

## Fəsil 3. Təsnifat (Klassifikasiya)

Fəsil 1'də qeyd etmişdik ki, ən geniş yayılmış nəzarətli öyrənmə məsələləri regressiya (dəyərlərin proqnozlaşdırılması) və təsnifatdır (siniflərin proqnozlaşdırılması). Fəsil 2'də biz mənzil dəyərlərinin proqnozlaşdırılması kimi bir regressiya məsələsini araşdırıq və bu zaman xətti regressiya, qərar ağacıları və təsadüfi meşələr (random forests) kimi müxtəlif alqoritmlərdən istifadə etdik (bunlar haqqında sonrakı fəsillərdə daha ətraflı izah ediləcək). İndi isə diqqətimizi təsnifat sistemlərinə yönəldəcəyik.

# MNIST

Bu fəsildə biz MNIST məlumat toplusundan istifadə edəcəyik. Bu, ABŞ Siyahıyalma Bürosunun əməkdaşları və orta məktəb şagirdləri tərəfindən əl ilə yazılmış 70,000 kiçik rəqəm təsvirindən ibarət bir toplusudur. Hər bir təsvir, təmsil etdiyi rəqəmlə etiketlənib. Bu toplu o qədər çox tədqiq edilib ki, onu tez-tez maşın öyrənməsinin "salam, dünya"sı ("hello world") adlandırırlar: insanlar hər dəfə yeni bir təsnifat alqoritmi hazırlayanda, onunla MNIST üzərində necə bir nəticə əldə edəcəklərini aşadırlar və maşın öyrənməsini öyrənən hər kəs gec-tez bu məlumat toplusu ilə qarşılaşır.

Scikit-Learn məşhur məlumat toplularını yüklemək üçün bir çox köməkçi funksiyani içərisində cəmləyir, hansı ki onlardan biri də MNIST'dir. Aşağıdakı kod MNIST məlumat toplusunu OpenML.org saytından yükleyir:

```
from sklearn.datasets import fetch_openml  
  
mnist = fetch_openml('mnist_784', as_frame=False)
```

`sklearn.datasets` paketinə əsas üç növ funksiya daxildir:

- `fetch_openml()` kimi `fetch_*` funksiyaları – real həyatdan olan məlumat toplularını internetdən yüklemək üçün;
- `load_*` funksiyaları – Scikit-Learn ilə birləşdə paketlənmiş kiçik "oyuncaq" məlumat toplularını yüklemək üçün (beləliklə, onların internetdən endirilməsinə ehtiyac qalmır);
- və `make_*` funksiyaları – testlər üçün faydalı olan saxta məlumat topları yaratmaq üçün istifadə edilir.

Yaradılmış məlumat topları adətən həm giriş məlumatlarını (`X`), həm də hədəfləri (`y`) NumPy massivləri şəklində saxlayan (`X, y`) korteji (`tuple`) olaraq qaytarılır. Digər məlumat topları isə `sklearn.utils.Bunch` obyektləri kimi qaytarılır; bunlar lügətlərdir (dictionaries) və onların elementlərinə həm də atribut kimi

müraciət etmək mümkündür. Onlar adətən aşağıdakı elementləri saxlayır:

"DESCR"

Məlumat toplusunun təsviri.

"data"

Giriş məlumatları; adətən 2-ölçülü NumPy massivi şəklində.

"target"

Etiketlər (hədəflər); adətən 1-ölçülü NumPy massivi şəklində.

`fetch_openml()` funksiyası bir qədər qeyri-adidir, çünki standart olaraq o, giriş məlumatlarını Pandas DataFrame (məlumat çərçivəsi) və etiketləri isə Pandas Series (sırası) kimi qaytarır (məlumat toplusunun seyrədilmiş olduğu hallar istisna olmaqla).

Lakin MNIST məlumat toplusu təsvirlərdən ibarətdir və DataFrame-lər bunun üçün o qədər də əlverişli deyil. Buna görə də, məlumatları NumPy massivləri kimi əldə etmək üçün `as_frame=False` təyin etmək daha məqsədə uyğundur.

Gəlin bu massivlərə nəzər salaq:

```
>>> 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,)
```

