هوالعلیم

پروژه ی تشخیص اعداد دست نوشته

استاد : مهندس تورانی

اعضای گروه:

هلیا رسولی

زهرا بختیار

بهاره بهروزی

سیده زهرا فلاح 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 دست‌نویس انگلیسی مانند تصاویر زیر است که به کوشش Lecun جمع آوری شده است:

ما مجموعه ی داده ی موردنیاز خود را از لینک ایشان برداشتیم :

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

این مجموعه شامل ۶۰۰۰۰ داده ی آموزشی که هرکدام نشانده ی تصویری با ۲۸*۲۸ پیکسل است. و همچنین ۱۰۰۰۰ داده ی تست داریم . هم چنین در این مجموعه، برچسب هایی برای هر تصویر وجود دارد که بیانگر این است که هر تصویر نمایان گر چه رقمی است.

کتابخانه های مورد استفاده :

* Matplotlib.pyplot : یک کتابخانه گرافیکی برای ترسیم نمودار های دو بعدی است .

* Pandas : یکی از ابزارهای 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 پکیج 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:

آرگومان : ۱. آرایه ای شامل ارقام حدس زده شده توسط الگوریتم که قبلا آموزش دیده

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

خروجی : درصد درستی حدس های الگوریتم مورد نظر.

calculateLettersAccuracy:

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

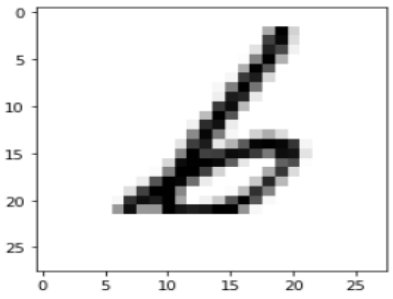
showImage:

آرگومان : آرایه ای شامل ۷۸۴ عضو از پیکسل ها نشاندهنده ی رقم.

کارکرد: نمایش تصویر مورد نظر.

an example of number 6 in dataset :

```
In [7]: showImage(trainData[1301])
```



showPlot:

آرگومان: 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

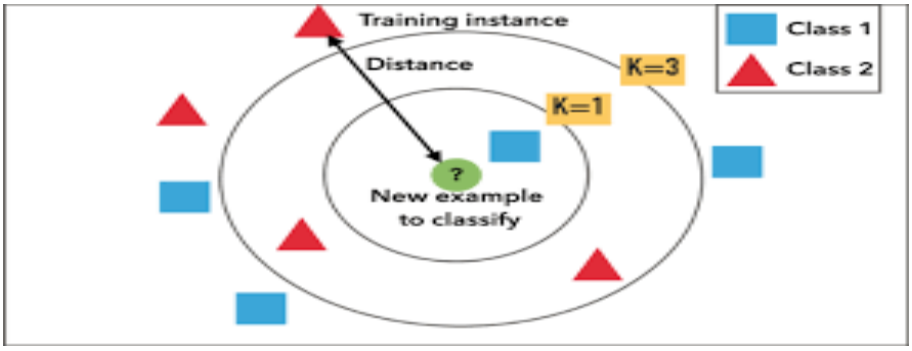
compareScores :

آرگومان : 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)

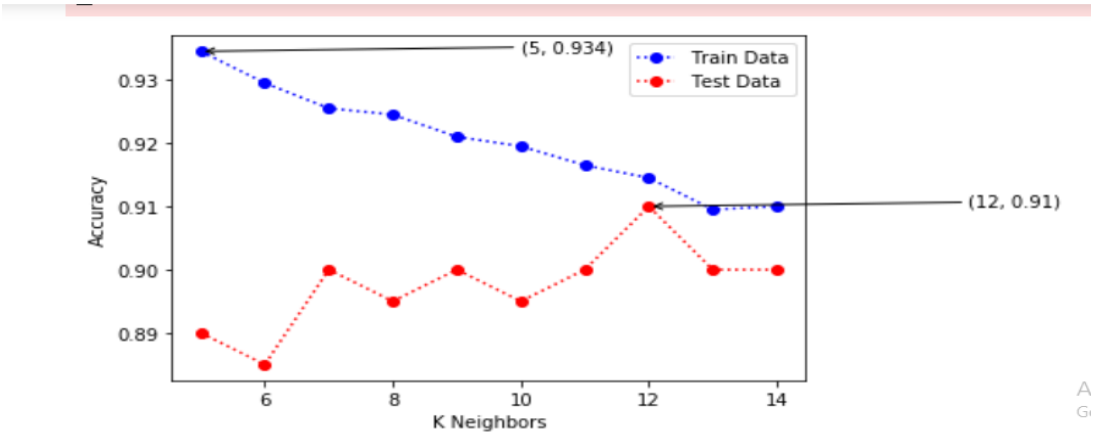
این الگوریتم یک نمونه تستی را بر اساس k همسایه نزدیک دسته‌بندی می‌کند. نمونه‌های آموزشی به عنوان بردارهایی در فضای ویژگی چندبعدی مطرح می‌شوند. فضا به ناحیه‌هایی با نمونه‌های آموزشی پارتیشن‌بندی می‌شود. یک نقطه در فضا به کلاسی تعلق می‌یابد که بیشترین نقاط آموزشی متعلق به آن کلاس در داخل نزدیک‌ترین نمونه‌ی 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 در پارامتر n_neighbors تعداد همسایه ها یعنی همان k را تعیین میکنیم . و با تابع fit آموزش داده میشود و بعد با predict انتظار داریم به ازای ورودی ها به ما رقم درست را در خروجی بدهد.

