

هو العليم

پروژه ی تشخیص اعداد دست نوشته

استاد: مهندس تورانی

اعضای گروه:

هلیار سولی

زهرابختیار

بهاره بهروزی

سیده زهرافلاح میرموسوی اجداد

گزارش فعالیت های اولیه:

ابتدا با زبان پایتون و سپس با jupyter notebook آشنا شدیم:

<https://jupyter.org/>

و در مرحله ی بعد جستجو در اینترنت و جمع آوری اطلاعات را انجام دادیم.

شروع کار:

MNIST Dataset : این مجموعه داده شامل تصاویری از ارقام دست نویس انگلیسی مانند تصاویر زیر است که

به کوشش Lecun جمع آوری شده است:



با مجموعه ی داده ی مورد نیاز خود را از لینک ایشان برداشتیم:

<http://yann.lecun.com/exdb/mnist/>

این مجموعه شامل ۶۰۰۰۰ داده ی آموزشی که هر کدام نشانه ی تصویری با 28×28 پیکسل است. و همچنین ۱۰۰۰۰ داده ی

تست داریم. هم چنین در این مجموعه، برچسب هایی برای هر تصویر وجود دارد که میگوید این است که هر تصویر نمایانگر چه

رقمی است.

کتابخانه های مورد استفاده:

* `Matplotlib.pyplot`: یک کتابخانه گرافیکی برای ترسیم نمودار های دوبعدی است.

* `Pandas`: یکی از ابزار های رایج `data scientists` برای انجام تجزیه و تحلیل داده ها است، یکی از

کتابخانه های اساسی برای محاسبات علمی در پایتون به شمار می رود. ساختار ساده، سریع و انعطاف پذیر `Pandas` تجزیه و تحلیل داده ها در پایتون را بطور قابل توجهی آسان تر کرده است.

* `numpy`: یکی از اصلی ترین کیچ های محاسبات علمی در پایتون است که امکان ایجاد آرایه، ماتریس ها، مجموعه ها و ... را به ما می دهد.

* `scipy.ndimage`: توابع عملیاتی لازم جهت انجام عملیات روی آرایه های چندبُعدی (آرایه هایی که برای نمایش تصاویر، در قالب آرایه، مورد استفاده قرار می گیرند) را فراهم می آورد.

پیاده سازی توابع:

`calculateDigitsAccuracy`:

آرکومان: ۱. آرایه ای شامل ارقام حدس زده شده توسط الگوریتم که قبلاً آموزش دیده

۲. آرایه ای شامل برچسب هایی که مقادیر حقیقی ارقام را نمایش می دهد.

خروجی: درصد دستی حدس های الگوریتم مورد نظر.

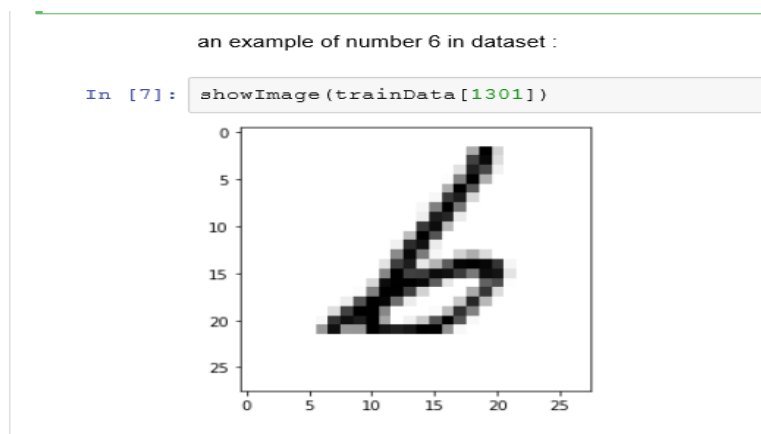
:calculateLettersAccuracy

مشابه تابع بالا برای محاسبه ی درصد دستی برای تشخیص حروف.

:showImage

آرکومان: آرایه ای شامل ۷۸۴ عضو از پیکسل ها نشان دهنده ی رقم.

