به نام خدا
خلاصه فصل ۳ کتاب HANDS-ON MACHINE LEARNING

CLASSIFICATION

- قبلا اشاره کردیم که از مهم‌ترین کارهایی که با supervised learning انجام میدیم، regressigon و classification هستش. تو فصل قبل با regression به طور کلی آشنا شدیم حالا سراغ classification میریم.
- اول از همه دیتاستی که باهاش این فصل کار داریم mnist هستش که شامل ۷۰۰۰۰ عکس از رقم های ۰ تا ۹ که به صورت دست خط نوشته شدند هستش. هر عکس با لیبل هستش که نشون میده چه عددی رو داخل عکس پرزنت می کنه. به این طریق دانلود می‌کنیم:

```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', as_frame=False)
```

The sklearn.datasets package contains mostly three types of functions: fetch_* functions such as fetch_openml() to download real-life datasets, load_* functions to load small toy datasets bundled with Scikit-Learn (so they don't need to be downloaded over the internet), and make_* functions to generate fake datasets, useful for tests. Generated datasets are usually returned as an (X, y) tuple containing the input data and the targets, both as NumPy arrays. Other datasets are returned

- اگه بخوایم میتونیم یه نگاه کلی به دیتامون بندازیم:

```
>>> X, y = mnist.data, mnist.target
>>> X
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
>>> X.shape
(70000, 784)
>>> y
array(['5', '0', '4', ..., '4', '5', '6'], dtype=object)
>>> y.shape
(70000,)
```

وقتی x shape رو نگاه میکنیم میبینیم که ۷۰۰۰۰ هزار عکسمون داخلشه که هر عکس ۷۸۴ تا فیچر داشته. در اصل به خاطر اینه عکسمون ۲۸*۲۸ هستش که شامل ۷۸۴ تا پیکسل (فیچر) میشه. که هر پیکسل یک عدد بین ۰ تا ۲۵۵ رو داره که ۰ نشون دهنده سفید و ۲۵۵ مشکی کامل هستش. پس طبعا عکسامون سیاه سفیده.

CONTINUE

- برای اینکه بتونیم یه عکس رو پلات کنیم و ببینیم میایم اول شکل آرایه یه بعدی رو به یه آرایه دو بعدی ۲۸ در ۲۸ تبدیل میکینیم:

```
import matplotlib.pyplot as plt

def plot_digit(image_data):
    image = image_data.reshape(28, 28)
    plt.imshow(image, cmap="binary")
    plt.axis("off")

some_digit = X[0]
plot_digit(some_digit)
plt.show()
```

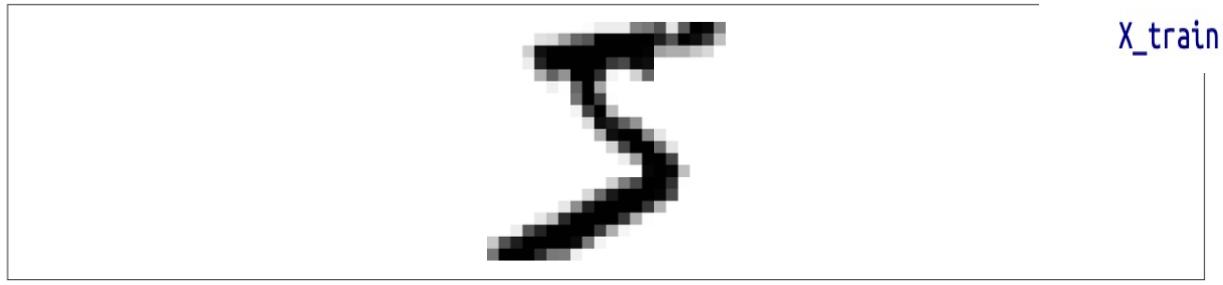


Figure 3-1. Example of an MNIST image

This looks like a 5, and indeed that's what the label tells us:

```
>>> y[0]
'5'
```

همونطور که میبینیم لیبل‌ش هم داخل دیتاست عدد ۵ عه که با شکلی که پلات کردیم تطابق داره. خوبی این دیتاست این هستش که به طور خوبی شافل شده و میتونیم تا یه ایندکس خاصی رو برای train کردن و از اون اینکدس به بعد رو برای test برداریم:

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

CONTINUE

- مسئله رو ساده می‌کینم اول به این می‌پردازیم که یک مدل رو طراحی کنیم که توانایی اینو داشته باشه بتونه تشخیص بده یک عدد ۵ هست یا نه اول از همه باید یک training set برای این کار تشکیل بدیم

```
y_train_5 = (y_train == '5')  # True for all 5s, False for all other digits
y_test_5 = (y_test == '5')
```

- حالا وقت انتخاب یک مدل هستش. برای شروع SGD (Stochastic Gradient Descent) رو انتخاب میکنیم. این classifier یک قدرتی که داره اینه میتونه دیتاست های خیلی بزرگ رو به طور افیشنت هندل کنه. دلیلش هم از روس اسمش مشخصه بخاطر اینه که دیتا هارو دونه به دونه میبینه و یادمیگیره و به همین دلیل برای سیستم های online learning مناسبه.