Burada 70,000 təsvir var və hər təsvirin 784 əlaməti mövcuddur. Bunun səbəbi hər bir təsvirin  $28 \times 28$  piksel ölçüsündə olmasıdır və hər bir xüsusiyyət sadəcə bir pikselin 0-dan (ağ) 255-ə (qara) qədər olan intensivliyini ifadə edir. Gəlin məlumat toplusundan bir rəqəmə nəzər yetirək ([Şəkil 3-1](#)). Bunun üçün bizə lazım olan tək şey bir nümunənin xüsusiyyət vektorunu götürmək, onu  $28 \times 28$  ölçülü massivə çevirib (reshape), Matplotlib-in `imshow()` funksiyasının köməyi ilə görüntüləməkdir. 0-in ağ, 255-in isə qara olduğu boz çalarlı (grayscale) rəng xəritəsini əldə etmək üçün isə `cmap="binary"` parametrindən istifadə edirik:

```
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. MNIST Şəkil Nümunəsi*

Bu, 5-ə oxşayır və həqiqətən də etiket bizə bunu deyir:

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

Təsnifat məsələsinin mürəkkəbliyi haqqında sizə təəssürat yaratmaq üçün **Şəkil 3-2** MNIST məlumat toplusundan daha bir neçə təsviri nümayiş etdirir. Fəqət məlumatları yaxından yoxlamazdan əvvəl hər zaman bir test toplusu yaratmalı və onu kənara qoymalıyıq. **fetch\_openml()** tərəfindən qaytarılan MNIST

məlumat toplusu əslində artıq təlim toplusuna (ilki 60,000 təsvir) və test toplusuna (son 10,000 təsvir) bölünmüştür:

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

Təlim toplusu bizim üçün artıq qarışdırılıb ki, bu da yaxşı haldır. Çünkü bu, bütün çarbaz yoxlama (cross-validation) qruplarının oxşar olacağına zəmanət verir (biz bir qrupda hansısa rəqəmlərin əskik olmasını istəmirik). Bundan əlavə, bəzi öyrənmə alqoritmləri təlim nümunələrinin ardıcılığına qarşı həssasdır və onlar ardıcıl olaraq çoxlu oxşar nümunə ilə qarşılaşdıqda zəif nəticə göstərirlər. Məlumat toplusunun qarışdırılması bunun baş verməyəcəyini təmin edir.



Şəkil 3-2. MNIST məlumat toplusundan rəqəmlər

# Training a Binary Classifier

Gəlin hələlik məsələni sadələşdirək və yalnız bir rəqəmi – məsələn, 5 rəqəmini müəyyən etməyə çalışaq. Bu "5-detektoru" cəmi iki sinfi, yəni 5 və qeyri-5 (5-olmayan) siniflərini bir-birindən ayırd edə bilən ikili təsnifatçının (binary classifier) bir nümunəsi olacaq.

Əvvəlcə bu təsnifat məsələsi üçün hədəf vektorlarını yaradaq:

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

İndi də bir təsnifatçı seçib, onu təlim edək. Başlamaq üçün yaxşı bir yer Scikit-Learn-ün **SGDClassifier** sinfindən *Stoxastik Qradiyent Enişi* (SQE və ya stoxastik QE) təsnifatçısıdır. Bu təsnifatçı çox böyük məlumat toplularını səmərəli şəkildə emal etmək qabiliyyətinə malikdir. Bu, qismən ona görədir ki, SQE təlim nümunələrini hər dəfə birini ayrı-ayrı emal edir, bu da SQE-ni sonradan görəcəyiniz kimi onlayn öyrənmə üçün çox əlverişli edir. Gəlin bir **SGDClassifier** yaradaq və onu bütün təlim toplusu üzərində təlim edək:

```
from sklearn.linear_model import SGDClassifier  
  
sgd_clf = SGDClassifier(random_state=42)  
sgd_clf.fit(X_train, y_train_5)
```

İndi də bunu 5 rəqəmi şəkillərini tapması üçün istifadə edə bilərik:

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

Təsnifatçı bu təsvirin 5-i təmsil etdiyini təxmin edir (Doğru). Görünür, bu xüsusi halda düzgün təxmin etdi! İndi gəlin bu modelin performansını (və ya *fəaliyyət göstəricisini*) qiymətləndirək.

## Fəaliyyət Göstəriciləri

Bir təsnifatçını qiymətləndirmək çox vaxt bir regressorу qiymətləndirməkdən xeyli mürəkkəbdır, buna görə də bu fəslin böyük bir hissəsinə bu mövzuya həsr edəcəyik. Mövcud olan bir çox performans ölçüsü (səmərəlilik göstəricisi) var, odur ki, daha bir qəhvə götürün və bir yığın yeni anlayış və abreviatur öyrənməyə hazırlanın!

# Çarpaz Yoxlama ilə Dəqiqliyin Ölçülməsi

Bir modeli qiymətləndirməyin yaxşı yolu, eynilə **Fəsil 2**'də etdiyiniz kimi, çarpaz yoxlamadan (cross-validation) istifadə etməkdir. Gəlin SGDClassifier modelimizi qiymətləndirmək üçün k-qatlı çarpaz yoxlama (k-fold cross-validation) texnikasını 3 qat ilə tətbiq edərək `cross_val_score()` funksiyasını istifadə edək.

Xatırladaq ki, k-qatlı çarpaz yoxlama təlim toplusunu k qata (bizim halda üç) bölmək, sonra modeli k dəfə təlim etmək deməkdir; hər dəfə fərqli bir qat qiymətləndirmə üçün kənarda saxlanılır (bax: **Fəsil 2**):

```
>>> 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])
```

Bütün çarpaz yoxlama qatlarında 95%-dən yuxarı dəqiqlik (düzgün proqnozların nisbəti)? Bu, heyrətamız görünür, elə deyilmi? Yaxşı, çox həyəcanlanmazdan əvvəl, gəlin hər bir təsviri sadəcə ən çox rast gələn sinfə (bizim halda bu, mənfi sinifdir, yəni "5-olmayan") təsnif edən bəsit bir təsnifatçıya (*dummy classifier*) nəzər salaq:

```
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
```

İndi də modelimizin dəqiqliyini müəyyən edək:

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

Düzdür, onun dəqiqliyi 90%-dən çoxdur! Bu, sadəcə olaraq ona görədir ki, təsvirlərin yalnız təxminən 10%-i 5-dir, buna görə də əgər siz həzaman bir təsvirin 5 *olmadığını* təxmin etsəniz, təxminən 90% hallarda haqlı çıxacaqsınız. Nostradamusa bu cür üstün gəlir.

Bu, nə üçün dəqiqliyin, xüsusən də qeyri-balanslı məlumat topluları (yəni, bəzi siniflərin digərlərindən qat-qat daha tez-tez rast gəlindiyi hallar) ilə işləyərkən, təsnifatçılar üçün ümumiyyətlə üstünlük verilən performans ölçüsü olmadığını nümayiş etdirir.

Bir təsnifatçının performansını qiymətləndirməyin daha yaxşı bir yolu *xəta matrisinə* (Confusion Matrix - CM) baxmaqdır.

## ÇARPAZ YOXLAMANIN TƏTBİQİ

Bəzən çarraz yoxlama prosesi üzərində Scikit-Learn-ün hazır şəkildə təqdim etdiyindən daha çox nəzarətə ehtiyacınız olur. Bu hallarda, çarraz yoxlamanı özünüz tətbiq edə bilərsiniz. Aşağıdakı kod Scikit-Learn-ün `cross_val_score()` funksiyası ilə təxminən eyni işi görür və eyni nəticəni çap edir:

```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits=3) # add shuffle=True if the dataset is
# not already shuffled
for train_index, test_index in skfolds.split(X_train, y_train_5):
    clone_clf = clone(sgd_clf)
    X_train_folds = X_train[train_index]
    y_train_folds = y_train_5[train_index]
    X_test_fold = X_train[test_index]
    y_test_fold = y_train_5[test_index]

    clone_clf.fit(X_train_folds, y_train_folds)
    y_pred = clone_clf.predict(X_test_fold)
    n_correct = sum(y_pred == y_test_fold)
    print(n_correct / len(y_pred)) # prints 0.95035, 0.96035, and 0.9604
```

`StratifiedKFold` sinfi, ([Fəsil 2](#)-də izah edildiyi kimi) hər bir sinfin təmsiledici nisbetini ehtiva edən qatlar (folds) yaratmaq üçün təbəqəli nümunə götürmə (stratified sampling) həyata keçirir. Hər iterasiyada kod təsnifatçının bir klonunu yaradır, həmin klonu təlim qatları üzərində təlim edir və test qatı üzərində proqnozlar verir. Sonra düzgün proqnozların sayını hesablayır və düzgün proqnozların nisbətini çıxışa verir.

## Xəta Matrisi

Xəta matrisinin (confusion matrix) əsas ideyası, bütün A/B cütlükleri üçün A sinfinin nümunələrinin neçə dəfə B sinfi olaraq təsnif edildiyini saymaqdır. Məsələn, təsnifatçının 8 rəqəminin təsvirlərini neçə dəfə 0 ilə səhv saldığını bilmək üçün xəta matrisinin 8-ci sətrinə və 0-ci sütununa baxmaq lazımdır.

Xəta matrisini hesablamaq üçün əvvəlcə bir proqnozlar toplusuna ehtiyacınız var ki, onları faktiki hədəflər ilə müqayisə etmək mümkün olsun. Proqnozları test toplusu üzərində edə bilərsiniz, lakin hələlik ona toxunmamaq daha yaxşıdır (unutmayın ki, test toplusundan yalnız layihənizin ən sonunda, artıq istifadəyə verməyə hazır olduğunuz bir təsnifatçıınız olduqda istifadə etmək istəyirsiniz). Bunun əvəzinə `cross_val_predict()` funksiyasından istifadə edə bilərsiniz:

```
from sklearn.model_selection import cross_val_predict  
  
y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
```

Eynilə `cross_val_score()` funksiyası kimi, `cross_val_predict()` də k-qatlı çarpez yoxlama (k-fold cross-validation) həyata keçirir, lakin qiymətləndirmə xallarını (evaluation scores) qaytarmaq əvəzinə, hər bir test qatı üzərində edilən proqnozları qaytarır. Bu o deməkdir ki, siz təlim toplusundakı hər bir nümunə üçün "təmiz" bir proqnoz əldə edirsiniz (Burada "təmiz" ilə "nümunədən kənar" (out-of-sample) nəzərdə tuturam: yəni model, təlim zamanı heç vaxt görmədiyi məlumatlar üzərində proqnozlar verir).

İndi siz `confusion_matrix()` funksiyasından istifadə edərək xəta matrisini almağa hazırlısınız. Sadəcə olaraq ona hədəf sinifləri (`y_train_5`) və proqnozlaşdırılan sinifləri (`y_train_pred`) daxil edirik:

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

Xəta matrisində (confusion matrix) hər bir sətir faktiki sinfi (actual class), hər bir sütun isə proqnozlaşdırılan sinfi (predicted class) təmsil edir. Bu matrisin birinci sətri "5-olmayan" təsvirlərə (mənfi sinif) baxır: onlardan 53,892-si düzgün şəkildə "5-olmayan" kimi təsnif edilib (bunlar həqiqi mənfilər – *true negatives* adlanır), qalan 687-si isə səhvən 5 kimi təsnif edilib (yalançı müsbətlər – *false positives*, həmçinin I növ xətalar da deyilir).

İkinci sətir 5 rəqəminin təsvirlərinə (müsəbət sinif) baxır: onlardan 1,891-i səhvən "5-olmayan" kimi təsnif edilib (yalançı mənfilər – *false negatives*, həmçinin II növ xətalar da deyilir), qalan 3,530-u isə düzgün şəkildə 5 kimi təsnif edilib (həqiqi müsbətlər – *true positives*).

Mükəmməl bir təsnifatçının yalnız həqiqi müsbətləri və həqiqi mənfiləri olardı, buna görə də onun xəta matrisində sıfırdan fərqli dəyərlər yalnız əsas diaqonal (yuxarı soldan aşağı sağa) üzərində yerləşərdi:

```
>>> y_train_perfect_predictions = y_train_5 # pretend we reached perfection
>>> confusion_matrix(y_train_5, y_train_perfect_predictions)
array([[54579,  0],
       [ 0,  5421]])
```

Xəta matrisi (confusion matrix) bizə xeyli məlumat verir, lakin bəzən daha yiğcam bir ölçüyə (metrikaya) üstünlük verə bilərik. Baxmaq üçün maraqlı olan göstəricilərdən biri müsbət proqnozların dəqiqliyidir (precision). (**Tənlik 3-1**).

*Tənlik 3-1. Dəqiqlik (Precision)*  

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

**TP həqiqi müsbətlərin** (true positives) sayı, FP isə **yalançı müsbətlərin** (false positives) sayıdır.

Mükəmməl dəqiqlik(precision) əldə etməyin bəsir (trivial) bir yolu, ən əmin olduğu bir nümunə üzrə yeganə müsbət proqnoz istisna olmaqla, hər zaman mənfi proqnozlar verən bir təsnifatçı yaratmaqdır. Əgər bu bir proqnoz düzgündürse, o zaman təsnifatçının presizyonu 100% olur ( $dəqiqlik = 1/1 = 100\%$ ).

Aydındır ki, belə bir təsnifatçı o qədər də faydalı olmazdı, çünki bir müsbət nümunə xaricində qalanların hamısına məhəl qoymayacaqdı. Buna görə də, dəqiqlik adətən recall (əhatəlilik - axtarılan informasiyanın nə qədərinin tapıldığı) adlanan başqa bir ölçü ilə birlikdə istifadə olunur, buna həmçinin həssaslıq (sensitivity) və ya həqiqi müsbət nisbəti (True Positive Rate - TPR) də deyilir: bu, təsnifatçı tərəfindən düzgün aşkarlanan müsbət nümunələrin nisbətidir (**Tənlik 3-2**).

### *Tənlik 3-2. Əhatəlilik (Recall)*

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

FN əlbəttə ki, **yalançı mənfilərin** (false negatives) sayıdır.

Əgər xəta matrisi ilə bağlı fikrinizdə qaranlıqlar varsa, **Şəkil 3-3**'ə nəzər salın, qaranlıq yoxdursa da salın, çünki xeyirlidir.

		Predicted		
		Negative	Positive	
Actual	Negative	8	3	9
	Positive	7	2	6
		5	5	5
		Precision (e.g., 3 out of 4)		
		Recall (e.g., 3 out of 5)		
		TN		FP
		FN		TP

Şəkil 3-3. Həqiqi mənfilərin (yuxarı sol), yalançı müsbətlərin (yuxarı sağ), yalançı mənfilərin (aşağı sol) və həqiqi müsbətlərin (aşağı sağ) nümunələrini göstərən təsvirli bir xəta matrisi.

## Dəqiqlik və Əhatəlilik

Scikit-Learn təsnifatçı ölçülərini (metrikalarını) hesablamaq üçün bir neçə funksiya təqdim edir, bura dəqiqlik və əhatəlilik (recall) də daxildir:

```
>>> 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
```

İndi bizim "5-detektorumuz" onun dəqiqliyinə (accuracy) baxduğımız zamankı qədər cəlbedici görünmür. O, bir təsvirin 5 olduğunu iddia etdiqdə, yalnız 83.7% hallarda düzgün olur. Üstəlik, o, mövcud 5-lərin yalnız 65.1%-ni aşkarlayır (tapa bilir).

Xüsusilə iki təsnifatçının müqayisə etmək üçün vahid bir ölçüyə (metrikaya) ehtiyacınız olduqda, dəqiqlik və əhatəliliyi (recall) F\_1 xalı (F\_1 score) adlanan tək bir ölçüdə birləşdirmək çox vaxt əlverişli olur. F\_1 xalı dəqiqlik və əhatəliliyin (recall) harmonik ortasıdır (**Tənlik 3-3**).

Sadə orta (ədədi orta) bütün dəyərlərə bərabər yanaşlığı halda, harmonik orta aşağı dəyərlərə daha çox ağırlıq verir. Nəticə etibarilə, təsnifatçı yalnız o halda yüksək F\_1 xalı alacaq ki, həm əhatəliliyi (recall), həm də dəqiqliyi yüksək olsun.

Tənlik 3-3. F\_1 xalı

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

F\_1 xalını hesablamaq üçün `f1_score()` funksiyasını çağırmaq kifayətdir.

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

F\_1 xalı (F\_1 score) oxşar dəqiqlik və əhatəliliyə (recall) malik olan təsnifatçılara üstünlük verir. Bu, hər zaman istədiyiniz şey deyil: bəzi kontekstlərdə sizi əsasən dəqiqlik maraqlandırır, digər kontekstlərdə isə həqiqətən də əhatəlilik (recall) sizin üçün daha vacibdir.

Məsələn, əgər siz uşaqlar üçün təhlükəsiz olan videoları aşkar etmək üçün bir təsnifatçı təlim etsəniz, çox güman ki, bir çox yaxşı videonu rədd edən (aşağı əhatəlilik/recall), lakin yalnız təhlükəsiz olanları saxlayan (yüksek dəqiqlik) bir təsnifatçıya üstünlük verərsiniz. Nəinki, daha yüksək əhatəliliyi (recall) olan, lakin məhsulunuzda bir neçə həqiqətən pis videonun görünməsinə imkan verən bir təsnifatçıya (belə hallarda, siz hətta təsnifatçının video seçimində nəzarət etmək üçün insan müdaxiləsi xətti əlavə etmək istəyə bilərsiniz).

Digər tərəfdən, fərz edək ki, siz müşahidə kameralarının təsvirlərində mağaza oğrularını aşkar etmək üçün bir təsnifatçı təlim edirsiniz: əgər təsnifatçınızın 99% əhatəliliyi (recall) varsa, onun cəmi 30% dəqiqliyə malik olması yəqin ki, problem deyil (əlbəttə, təhlükəsizlik işçiləri bir neçə yalançı həyəcan siqnalı alacaqlar, lakin demək olar ki, bütün mağaza oğruları yaxalanacaq).

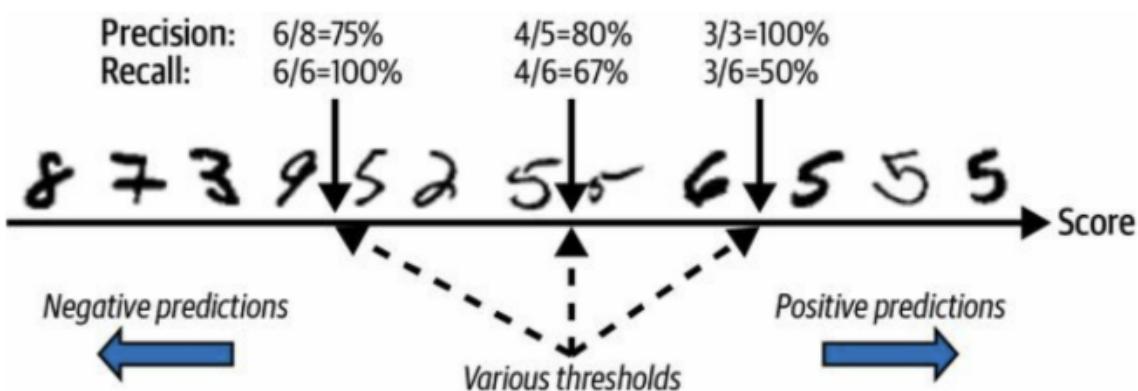
Təəssüf ki, siz hər ikisinə eyni anda sahib ola bilməzsınız: dəqiqliyi artırmaq əhatəliliyi (recall) azaldır və əksinə. Buna dəqiqlik/əhatəlilik (recall) kompromisi (*və ya qarşılıqlı güzəştü*) deyilir.

## Dəqiqlik/ Əhatəlilik Qarşılıqlı Güzəsti (Trade-off)

Bu kompromisi (karşılıqlı güzəsti) başa düşmək üçün gəlin **SGDClassifier**-in öz təsnifat qərarlarını necə verdiyinə baxaq. Hər bir nümunə üçün o, bir qərar funksiyası (decision function) əsasında bir xal (score) hesablayır. Əgər həmin xal müəyyən bir həddən (threshold) böyükdürsə, o, nümunəni müsbət sinfə təyin edir; əks halda onu mənfi sinfə təyin edir.

**Şəkil 3-4** solda ən aşağı xaldan sağda ən yüksək xala doğru sıralanmış bir neçə rəqəmi göstərir. Fərz edək ki, qərar həddi mərkəzi oxun üzərində (iki 5-liyin arasında) yerləşdirilib: bu həddin sağında siz 4 həqiqi müsbət (faktiki 5-lər) və 1 yalançı müsbət (əslində 6 olan) tapacaqsınız. Buna görə də, bu hədlə dəqiqlik 80% (5-dən 4-ü) təşkil edir. Lakin 6 faktiki 5-dən təsnifatçı yalnız 4-ünü aşkarlayır, beləliklə əhatəlilik (recall) 67% (6-dan 4-ü) olur.

Əgər siz həddi qaldırsanız (onu sajdakı oxa çəksəniz), yalançı müsbət (həmin 6) həqiqi mənfiyə çevrilir və bununla da dəqiqliyi artırır (bu halda 100%-ə qədər), lakin bir həqiqi müsbət yalançı mənfiyə çevrilir və əhatəliliyi (recall) 50%-ə qədər azaldır. Əksinə, həddi aşağı salmaq əhatəliliyi (recall) artırır və dəqiqliyi azaldır.



Şəkil 3-4. Dəqiqlik/Əhatəlilik (Recall) kompromisi: təsvirlər öz təsnifatçı xallarına görə sıralanıb və seçilmiş qərar həddindən yuxarıda olanlar müsbət hesab edilir; hədd nə qədər yüksək olarsa, əhatəlilik (recall) o qədər aşağı olar, lakin (ümumilikdə) dəqiqlik o qədər yüksək olar.

Scikit-Learn sizə həddi (threshold) birbaşa təyin etməyə imkan vermir, lakin sizə proqnozlar vermək üçün istifadə etdiyi qərar xallarına (decision scores) çıxış (access) təmin edir.

Təsnifatçının `predict()` metodunu çağırmaq əvəzinə, siz onun `decision_function()` metodunu çağrıra bilərsiniz. Bu metod hər bir nümunə üçün bir xal qaytarır və siz də həmin xallara əsaslanaraq proqnozlar vermək üçün istədiyiniz hər hansı bir həddi istifadə edə bilərsiniz:

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

`SGDClassifier` hədd olaraq 0-a bərabər olan bir dəyər istifadə edir, buna görə də əvvəlki kod `predict()` metodu ilə eyni nəticəni qaytarır (yəni, `True`). Gəlin həddi qaldırıraq:

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

Bu, həddin (threshold) qaldırılmasının əhatəliliyi (recall) azaltdığını təsdiqləyir. Təsvir əslində 5-i təmsil edir və hədd 0 olduqda təsnifatçı onu aşkarlayır, lakin hədd 3,000-ə qaldırıldıqda onu gözdən qaçırır.

Bəs hansı həddi istifadə edəcəyinizə necə qərar verirsınız? Əvvəlcə, təlim toplusundakı bütün nümunələrin xallarını almaq üçün `cross_val_predict()` funksiyasından istifadə edin, lakin bu dəfə proqnozlar əvəzinə qərar xallarını (decision scores) qaytarmasını təyin edin:

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

İndi bu xallarla, bütün mümkün hədlər (thresholds) üçün dəqiqlik və əhatəliliyi (recall) hesablamaq məqsədilə `precision_recall_curve()` funksiyasından istifadə edin (bu funksiya, sonsuz bir həddə qarşılıq gələn, sonuncu dəqiqlik olaraq 0 və sonuncu əhatəlilik (recall) olaraq 1 əlavə edir):

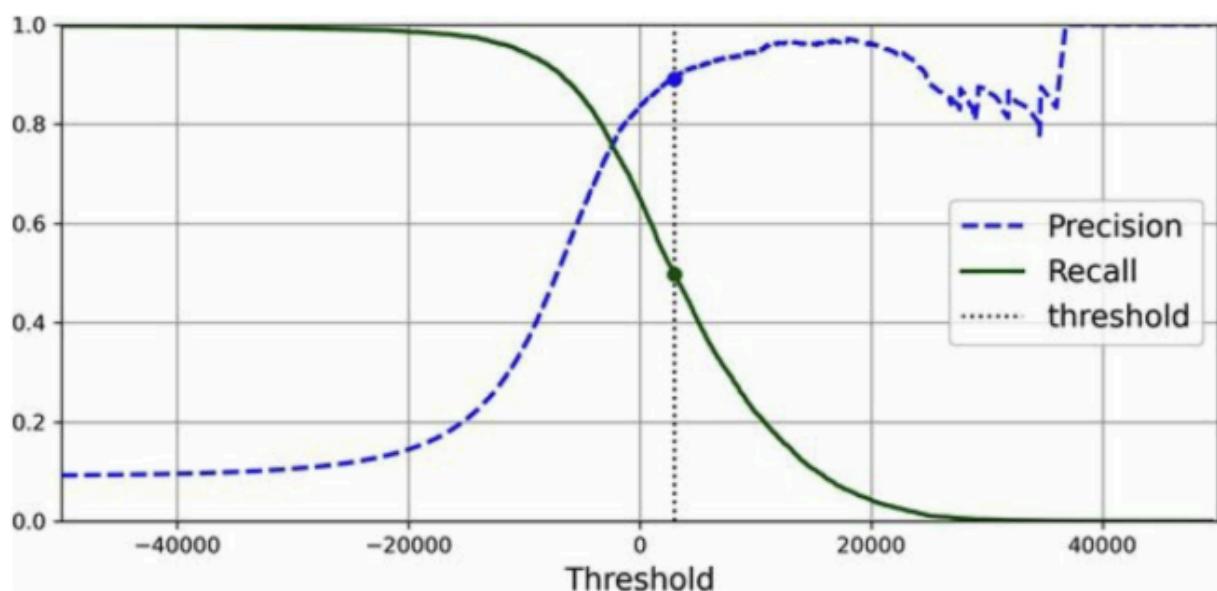
```
from sklearn.metrics import precision_recall_curve

