in range(len(Y)):
            if Y[i] == s:
                l.append(X[i])
        l = np.array(l)
        myangin = l.mean(axis=0)
        mean[s] = myangin
    return mean
```

توابعی که شرح آنها در ادامه آمده است در هر دو الگوریتم با/بدون فرض نایو بیز مشترک هستند.

شرح توابع داخل هر کلاس در ادامه خواهد آمد. اعمال الگوریتم روی هر مجموعه داده بطور جداگانه در فایلی با نام متناسب پیاده سازی شده است که شرح آن نیز در ادامه وجود دارد. برای پیاده سازی این روش در ابتدا کتابخانه های مورد نیاز فراوانی شده اند و سپس برای اعمال الگوریتم توابع زیر را تعریف کرده ایم:

تابع $\text{prob_dens_func}(i, std, mean)$: این تابع همان تابع چگالی احتمال می باشد و تابع سه پارامتر شامل یک نمونه و کوواریانس کلاس ها و میانگین هر کلاس و سپس چگالی به دست آمده را بر می گرداند.

```
def prob_dens_func(self, i, std, mean):
    """
    prob_dens_func function
    """
    pdf = (1/(((2*np.pi)**-(self.n/2)) * (np.linalg.det(std))**-0.5)) * np.exp((i - mean).T * np.linalg.pinv(std))
    return pdf
```

تابع $\text{cal_mean_cov}(X, Y)$: این تابع تمام نمونه ها و برچسب ها را به عنوان ورودی می گیرد و میانگین هر کلاس و کوواریانس هر کلاس را محاسبه می کند و میانگین های محاسبه شده را در یک دیکشنری و کوواریانس های به دست آمده را در یک یکشنبه دیگر ذخیره می کند و در پایان هر دو دیکشنری را بر می گرداند.

```
def cal_mean_cov(self, X, Y):
    """
    calculate mean and covariance of any class
    """
    k = len(np.unique(Y))
    mean = dict()
    cov = dict()
    for s in range(1, k+1):
        l = []
        for i in range(len(Y)):
            if Y[i] == s:
                l.append(X[i])
        l = np.array(l)
        myangin = l.mean(axis=0)
        covar = np.cov(l.T) #covariance of feature
        mean[s] = myangin
        cov[s] = covar
    return mean, cov
```

تابع **evaluation(Y_eval, result)** برای محاسبه و چاپ precision و f1-score, recall است.

```
def evaluation(self, Y_eval, result):
    """
    for calculating precision, recall and f1-score
    """
    print(classification_report(Y_eval, label))
    return result
```

تابع **eval_roc_auc(y_test, y_score)** برای محاسبه و رسم نمودار معیار ارزیابی ROC_AUC استفاده شده است.

```
def eval_roc_auc(self, y_test, y_score):
    """
    calculate ROC-AUC estimation
    """
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    n_classes = y_test.shape[1]
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    plt.figure()
    lw = 2
    plt.plot(fpr[2], tpr[2], color='darkorange',
              lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[2])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

تابع ($\text{prior_of_any_class}(Y)$) : این تابع احتمال پیشین هر کلاس را محاسبه می کند.

```
def prior_of_any_class(self, Y):
    ...
    calculate prior probability of any class
    ...
    k = len(np.unique(Y))
    class_prior = dict()
    for s in range(1, k+1):
        counter = 0
        for i in range(len(Y)):
            if Y[i] == s :
                counter += 1
        res = counter/len(Y)
        class_prior[s] = res
    return class_prior
```

تابع ($\text{predict_class}(X, \text{class_prior}, \text{cov}, \text{mean}, Y)$) : این تابع نیز با استفاده از محاسبات انجام شده به وسیله دیگر توابع برچسب نمونه را پیش بینی می کند و پارامترهای آن عبارتند از نمونه ها، احتمال پیشین هر کلاس، کواریانس هر کلاس، میانگین هر کلاس و مقدار واقعی برچسب ها. به عبارت دیگر این تابع برای هر نمونه برای تمام کلاس ها احتمال پیشین کلاس را در تابع چگالی احتمال ضرب می کند و نتیجه را برای ارزیابی ذخیره می کند.

```
def predict_class(self, X, class_prior, cov, mean, Y):
    ...
    predict target class
    ...
    k = len(np.unique(Y))
    prediction = []
    out_come = []
    for i in range(len(X)):
        label = []
        for j in range(1, k):
            res = class_prior[j] * self.prob_dens_func(i, cov[j], mean[j])
            label.append(res)
        ind = np.argmax(label, 1)
        out_come.append(label)
        prediction.append(ind)
    return prediction, out_come
```