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

- حالا میتونیم ببینیم مدلمون روی عکس اولیه که پلات کردیم چی کار میکنه

Now we can use it to detect images of the number 5:

```
>>> sgd_clf.predict([some_digit])
array([ True])
```

PERFORMANCE MEASURES

- فهمیدن پرفورمنس یک classifier یه ذره مشکل تر از پرفورمنس regressor هستش. بخش زیادی از این فصل هم به همین مبحث پرداخته.
- ۱. Measuring Accuracy Using Cross-Validation : بیایم اول با cross validation که فصل قبل هم دیدیم شروع کنیم کاری که میکرد training set رو به k فولد تقسیم میکرد. بعد مدلمون رو k بار train میکنه و هر سری یک دونه از اون فولد هارو برای ارزیابی به کار میبره

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.95035, 0.96035, 0.9604 ])
```

- به نظر میرسه مدل خوبیه تو ۹۵ درصد مواقع درست عمل کرده حالا بیایم از dummy classifier استفاده کنیم که هر نمونه رو به کلاسی که بیشترین تکرار رو داره نسبت میده (واقعا احمقانس). که تو مثال ما بیشترین نمونه متعلق به کلاس غیر ۵ هستش نه کلاس ۵. پس هر عکس رو میاد به راحتی میگه ۵ نیست. ببینیم این چه قدر دقت داره:

```
from sklearn.dummy import DummyClassifier

dummy_clf = DummyClassifier()
dummy_clf.fit(X_train, y_train_5)
print(any(dummy_clf.predict(X_train)))  # prints False: no 5s detected
```

Can you guess this model's accuracy? Let's find out:

```
>>> cross_val_score(dummy_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.90965, 0.90965, 0.90965])
```

That's right, it has over 90% accuracy! This is simply because only about 10% of the images are 5s, so if you always guess that an image is *not* a 5, you will be right about 90% of the time. Beats Nostradamus.

CONTINUE

- مثال قبل نشون داد صرفا یک عدد خیلی معیار خوبی برای اینکه بگیم مدل خوبی هست یا نه نیست. یه راه بهتر اینه از CM (confusion matrix) استفاده کنیم. ایده کلی این ماتریس اینه بیاد برای هر کلاس، مثلا a و b داخل دیتاست یه ماتریس درست کنه یه درایه ab معنیش اینه چند تا نمونه بودن که در اصل a بودن ولی b پردیکت شدن. پس با همون کانسپت cross validation خروجی‌هاش رو سیو میکنیه.

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

- حالا میتونیم ماتریس رو تشکیل بدیم:

```
>>> from sklearn.metrics import confusion_matrix
>>> cm = confusion_matrix(y_train_5, y_train_pred)
>>> cm
array([[53892,   687],
       [ 1891,  3530]])
```

- اولین درایه داره بهمون میگه ۵۳۸۹۲ تا عکس بوده که ۵ نبودن و به درستی هم همچین نتیجه ای توسط مدل پردیکت شده. در حالی که ۶۸۷ عکس بوده که متعلق به ۵ شناسایی شده در حالی که ۵ نبودن (خطا نوع ۱). ۱۸۹۱ عکس بودن که غیر ۵ پردیکت شدن در حالی که ۵ بودن (خطا نوع ۲) و ۳۵۳۰ تا عکس بوده که ۵ پردیکت شده و واقعا ۵ هم بودن.

PRECISION AND RECALL

- طبیعتا یه مدل وقتی خوبی که فقط روی اعضا قطر اصلی مقدار داشته باشه. حالا میتونیم یه سری معیار ها مثل precision برای classifierمون تعریف کنیم به این شکل که میاد مقدار هایی که به درستی و به اشتباه به کلاس ۵ مپ شده توسط مدل رو محاسبه میکنه اونایی که درست بوده رو اسمش رو میذاره true positive و غلط هارو false positive میذاره.

Equation 3-1. Precision

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

TP is the number of true positives, and *FP* is the number of false positives.

البته این معیار به تنهایی خیلی خوب نیست چرا؟ فرض کنیم یه مدل باشه بیاد روی یه نمونه که خیلی مطمئنه ۵ هستش tp بده و بقیه رو همه بگه ۵ نیستن اینجوری accuracy ۱۰۰ درصد میده بهمون. پس به یه معیار دیگه به اسم recall این معیار میاد ببینه چند تا پیش بینی درست داشتیم برای کیس هایی که یه عدد رو ۵ پردیکت کردیم و واقعا ۵ بوده (همونTP) و چند تا نمونه بوده که ۵ بودن ولی اشتباهی پردیکت کردیم که ۵ نیست (FN).

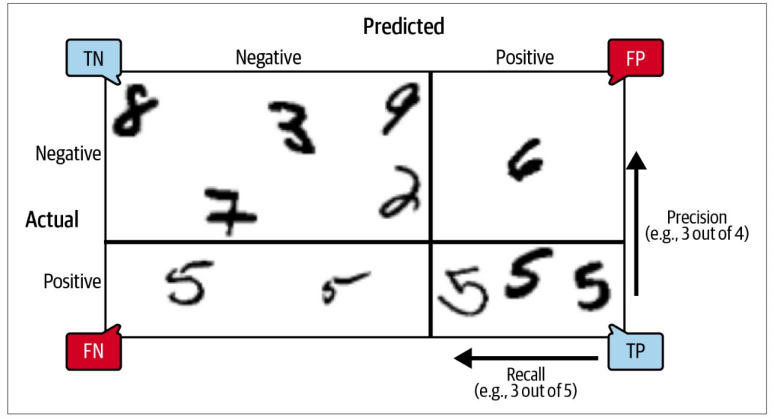


Figure 3-3. An illustrated confusion matrix showing examples of true negatives (top left), false positives (top right), false negatives (lower left), and true positives (lower right)

CONTINUE

به لطف scikit-learn میتونیم خیلی راحت این دو معیار رو حساب کنیم.

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred)  # == 3530 / (687 + 3530)
0.8370879772350012
>>> recall_score(y_train_5, y_train_pred)  # == 3530 / (1891 + 3530)
0.6511713705958311
```