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

Nəhayət, dəqiqlik və əhatəliliyi (recall) hədd dəyərinin (threshold value) funksiyaları kimi qrafikdə qurmaq (plot) üçün Matplotlib-dən istifadə edin ([Şəkil 3-5](#)). Gəlin bizim seçdiyimiz 3,000-lük həddi də göstərək:

```
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()
```



Şəkil 3-5. Dəqiqlik və Əhatəliliyin (Recall) Qərar Həddindən Asılılığı

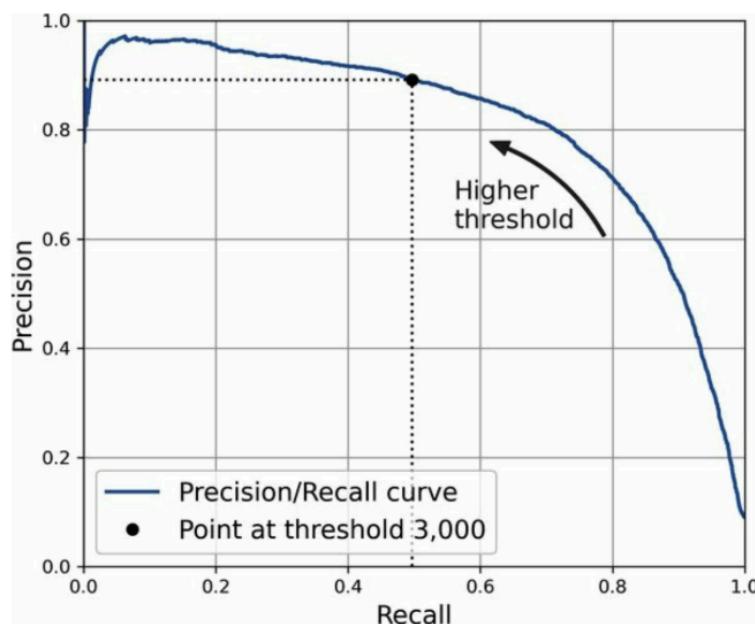
## QEYD

Şəkil 3-5-də dəqiqlik əyrisinin nə üçün əhatəlilik (recall) əyrisindən daha enişli-yoxuşlu (daha qeyri-sabit) olduğu sizə maraqlı gələ bilər. Səbəb odur ki, siz həddi (threshold) qaldırıldığda dəqiqlik bəzən aşağı düşə bilər (baxmayaraq ki, ümumilikdə o, yuxarı qalxacaq). Bunu başa düşmək üçün Şəkil 3-4-ə qayidib baxın və mərkəzi həddən başlayıb onu sadəcə bir rəqəm sağa çəkdikdə nə baş verdiyinə diqqət yetirin: dəqiqlik 4/5-dən (80%) 3/4-ə (75%) enir. Digər tərəfdən, hədd artırıldığda əhatəlilik (recall) yalnız aşağı düşə bilər ki, bu da onun əyrisinin nə üçün hamar görünüşünü izah edir.

Bu hədd dəyərində dəqiqlik 90%-ə yaxındır, əhatəlilik (recall) isə 50% civarındadır.

Yaxşı bir dəqiqlik/əhatəlilik kompromisini seçməyin başqa bir yolu, Şəkil 3-6-da göstərildiyi kimi, dəqiqlik birbaşa əhatəliliyin (recall) qarşılığında qrafikdə qurmaqdır (həmin hədd də göstərilib):

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



Şəkil 3-6. Dəqiqlik və Əhatəlilik Fərqi

Görə bilərsiniz ki, dəqiqlik təxminən 80% əhatəlilik (recall) cıvarında kəskin şəkildə enməyə başlayır. Siz yəqin ki, dəqiqlik/əhatəlilik kompromisini (qarşılıqlı güzəştü) məhz həmin enişdən bir az əvvəl – məsələn, təxminən 60% əhatəlilik (recall) ətrafında seçmək istəyecəksiniz. Amma bu seçim, əlbəttə ki, sizin layihənizdən asılıdır.