برای تعیین مناسب ترین تعداد همسایه ها 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 عنصری که به عنوان آرگومان اولش داده شده را برمیگرداند و درصورت نیاز فاصله را نیز برمیگرداند.

Decision tree(درخت تصمیم)

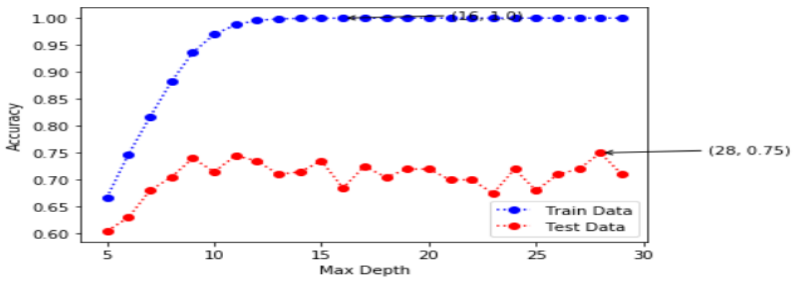
ابزاری برای پشتیبانی از تصمیم است که از درخت‌ها برای مدل کردن استفاده می‌کند. درخت تصمیم به طور معمول در

تحقیق‌ها و عملیات مختلف استفاده می‌شود. به‌طور 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 طبقه بندی 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 عمق های 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.

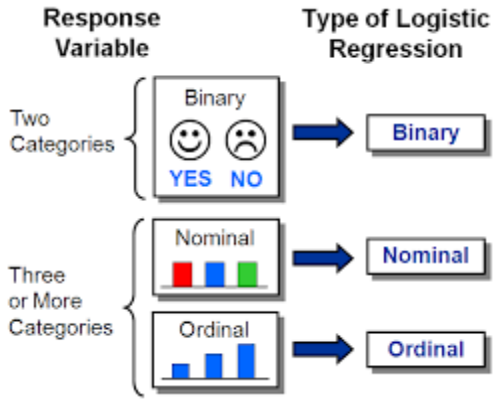


Logistic Regression(رگرسیون لجستیک)

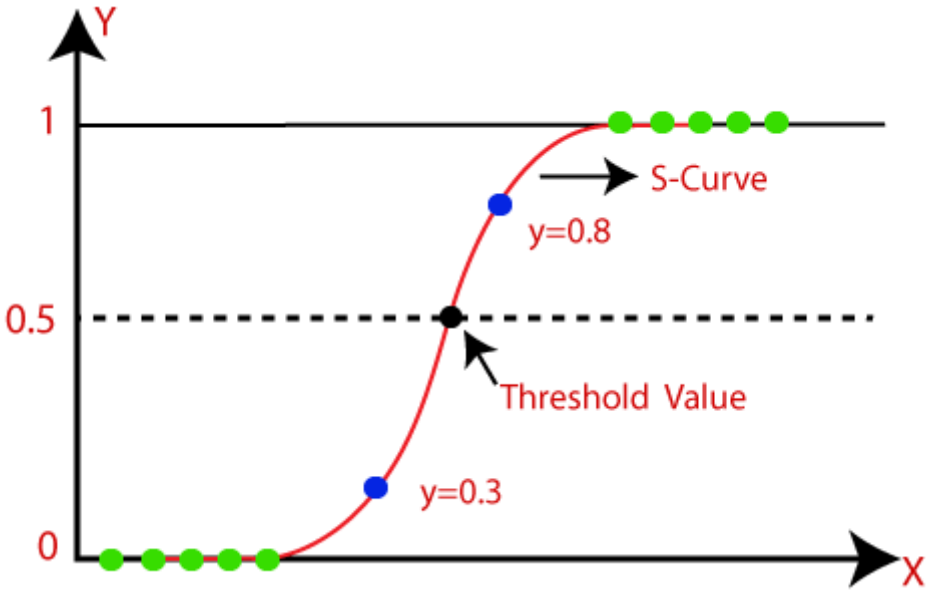
(Logistic Regression)یک روش «دسته‌بندی» (Classification) «نظارت شده» (SupervisedLearning)در پایتون است که در تخمین مقادیر گسسته مانند ۰/۱، بله/خیر و درست/غلط کاربرد دارد. این کار بر مبنای مجموعه‌ای از متغیرهای مستقل انجام می‌شود. از تابع لجستیک برای پیش‌بینی احتمال یک رویداد استفاده می‌شود و به کاربر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