مثل اینکه خیلی مدل خوبی هم نداشتیم چون فقط ۶۵ درصد عکس هایی که ۵ بودن رو تونسته به درستی پیش‌بینی کنه که ۵ هستن.

بعضی وقت ها نیازه تا دو مدل رو صرفا با یه معیار برسی کنیم که اومدن یه F1 درست کردن که این شکلی حساب میشه:

Equation 3-3. F_1 score

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{FN + FP}{2}}$$

To compute the F_1 score, simply call the `f1_score()` function:

```
>>> from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred)
0.7325171197343846
```

این معیار به جای استفاده از regular mean که عادی میانگین میگیره از harmonic mean استفاده میکنه که تنها در صورتی که هر دو تا معیارمون مقدار های بالایی داشته باشن مقدار بالایی داره که کاملا مناسب کار ما هستش.

CONTINUE

- البته اینکه واقعا کدوم معیار برامون مهمه خیلی خیلی به اون کاری که داریم انجام میدیم ربط داره. برای مثال اگه یه مدل قراره بفهمه یه محتوایی برای بچه ها مناسب هست یا نه برامون خیلی مهمه که مواردی که دیتکت میکنه که مناسب نیست شامل همه موارد باشه حتی به قیمت این باشه که یه سری مورد که مناسب هستن هم نمایش پیدا نکنه. در واقع باید یه trade off بین این دو معیار برقرار کنیم.
- برای اینکه بفهمیم این trade off چطوری برقرار هستش، بیایم به نحوه پیش‌بینی sgd یه نگاه بندازیم، برای هر نمونه میاد یه عدد حساب میکنه و از یه عدد به بالایی رو میگه ۵عه پایین تر از اون رو میگه ۵ نیست. به این عدد که مبنای تصمیم گیری ما هستش decision threshold میگن.

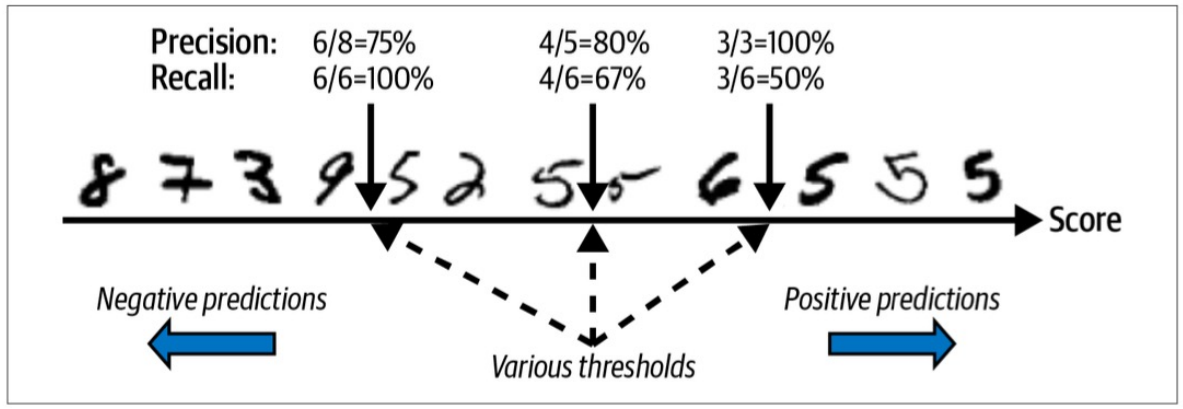


Figure 3-4. The precision/recall trade-off: images are ranked by their classifier score, and those above the chosen decision threshold are considered positive; the higher the threshold, the lower the recall, but (in general) the higher the precision

در واقع این trade off این شکلیه که decision threshold رو چند بذاریم این شکلی میزان درصد recall و precision تغییر میکنه. که دقیقا همین روال تو شکل رو به رو مشخصه. اگه ضریب اطمینان بالایی مثلا بخوایم برای نمونه هایی که میگیم ۵ هستن باید threshold رو بالا بذاریم.