Fərz edək ki, siz 90% dəqiqlik hədəfləməyə qərar verdiniz. İstifadə etməli olduğunuz həddi (threshold) tapmaq üçün birinci qrafikdən istifadə edə bilərsiniz, lakin bu, o qədər də dəqiqlik deyil. Alternativ olaraq, sizə ən azı 90% dəqiqlik verən ən aşağı həddi axtara bilərsiniz.

Bunun üçün NumPy massivinin `argmax()` metodundan istifadə edə bilərsiniz. Bu, maksimum dəyərin ilk indeksini qaytarır ki, bu da bizim halda ilk `True` (Doğru) dəyər deməkdir:

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

Proqnozları etmək üçün (hələlik təlim toplusu üzərində) təsnifatçının `predict()` metodunu çağırmaq əvəzinə, bu kodu işə sala bilərsiniz:

```
y_train_pred_90 = (y_scores >= threshold_for_90_precision)
```

Gəlin bu proqnozların dəqiqliyini və əhatəliliyini (recall) yoxlayaq:

```
>>> precision_score(y_train_5, y_train_pred_90)  
0.9000345901072293  
>>> recall_at_90_precision = recall_score(y_train_5, y_train_pred_90)  
>>> recall_at_90_precision  
0.4799852425751706
```

Əla, artıq təsnifatçıımız 90% dəqiqliyə malikdir! Gördüyünüz kimi, istədiyiniz, demək olar ki, hər hansı bir dəqiqliyə malik təsnifatçı yaratmaq kifayət qədər asandır: sadəcə kifayət qədər yüksək bir hədd (threshold) təyin edin, vəssalam.

Amma dayanın, o qədər də tələsməyin – əgər əhatəliliyi (recall) çox aşağıdırsa, yüksək dəqiqliklı təsnifatçı o qədər də faydalı deyil! Bir çox tətbiq üçün 48%-lik əhatəlilik (recall) heç də əla göstərici olmazdı.

## MƏSLƏHƏT

Əgər kimsə "Gəlin 99% **presizyon** əldə edək" desə, siz sorusun:  
"Bəs hansı **əhatəlilikdə** (recall)?"

## ROC Əyrisi

ROC əyrisi (Receiver Operating Characteristic - Qəbululedici Əməliyyat Xarakteristikası) ikili təsnifatçılarda istifadə olunan daha bir geniş yayılmış alətdir. O, dəqiqlik/əhatəlilik (recall) əyrisinə çox oxşayır, lakin dəqiqlik əhatəlilik (recall) qrafiki əvəzinə, ROC əyrisi həqiqi müsbət nisbətini (True Positive Rate - TPR) (əhatəliliyin/recall-in başqa bir adı) yalançı müsbət nisbətinin (False Positive Rate - FPR) qarşısında qurur.

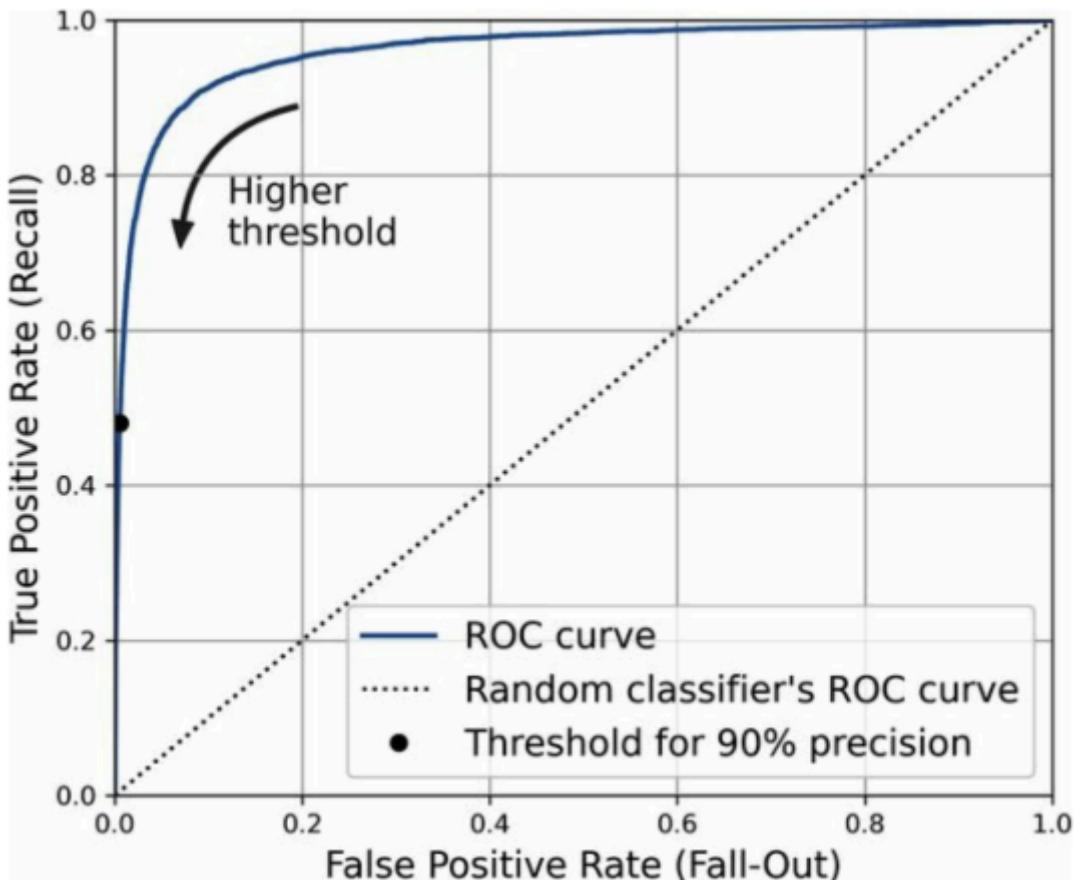
FPR (həmçinin *fall-out* da adlanır) səhvən müsbət kimi təsnif edilən mənfi nümunələrin nisbətidir. Bu,  $1 - \text{həqiqi mənfi nisbətinə}$  (True Negative Rate - TNR) bərabərdir; TNR isə düzgün şəkildə mənfi kimi təsnif edilən mənfi nümunələrin nisbətidir. TNR həmçinin spesifiklik (specificity) də adlanır. Beləliklə, ROC əyrisi həssaslığı (yəni, əhatəliliyi/recall)  $1 - \text{spesifikasiyin}$  qarşısında qurur.