کارکرد: نمایش تصویر مورد نظر.



:showPlot

آرکومان: ۱. مجموعه ی زوج مرتب هایی که باید روی نمودار نمایش داده شود. ۲. برچسب محور افقی ۳. برچسب محور عمودی

کارکرد: رسم نمودار بر اساس زوج مرتب های داده شده

:compareScores

آرکومان : ۱. مجموعه نقاطی که میخواهیم روی آنها مقایسه صورت گیرد. (نقاط روی محور افقی) ۲. مجموعه نقاط روی محور عمودی

مربوط به `trainscore` به ازای x های مختلف ۳. مجموعه نقاط روی محور عمودی مربوط به `testscore` به ازای

x های مختلف ۴. برچسب محور افقی ۵. برچسب محور عمودی

کارکرد: رسم دو نمودار در یک صفحه ی مختصات به منظور مقایسه

K-Nearest Neighbors (k نزدیک ترین همسایگی)

این الگوریتم یک نمونه تستی را بر اساس k همسایه نزدیک دسته بندی می کند. نمونه های آموزشی به عنوان بردارهایی در فضای ویژگی چند بعدی مطرح می شوند. فضا به ناحیه هایی با نمونه های آموزشی پارتیشن بندی می شود. یک نقطه در فضا به کلاسی تعلق می یابد که بیشترین نقاط آموزشی متعلق به آن کلاس در داخل نزدیک ترین نمونه ی آموزشی به k در آن باشد.



```
In [23]: from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(n_neighbors=12)
clf.fit(trainData, trainLabels)

predictedTrain = clf.predict(trainData)
predictedTest = clf.predict(testData)

trainAcc = calculateDigitsAccuracy(predictedTrain, trainLabels)
testAcc = calculateDigitsAccuracy(predictedTest, testLabels)
```

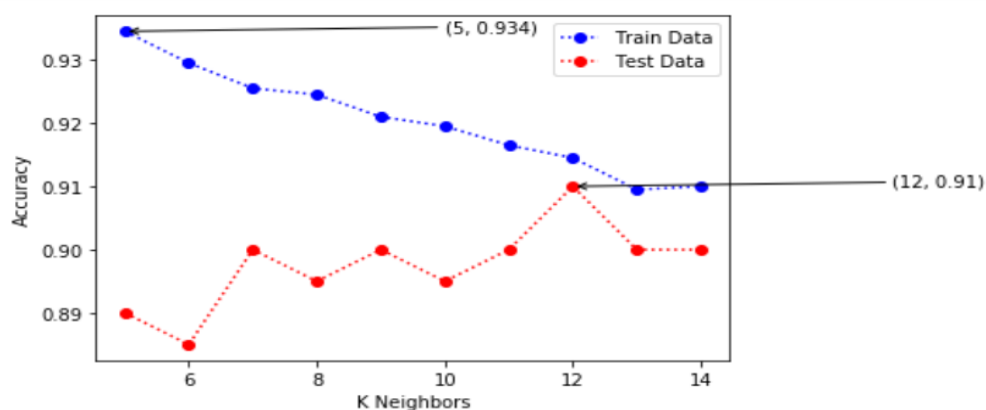
کتابخانه `scikit Learn` این امکان را فراهم آورده تا از این روش استفاده کنیم. ابتدا پارامتر

`n_neighbors` تعداد همسایه های یعنی همان `k` را تعیین میکنیم. و با تابع `fit` آموزش داده میشود و بعد با `predict`

انتظار داریم به ازای ورودی های ما به ما رقم درست را در خروجی بدهد.

برای تعیین مناسب ترین تعداد همسایه های می توان از آزمایش اعداد مختلف استفاده کرد. در ادامه ما درصد درستی تخمین را

برای `k` های مختلف مقایسه کرده ایم.



یکی دیگر از توابعی که استفاده کردیم

```
clf.kneighbors([trainData[1042]],return_distance=False)
```

همسایه های غنصری که به عنوان آرگومان اولش داده شده را بر میگردداند و در صورت نیاز فاصله را نیز بر میگردداند.