CONTINUE

- مشکلی که هست اینه scikit learn بهمون اجازه نمیده به طور مستقیم threshold رو دستکاری کنیم. ولی همچین راه حل هایی برقراره:

```
>>> y_scores = sgd_clf.decision_function([some_digit])
>>> y_scores
array([2164.22030239])
>>> threshold = 0
>>> y_some_digit_pred = (y_scores > threshold)
array([ True])
```

The `SGDClassifier` uses a threshold equal to 0, so the preceding code returns the same result as the `predict()` method (i.e., `True`). Let's raise the threshold:

```
>>> threshold = 3000
>>> y_some_digit_pred = (y_scores > threshold)
>>> y_some_digit_pred
array([False])
```

- اما از کجا بفهمیم دقیقا چه thresholdای مناسب ما هستش؟ میایم از corss_val_predict استفاده میکنیم تا مقدار هایی که پردیکت شدن رو برگردونه بهمون صرفا بهش میگیم به جای مقادیر پیش‌بینی بهمون اون عددی که مبنا تصمیم گیری هر نمونه رو برگردونه.

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
```

With these scores, use the `precision_recall_curve()` function to compute precision and recall for all possible thresholds (the function adds a last precision of 0 and a last recall of 1, corresponding to an infinite threshold):

```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

CONTINUE

- حالا با استفاده از precision_recall_curve بیایم این دو معیار رو حساب کنیم و در نهایت پلات کنیم نتیجه رو به ازای threshold های مختلف:

با این thresholdای که بررسی کردیم فهمیدیم که مقادیر precision و recall مدلمون با یه threshold خاص چنده.

```
from sklearn.metrics import precision_recall_curve

precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

Finally, use Matplotlib to plot precision and recall as functions of the threshold value (Figure 3-5). Let's show the threshold of 3,000 we selected:

```
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
plt.vlines(threshold, 0, 1.0, "k", "dotted", label="threshold")
[...]  # beautify the figure: add grid, legend, axis, labels, and circles
plt.show()
```

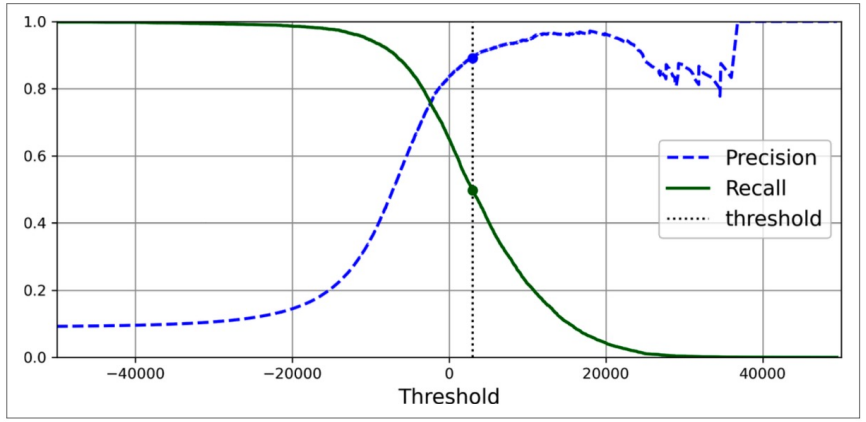


Figure 3-5. Precision and recall versus the decision threshold

```
plt.plot(recalls, precisions, linewidth=2, label="Precision/Recall curve")
[...]  # beautify the figure: add labels, grid, legend, arrow, and text
plt.show()
```

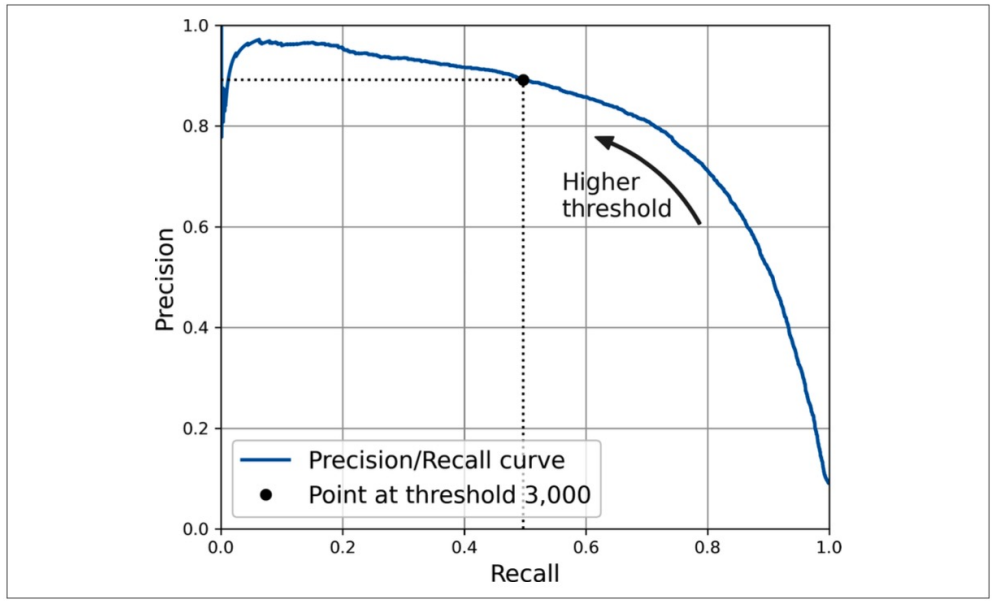


Figure 3-6. Precision versus recall

CONTINUE

- حالا مثلا فرض کنیم به این نتیجه رسیدیم پروژمون باید 90% precision داشته باشه میتونیم از از داخل نمودار هایی که کشیدیم پیدا کنیم که خیلی کار دقیقی نیست ولی میتونیم بیایم این شکلی پیداش کنیم با استفاده از argmax:

```
>>> idx_for_90_precision = (precisions >= 0.90).argmax()
>>> threshold_for_90_precision = thresholds[idx_for_90_precision]
>>> threshold_for_90_precision
3370.0194991439557
```

۲. ROC Curve: این نمودار میاد recall رو برحسب flase positive rate (نسبت نمونه هایی که ۵ نبودن اما ۵ پیش‌بینی شدن) رو حساب میکنه

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

```
idx_for_threshold_at_90 = (thresholds <= threshold_for_90_precision).argmax()
tpr_90, fpr_90 = tpr[idx_for_threshold_at_90], fpr[idx_for_threshold_at_90]

plt.plot(fpr, tpr, linewidth=2, label="ROC curve")
plt.plot([0, 1], [0, 1], 'k:', label="Random classifier's ROC curve")
plt.plot([fpr_90], [tpr_90], "ko", label="Threshold for 90% precision")
[...]  # beautify the figure: add labels, grid, legend, arrow, and text
plt.show()
```

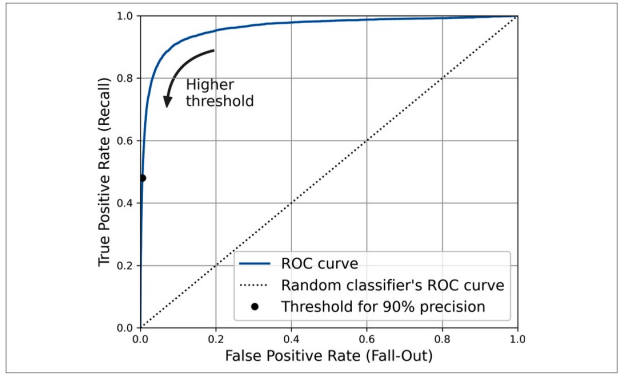


Figure 3-7. A ROC curve plotting the false positive rate against the true positive rate for all possible thresholds; the black circle highlights the chosen ratio (at 90% precision and 48% recall)

CONTINUE

- یک معیار برای برسی خوب بودن دو تا مدل اینه سطح زیر نمودار ROC رو حساب کنیم هرکدوم بزرگ تر باشه مدل بهتریه (چون یه thresholdی داره که همزمان recall بالا و FPR پایینی داره)

```
>>> from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.9604938554008616
```

- بیایم یه randomForestClassifier استفاده کنیم و با مدل قبلی مقایسش کنیم روی معیارهایی که تا الان برسی کردیم.

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(random_state=42)
```

```
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")
```

- حالا اگه خروجی مدل رو برای دو تا دیتا اولیمون ببینیم این شکلیه که اولین عکس رو به احتمال ۸۹ درصد میگه ۵ هست و دومین عکس رو به احتمال ۹۹ درصد میگه ۵ نیست

```
>>> y_probas_forest[:2]
array([[0.11, 0.89],
       [0.99, 0.01]])
```

CONTINUE

The second column contains the estimated probabilities for the positive class, so let's pass them to the `precision_recall_curve()` function:

```
y_scores_forest = y_probas_forest[:, 1]
precisions_forest, recalls_forest, thresholds_forest = precision_recall_curve(
    y_train_5, y_scores_forest)
```

Now we're ready to plot the PR curve. It is useful to plot the first PR curve as well to see how they compare (Figure 3-8):

```
plt.plot(recalls_forest, precisions_forest, "b-", linewidth=2,
         label="Random Forest")
plt.plot(recalls, precisions, "--", linewidth=2, label="SGD")
[...]  # beautify the figure: add labels, grid, and legend
plt.show()
```