تابع ($\text{softmax}(x)$) : این تابع نیز یک بردار را از ورودی می گیرد که این بدار شامل احتمال تعلق یک نمونه برای هر کلاس می باشد ، سپس کلاس مربوط به نمونه را بر می گرداند.

```
def softmax(self, x):
    """
    Compute softmax values for each sets of scores in x.
    """
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum(axis=0)
```

تابع **(Y pred_eval(self, X, Y))**: در پایان نیز برای اعمال الگوریتم توابع به ترتیب نیاز فراخوانی شده اند.

```
def pred_eval(self, X, Y):
    class_num = np.unique(Y)
    y_bin = label_binarize(Y, classes=class_num)

    mean, cov = self.cal_mean_cov(X, Y)
    self._pred_eval(X, Y, mean, cov)

def _pred_eval(self, X, Y, mean, cov):
    print("mean of any class :" + "\n")
    print(mean, "\n", "\n")
    print("---*20)
    print("covariance of any class :" + "\n")
    print(cov, "\n", "\n")
    print("---*20)
    class_prior = self.prior_of_any_class(Y)
    print("prior of any class :" + "\n")
    print(class_prior, "\n", "\n")
    expectancy, out_come = self.predict_class(X, class_prior, cov, mean, Y)
    #print(expectancy)
    self.evaluation(Y_eval, expectancy)
    out_come = np.array(out_come)
    out_come = out_come[0]
    self.eval roc auc(y bin, out come)
```

مجموعه داده MSRC

فایل gen_MSRC.py در پوشش MSRC Generative مجموعه داده loadmat کتابخانه آماده scipy.io خوانده می‌شود و برچسبها و مجموعه داده از هم تفکیک می‌شود سپس یک نمونه Instance() از کلاس Generative که در بخش بالا توضیح داده شد ایجاد می‌شود. آنگاه تابع pred_eval (در بالا توضیح داده شده است) با پارامترهای مجموعه داده آموزشی و برچسبها، فراخوانی می‌شود.

مجموعه داده VOC

فایل gen_VOC.py در پوشش VOC Generative مجموعه داده loadmat کتابخانه آماده scipy.io خوانده می‌شود و برچسبها و مجموعه داده از هم تفکیک می‌شود سپس یک نمونه Instance() از کلاس Generative که در بخش بالا توضیح داده شد ایجاد می‌شود. آنگاه تابع pred_eval (در بالا توضیح داده شده است) با پارامترهای مجموعه داده آموزشی و برچسبها، فراخوانی می‌شود.

مجموعه داده PIE

فایل gen_PIE.py در پوشه Generative مجموعه داده PIE با استفاده از تابع loadmat کتابخانه آماده scipy.io خوانده می‌شود و پس از تفکیک برچسبها از مجموعه داده‌های آموزشی، برچسبها بصورت یک آرایه و مجموعه داده‌های آموزشی نیز بصورت یک آرایه واحد تبدیل می‌شوند. سپس یک شی از کلاس Generative ایجاد می‌شود و تابع pred_eval همراه با پارامترهای مجموعه داده آموزشی و برچسبها فراخوانی می‌شود و نتایج در خروجی چاپ می‌شود. نکته‌ای که وجود دارد برای اعمال این الگوریتم روی مجموعه داده PIE بدلیل تعداد زیاد نمونه‌ها و همینطور ویژگی‌ها پس از گذشت ۴ ساعت از زمان اجرا با خطای کمبود حافظه مواجه شدیم و امکان تولید خروجی وجود نداشت.

فصل پنجم - نتایج ارزیابی

نتایج ارزیابی و معیارهای Precision، Recall، F1-Score، AUC-ROC و نمودارهای مربوط به آن در پوشاهای جدأگانه بنام Evaluation در کنار همین فایلی که در حال مطالعه آن هستید وجود دارد.