Decision tree (درخت تصمیم)

ابزاری برای پشتیبانی از تصمیم است که از درخت های برای مدل کردن استفاده می کند. درخت تصمیم به طور معمول در

تحقیق‌ها و عملیات مختلف استفاده می‌شود. به طور خاص در آنالیز تصمیم، برای مشخص کردن استراتژی که با بیشترین احتمال به هدف برسد بکار می‌رود. در این درخت هر گره داخلی نشان دهنده ی یک تست بر روی یک ویژگی و هر شاخه نشان دهنده ی نتیجه ی تست و هر برگ نشان دهنده ی برچسب کلاسی می‌باشد. تمام مسیرها از ریشه به برگ نشان دهنده ی قوانین طبقه‌بندی می‌باشند.

DecisionTreeClassifier یک کلاس با قابلیت انجام چند طبقه‌بندی بر روی مجموعه داده‌ها است. یکی از پارامترهای مهم آن حداکثر عمق درخت (**max_depth**) است. که در ادامه با دادن عمق‌های متفاوت نرخ درستی را به ازای مقادیر مختلف عمق بررسی کردیم.

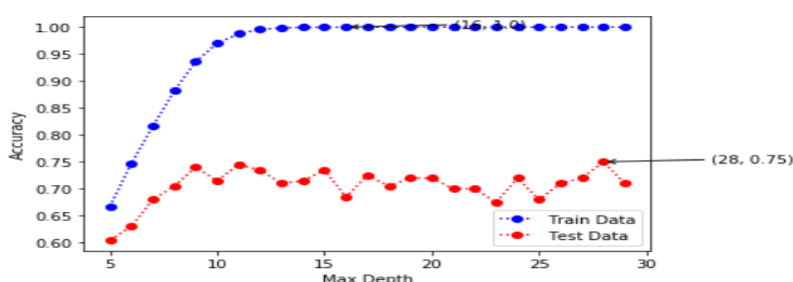
```
X = range(5, 30)

for depth in X:
    clf = tree.DecisionTreeClassifier(max_depth=depth)
    clf.fit(trainData, trainLabels)

    predictedTrain = clf.predict(trainData)
    predictedTest = clf.predict(testData)

    trainAcc = calculateDigitsAccuracy(predictedTrain, trainLabels)
    testAcc = calculateDigitsAccuracy(predictedTest, testLabels)
```

بعد از fit شدن، می‌توان از این مدل برای پیش‌بینی سطح نمونه‌ها استفاده کرد.

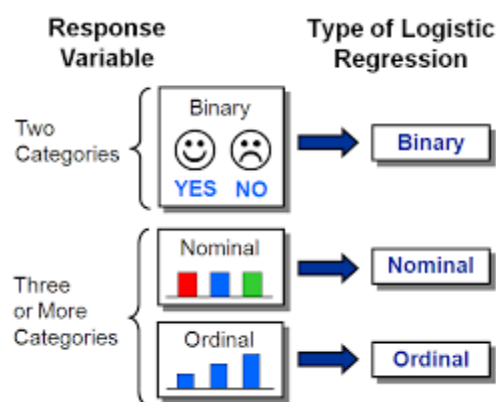


Logistic Regression (رگرسیون لجستیک)