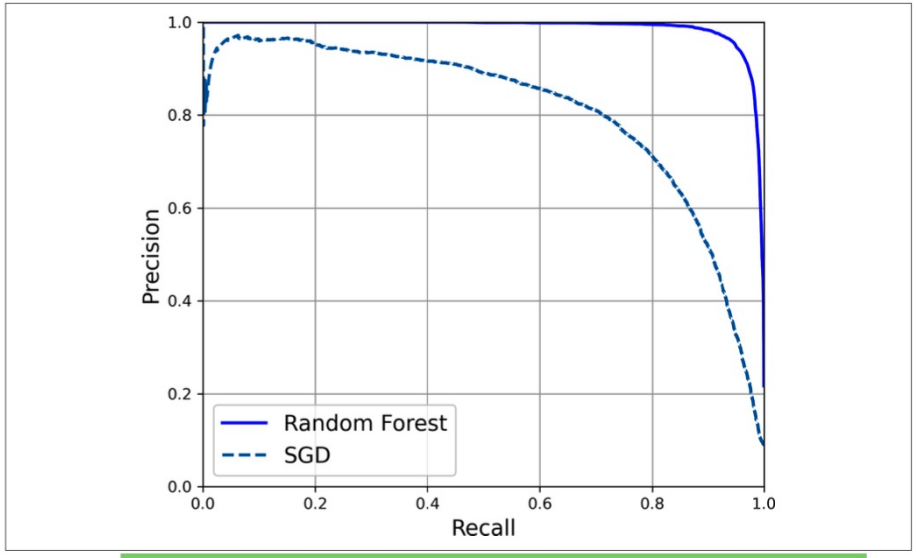


Figure 3-8. Comparing PR curves: the random forest classifier is superior to the SGD classifier because its PR curve is much closer to the top-right corner, and it has a greater AUC

MULTICLASS CLASSIFICATION

- تا الان فهمیدیم چطوری میتونیم binary classification انجام بدیم، یعنی بگیم یه نمونه متعلق به یک کلاس هست یا نه. حالا میخوایم بگیم یه نمونه متعلق به کدوم کلاس هست. یک سری مدل هستن که ذاتا بیس multiclass دان مثل random forest classifier اما یک سری مدل مثل sgd ذات binary classification دارن. میتونیم از همین binary classification ها استفاده کنیم و برای هر کلاس یه مدل train کنیم و هر مدلی که بیشترین احتمال رو خروجی داد به اون کلاس تعلق بدیم نمونه رو. به این روش کار میگن (OvR) one - versus-the-rest.
- یه استراتژی دیگه اینه مدل هایی مثل 0s and 2s تشکیل بدیم اونجوری چون ۹ تا کلاس داریم و باید به ازای هر دو کلاس یه مدل ترین کنیم این شکلی ۴۵ تا کلاس برای این مثال داریم و هر کلاسی که بیشترین دوئل هارو ببره اون انتخاب میشه به این کار one-versus-one میگن.
- خوبی scikit-learn اینه بسته به الگوریتم و ورودی هامون انتخاب میکنه از دو تا استراتژی بالا تا کدوم رو انجام بده.
- بیایم فعلا یه مدل به اسم svm بزنیم روی دیتامون با ۲۰۰۰ تا عکس بزنیم

```
svm_clf = SVC(random_state=42)
svm_clf.fit(X_train[:2000], y_train[:2000])  # y_train, not y_train_5
```

That was easy! We trained the SVC using the original target classes from 0 to 9 (y_train), instead of the 5-versus-the-rest target classes (y_train_5). Since there are

```
>>> svm_clf.predict([some_digit])
array(['5'], dtype=object)
```

ERROR ANALYSIS

- برای اینکه بتونیم مدلمون رو بهتر کنیم از جمله کارهایی که میتونیم انجام بدیم اینه که ببینم رو چه دیتاهایی داره اشتباه میکنه و تحلیل کنیم. ماتریس confusion روش خوبی برای تحلیل هستش. میتونیم الان روی ۱۰ تا کلاسمون و پردیکت های یه مدل ماتریس رو رسم کنیم:

```
from sklearn.metrics import ConfusionMatrixDisplay

y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred)
plt.show()
```