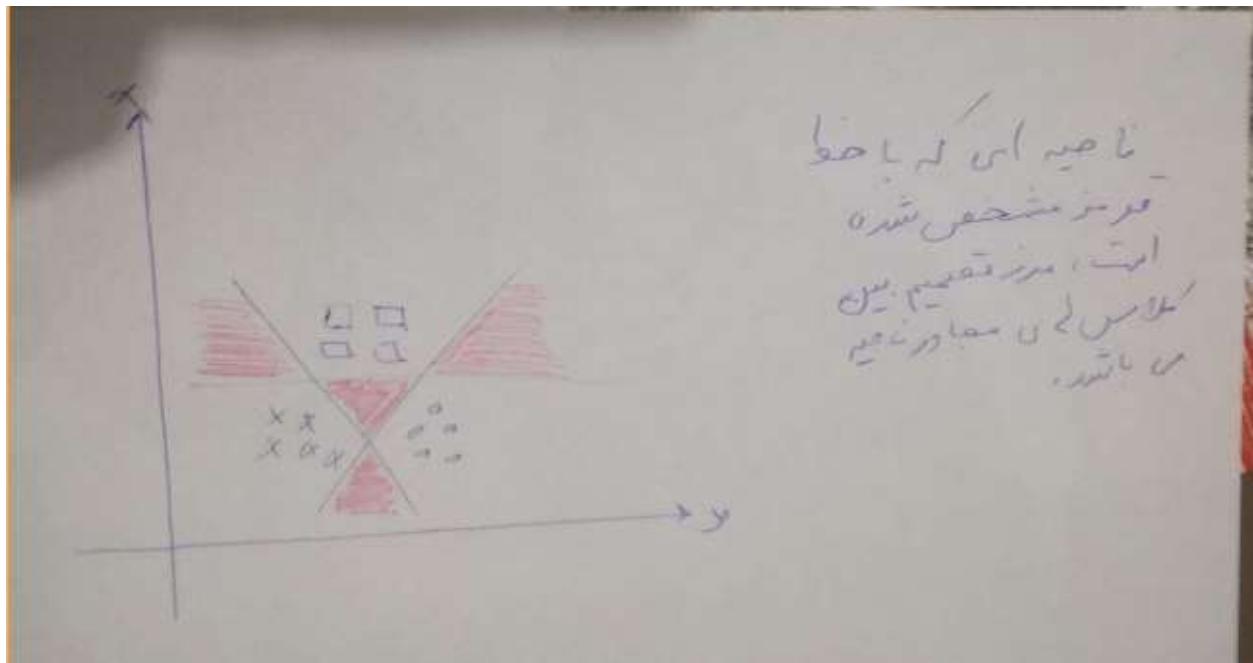
فصل ششم - گام دوم پروژه

در گام دوم بایستی راه حلی برای ترکیب الگوریتم‌های استفاده شده در مرحله قبل ارائه دهید به گونه‌ای که باعث افزایش دقت گردد. بدین منظور در حوزه یادگیری ماشین حوزه Ensemble Learning ارائه شده است که می‌توانید از این ایده‌ها نیز بهره بگیرید. برخی از لینکهایی که می‌توانند اطلاعات مفیدی در اختیار شما بگذارد در ادامه آورده شده است:

<http://tjzhifei.github.io/links/EMFA.pdf>

<http://people.csail.mit.edu/dsontag/courses/ml12/slides/lecture12.pdf>

پاسخ:



با توجه با شکل فوق وقتی تعداد کلاس‌ها بیشتر از دو تا باشد آنگاه وقتی از روش لجسيك رگرسيون برای کلاس‌بندی داده‌ها به کمک all-vs-all استفاده می‌کنیم، فضایی بین کلاس‌های مجاور ایجاد می‌شود که اگر نمونه‌ای در آن فضا قرار بگیرد آن نمونه روی مرز تصمیم قرار می‌افتد و کلاس آن نمونه غیر قابل تشخیص می‌شود، برای حل این مشکل یا حداقل کم کردن فضای مرز تصمیم احتمالاً بتوان به کمک روش ماشین‌های بردار پشتیبان (SVM) مرز تصمیم را بین کلاس‌های مجاور تقسیم کرد تا مرز تصمیم کوچک‌تر شود.