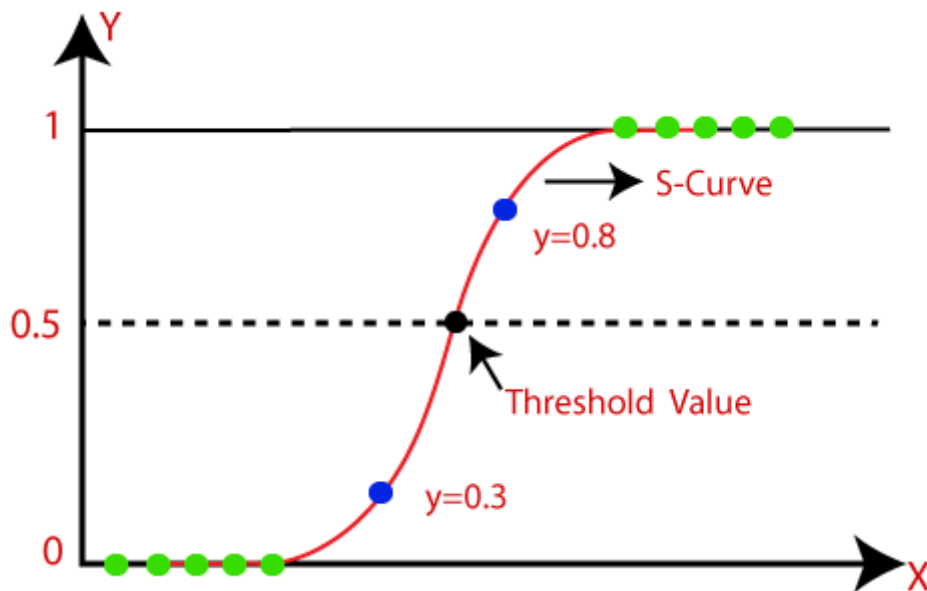
Logistic Regression یک روش «دسته‌بندی» (Classification) «تطارت شده» (Supervised Learning) در پاتون است که در تخمین مقادیر گسته مانند ۱/۰، بله / خیر و درست / غلط کاربرد دارد. این کار بر بنای مجموعه‌ای از متغیرهای مستقل انجام می‌شود. از تابع لجستیک برای پیش‌بینی احتمال یک رویداد استفاده می‌شود و به کار بر یک خروجی بین ۰ و ۱ می‌دهد.

انواع رگرسیون لجستیک:

1. Binary logistic regression (رگرسیون لجستیک باینری یا دو وجهی)
2. Multinomial logistic regression (رگرسیون لجستیک چند سطحی یا چند وجهی)
3. Ordinal logistic regression (رگرسیون لجستیک ترتیبی)



این روش را برای کلاس‌بندی اعداد و حروف به کار بردیم.



MLP

پرسترون چند لایه، (Multilayer perceptron) دسته ای از شبکه های عصبی مصنوعی پیشخور است.

یک MLP شامل حداقل سه لایه گره است: یک لایه ورودی، یک لایه پنهان و یک لایه خروجی. به جز گره های ورودی، هر گره یک نورون است که از یک تابع فعل سازی غیر خطی استفاده می کند. لایه های متعدد آن و فعال سازی غیر خطی آن MLP را از یک پرسترون خطی متمایز می کند. در واقع می تواند داده هایی را متمایز کند که به صورت خطی قابل تمایز نیستند.

یادگیری در شبکه عصبی با تغییر وزن اتصال پس از پردازش هر قطعه از داده ها، بر اساس میزان خطا در خروجی در مقایسه با نتیجه مورد انتظار رخ می دهد. این نمونه که از یادگیری با نظارت و از طریق بازگشت به عقب و تعمیم الگوریتم حداقل مربعات در

پرسترون خطی انجام می شود. خطای موجود در گره خروجی j در n -امین نقطه داده به صورت

$$e_j(n) = d_j(n) - y_j(n)$$

نشان می‌دهیم که در آن Δ مقدار هدف و η مقدار تولید شده توسط پرسپترون می‌باشد. مقادیر کره براساس تصحیحات تنظیم می‌شوند که میزان خطا در کل خروجی را به حداقل می‌رساند و به صورت زیر است:

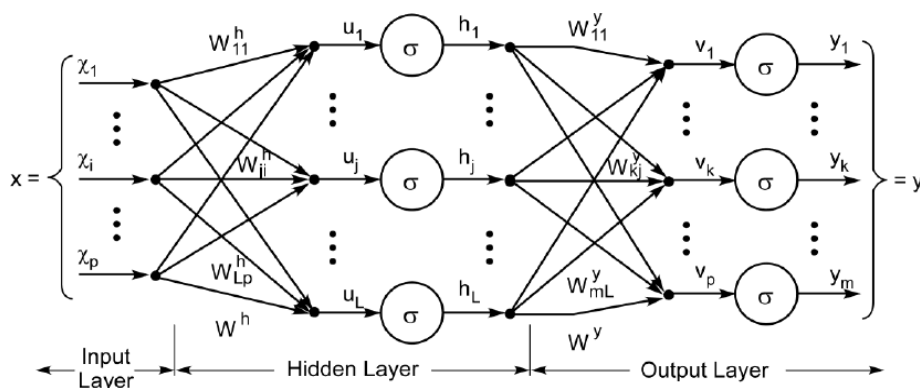
$$\mathcal{E}(n) = \frac{1}{2} \sum_j e_j^2(n)$$

با استفاده از گرادیان، تغییر در وزن به صورت زیر است:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

که در آن v_j خروجی نورون پیشین و η میزان یادگیری است که انتخاب شده تا اطمینان حاصل شود که وزن ها به سرعت به واکنش بدون نوسان همگرا می‌شوند.

ساختار شبکه های عصبی پرسپترون چند لایه به صورت زیر است:



کلاس `MLPClassifier` یک الگوریتم چند لایه پرسپترون را اجرا می‌کند که با استفاده از `backpropagation` آموزش می‌دهد.

از پارامترهای مهم آن: `hidden_layer_size` (عنصر نام نشان دهنده ی تعداد نورون های پنهان نام است)، `shuffle` (این که نمونه های `shuffle` در هر تکرار جابجا شوند یا نه)، `momentum` (نیروی حرکت برای به روز رسانی گرادینت نزولی بایده بین تا باشد)، `activation` (فعال کردن تابع فعال سازی برای لایه ی پنهان)، `max_iter` (حد اکثر تعداد تکرار ها)، `learning_rate_init` (نرخ یادگیری اولیه مورد استفاده قرار می گیرد این دستگاه اندازه گام را در به روز رسانی وزن ها کنترل می کند)

توابع

`Def predicts(weights, testdata)`

آرکومان ۱: وزن هایی که با آموزش الگوریتم بدست آمده است.

۲- داده های تست (همان مقادیری که حدس میزنیم چه رقمی است).

مقدار بازگشتی: آریه ای از بزرچسب های حدس زده شده (ارقام حدس زده شده)

نحوه ی کار: به همه ی تصاویر یا همه ی پارامترها (θ) یک ۱ اضافه میکنیم که همان θ .

است و پس با ضرب نقطه ای وزن ها در مقادیر داده های تست اعمال تابع فعال سازی (signum) مقادیری بدست می آید که رقم مورد نظر آن است که بیشترین مقدار به ازای لیبل تطبیق

بدست آمده.

تابع فعال سازی:

از تابع علامت استفاده کردیم.

تابع learning:

این تابع در تابع train به کار میرود و برای بدست آوردن وزن ها به ازای هر لیبل در نهایت call میشود. مقدار گردش های مورد نیاز (epochs) که در ابتدای برنامه مشخص کردیم ، این عمل را تکرار میکنیم. وزن های اولیه را ضرب نقطه ای در داده های آموزشی کرده و تابع فعال سازی را بر روی آن اعمال میکنیم. اگر حاصل با برچسب تناظر برابر نبود، یعنی اشتباه

حدس زده شده. پس کرادیان را محاسبه میکنیم و آنرا در نرخ یادگیری ضرب میکنیم و از وزن فعلی کم میکنیم (روش کرادیان کاهشی) و این عمل را برای همه ی داده ها انجام میدهیم و در نهایت وزن های بهتر را برمیگردانیم.

تابع train:

آرکومان ها: ۱- داده های آموزشی ۲- برچسب های متناظر داده های آموزشی

این تابع وظیفه ی آموزش الگوریتم را برعهده دارد.

که به ازای هر label تابع learn را اجرا کرده و وزن را برای دسته های مختلف بدست می آورد.

تابع main:

تابع run:

سجش های مختلف عملکرد برنامه را گزارش میدهد و زبان اجرا را نشان میدهد.