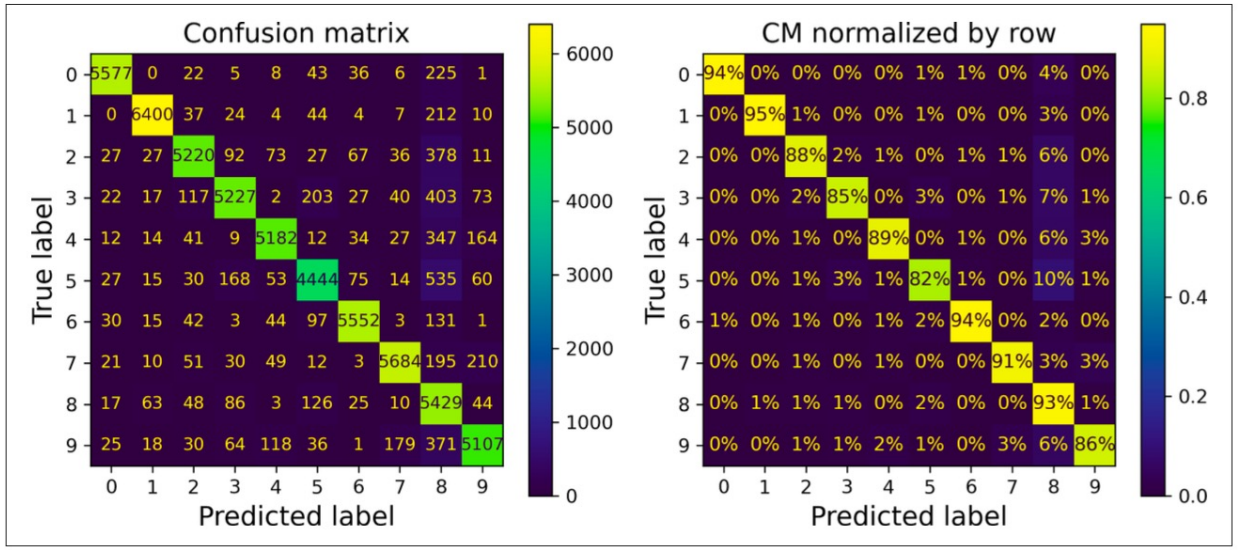


Figure 3-9. Confusion matrix (left) and the same CM normalized by row (right)

ماتریس سمت چپ نشون میده تقریبا اوضاع خوبه و اکثرا مواردی که پیش‌بینی کرده مدل با مقدار دقیقش یکیه. اگه ببینیم ۵ مقدار کمتری درست داره که یه دلیلش میتونه این باشه که مقدار ۵های کمتری داخل training set بوده. برای پیشگیری از همچین مشکل هایی میایم normalize میکنیم تا ماتریس سمپ راست ساخته بشه.از روی ماتریس میفهمیم نزدیک به ۱۰ درصد مواردی که ۵ شناخته شده توسط مدل در واقع ۸ بودن و این درصد بالاییه در حالی که فقط ۲ درصد مواردی که ۸ شناخته شدن ۵ بوده.

CONTINUE

- طبق آنالیزی که داشتیم باید اکثر تلاشمون رو روی عدد ۸ بذاریم، یا بریم دیتا بیشتر از ۸ بیاریم یا اینکه یه الگوریتمی پیدا کنیم برای اینکه ۸ رو بهتر بشناسه.
- به جز این موارد میتونیم ۲ کلاس رو ۲ به ۲ تحلیل کنیم ، مثلا ۳ و ۵ رو میتونیم ببینیم چه طور دیتاهایی بودن که درست یا اشتباه پیش‌بینی شدن.

```
cl_a, cl_b = '3', '5'
X_aa = X_train[(y_train == cl_a) & (y_train_pred == cl_a)]
X_ab = X_train[(y_train == cl_a) & (y_train_pred == cl_b)]
X_ba = X_train[(y_train == cl_b) & (y_train_pred == cl_a)]
X_bb = X_train[(y_train == cl_b) & (y_train_pred == cl_b)]
[...]  # plot all images in X_aa, X_ab, X_ba, X_bb in a confusion matrix style
```

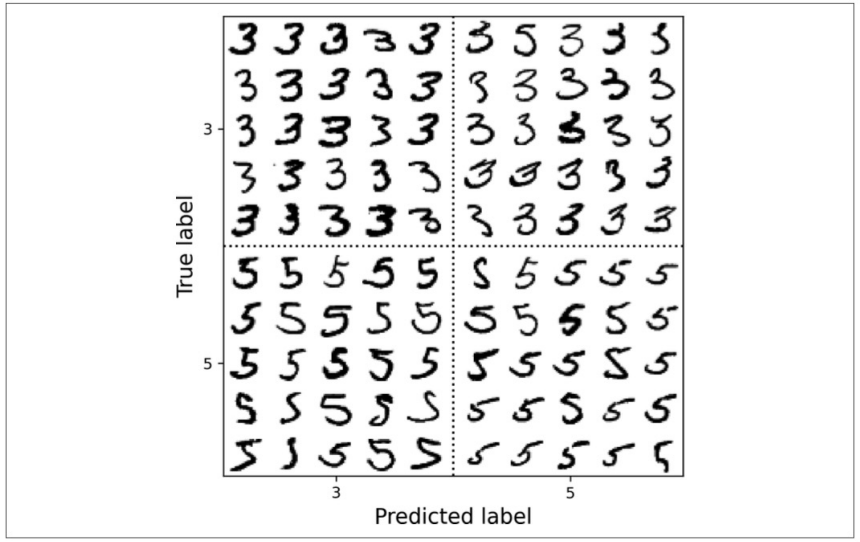


Figure 3-11. Some images of 3s and 5s organized like a confusion matrix

وقتی خطاهارو میبینیم بعضیاشون حتی مارو هم به خطا میندازن ولی نکته ای که هست مغز ما با یه مدل کلی تفاوت داره مدل کارش اینه بیاد به هر پیکسل یه ضریب بده و جمع کنه تا بتونه تشخیص بده، برای همین اگه جای ۳ و ۵ یه ذره rotate بشه یا شیفت پیدا کنه راست و چپ ممکنه اشتباه کنه و چون ۳ و ۵ صرفا داخل یه سری پیکسل محدود با هم اختلاف دارن احتمال اشتباه بالاس نسبتا.