انواع رگرسیون لجستیک:

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



این روش را برای کلاس بندی اعداد و حروف به کار بردیم.



MLP

پرسپترون 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 گره‌های ورودی، هر گره یک نورون است که از یک تابع فعلسازی غیر خطی استفاده می‌کند.. لایه‌های متعدد آن و فعال سازی غیر خطی آن MLP را از یک پرسپترون خطی متمایز می‌کند. در واقع می تواند داده‌هایی را متمایز کند که به صورت خطی قابل تفکیک نیستند.

یادگیری در شبکه عصبی با تغییر وزن اتصال پس از پردازش هر قطعه از داده‌ها، براساس میزان خطا در خروجی در مقایسه با نتیجه مورد انتظار رخ می‌دهد. این نمونه که از یادگیری با نظارت و از طریق بازگشت به عقب و تعمیم الگوریتم حداقل مربعات در پرسپترون خطی انجام می‌شود. خطای موجود در گره خروجی j را در n-امین نقطه داده به صورت $e_j(n) = d_j(n) - y_j(n)$

نشان می دهیم که در آن d_k مقدار هدف و y_k مقدار تولید شده توسط پرسپترون می باشد. مقادیر گره براساس تصحیحات تنظیم می شوند که میزان خطا در کل خروجی را به حداقل می رساند و به صورا زیر است :

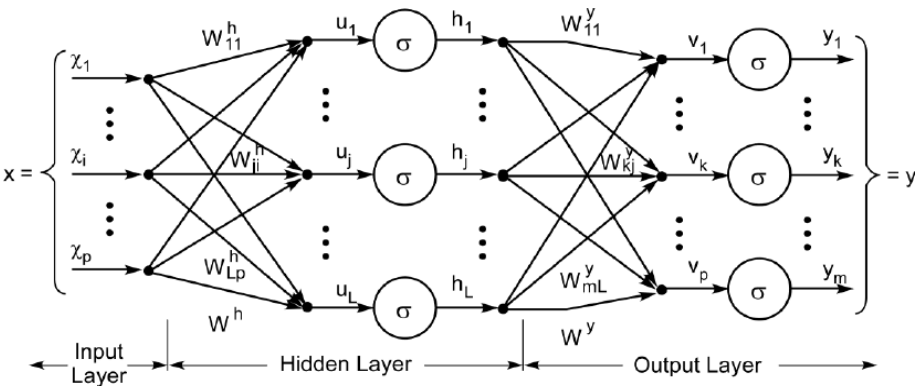
$$\mathcal{E}(n) = \frac{1}{2}\sum_{j} e_j^2(n)$$

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 y_i 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 می شوند .

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 پارمترهای مهم آن: **hidden_layer_size** (عنصر iام نشان دهنده ی تعداد نورون ها در لایه ی پنهان iام است)، **shuffle** (این که نمونه‌های shuffle در هر تکرار جابجا شوند یا نه)، **momentum** (نیروی حرکت برای به روز رسانی گرادیان نزولی باید بین ۰ تا ۱ باشد)، **activation** (فعال کردن تابع فعالسازی برای لایه ی پنهان)، **max_iter** (حداکثر تعداد تکرار ها)، **learning_rate_init** (نرخ یادگیری اولیه مورد استفاده قرار می گیرد این دستگاه اندازه گام را در به روز رسانی وزن‌ها کنترل می‌کند)

توابع

Defpredicts(weights,testdata)

آرگومان ها: ۱_ وزن هایی که با آموزش الگوریتم بدست آمده است.

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

مقدار بازگشتی: آرایه ای از برچسب 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_0.

است و سپس 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 نظیرش 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 موردنیاز (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 فعالسازی را بر روی آن اعمال میکنیم .اگر حاصل با برچسب متناظر برابر نبود، یعنی اشتباه

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

تابع train:

آرگومان 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:

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