MULTILABEL CLASSIFICATION

- تو بعضی از موارد نیاز داریم یه سیستمی داشته باشیم تا خروجی اون شامل این باشه که بگه یه نمونه متعلق به چه کلاس هایی هستش. برای مثال یه عدد هم میتونه بزرگتر از ۷ باشه هم میتونه فرد باشه.
- میتونیم از مدل Kneightbors استفاده کنیم.

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

y_train_large = (y_train >= '7')
y_train_odd = (y_train.astype('int8') % 2 == 1)
y_multilabel = np.c_[y_train_large, y_train_odd]
```

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_multilabel)
```

```
>>> knn_clf.predict([some_digit])
array([[False,  True]])
```

And it gets it right! The digit 5 is indeed not large (False) and odd (True).

- اگه بخوایم مدل های مختلف رو اینجا ارزیابی کنیم میتونیم از همون f1 استفاده کنیم:

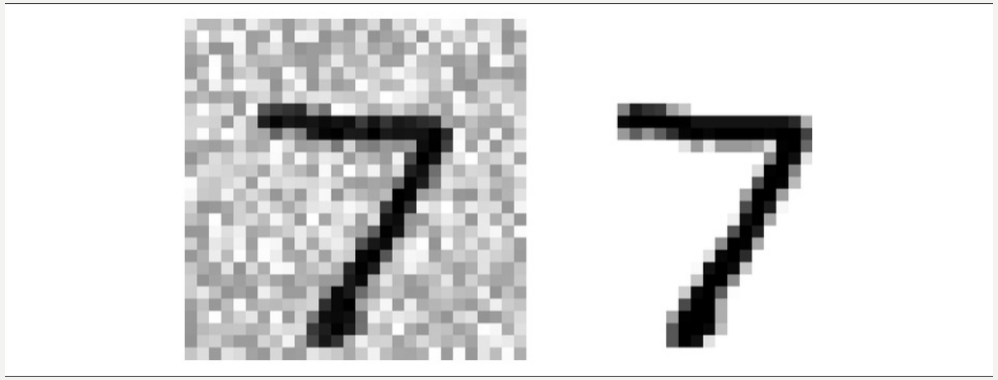
```
>>> y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3)
>>> f1_score(y_multilabel, y_train_knn_pred, average="macro")
0.976410265560605
```

MULTIOUTPUT CLASSIFICATION

- در واقع یک کیس از نسخه عمومی تر multilabel classification هستش. برای مثال اگه یه مدل باشه که نویز داخل عکسامونو پاک کنه باید به ازای هر پیکس یک لیبل و به ازای هر لیبل یک مقدار بین ۰ تا ۲۵۵ داریم.
- بیایم اول یه سری نویز بندازیم رو عکسامون:

```
np.random.seed(42)  # to make this code example reproducible
noise = np.random.randint(0, 100, (len(X_train), 784))
X_train_mod = X_train + noise
noise = np.random.randint(0, 100, (len(X_test), 784))
X_test_mod = X_test + noise
```

```
y_train_mod = X_train
y_test_mod = X_test
```



- عکسامون این شکلی میشه. حالا میتونیم یه مدل براش train کنیم.

```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train_mod, y_train_mod)
clean_digit = knn_clf.predict([X_test_mod[0]])
plot_digit(clean_digit)
plt.show()
```

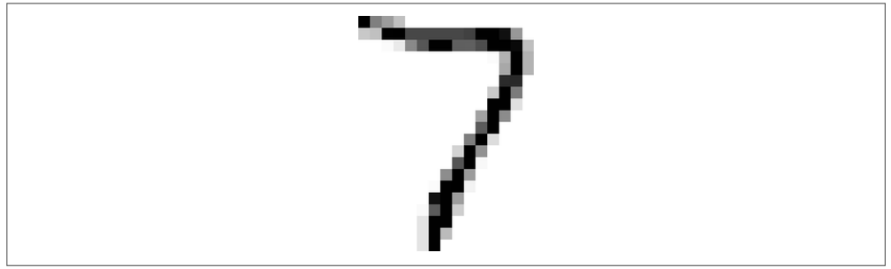


Figure 3-13. The cleaned-up image

END

Multiclass Algorithms

A Multiclass algorithm is a type of machine learning technique designed to solve ML tasks that involve classifying instances into classifying instances into more than two classes or categories. Some algorithms used for multiclass classification include Logistic Regression, Support Vector Machine, Random Forest, KNN and Naive Bayes.

Multioutput Algorithms

Multioutput algorithms are a type of machine learning approach designed for problems where th output consists of multiple variables, and each variable can belong to a different class or have a different range of values. In other words, multioutput problems involve predicting multiple dependent variables simultaneously.