ROC əyrisini qrafikdə qurmaq üçün, siz əvvəlcə müxtəlif hədd dəyərləri (threshold values) üçün TPR və FPR-i hesablamaq məqsədilə `roc_curve()` funksiyasından istifadə edirsiniz:

```
from sklearn.metrics import roc_curve  
  
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

Sonra Matplotlib istifadə edərək FPR-i TPR-in qarşısında qrafikdə qura (plot) bilərsiniz. Aşağıdakı kod **Şəkil 3-7**-dəki qrafiki yaradır. 90% dəqiqliyə uyğun gələn nöqtəni tapmaq üçün biz istədiyimiz həddin (threshold) indeksini axtarmalıyıq. Bu halda hədlər azalan qaydada sıralandığı üçün biz birinci sətirdə  $\geq$  əvəzinə  $\leq$  istifadə edirik:

```
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()
```



Şəkil 3-7. Bütün mümkün hədlər (thresholds) üçün yalançı müsbət nisbətinin (FPR) həqiqi müsbət nisbətinin (TPR) qarşısında qurulduğu ROC əyrisi; qara dairə seçilmiş nisbəti vurğulayır (90% dəqiqlik və 48% əhatəlilik/recall səviyyəsində).

Burada da yenə bir kompromis (qarşılıqlı güzəşt) mövcuddur: əhatəlilik (TPR) nə qədər yüksək olarsa, təsnifatçı bir o qədər çox yalançı müsbət (FPR) yaradır. Nöqtəli xətt tamamilə təsadüfi bir təsnifatçının ROC əyrisini təmsil edir; yaxşı təsnifatçı bu xətdən mümkün qədər uzaqda (yuxarı sol künçə tərəf) qalmalıdır.

Təsnifatçıları müqayisə etməyin bir yolu əyri altındakı sahəni (AUC - Area Under the Curve) ölçməkdir. Mükəmməl təsnifatçının ROC AUC göstəricisi 1-ə bərabər olacaq, tamamilə təsadüfi təsnifatçının ROC AUC göstəricisi isə 0.5-ə bərabər olacaq. Scikit-Learn ROC AUC-u hesablamaq (və ya qiymətləndirmək) üçün bir funksiya təqdim edir:

```
>>> from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.96049385554008616
```

## MƏSLƏHƏT

ROC əyrisi dəqiqlik/əhatəlilik (PR) əyrisinə bu qədər oxşar olduğu üçün hansını istifadə edəcəyinizi necə qərar verəcəyinizi düşünə bilərsiniz. Ümumi bir qayda olaraq, müsbət sinif (positive class) nadir hallarda rast gəlinirse və ya yalançı mənfilərdən (false negatives) daha çox yalançı müsbətlər (false positives) sizin üçün əhəmiyyətlidirse, PR əyrisinə üstünlük vermelisiniz. Əks halda, ROC əyrisindən istifadə edin. Məsələn, əvvəlki ROC əyrisinə (və ROC AUC xalına) baxaraq, təsnifatçının həqiqətən yaxşı olduğunu düşünə bilərsiniz. Lakin bu, əsasən ona görədir ki, mənfilərlə (5-olmayanlar) müqayisədə müsbətlərin (5-lər) sayı azdır. Bunun əksinə olaraq, PR əyrisi təsnifatçının təkmilləşdirilməyə ehtiyacı olduğunu aydın şəkildə göstərir: əyri həqiqətən də yuxarı sağ künçə daha yaxın ola bilərdi (yenidən [Şəkil 3-6](#)-ya baxın).

İndi isə gəlin bir Təsadüfi Meşələr Təsnifatçısı ([RandomForestClassifier](#)) yaradaq və onun PR əyrisini və F\_1 xalını [SGDClassifier](#)-inki ilə müqayisə edək:

```
from sklearn.ensemble import RandomForestClassifier  
  
forest_clf = RandomForestClassifier(random_state=42)
```

[precision\\_recall\\_curve\(\)](#) funksiyası hər bir nümunə üçün etiketləri və xalları gözləyir, buna görə də biz Təsadüfi Meşələr təsnifatçısını təlim etməli və onun hər bir nümunəyə bir xal təyin etməsini təmin etməliyik. Lakin [RandomForestClassifier](#) sinfinin, onun işləmə prinsipinə görə, [decision\\_function\(\)](#) metodu yoxdur (bunu [Fəsil 7](#)-də əhatə edəcəyik). Xoşbəxtlikdən, onun hər bir nümunə üçün sinif ehtimallarını qaytaran [predict\\_proba\(\)](#) metodu var və biz xal olaraq sadəcə müsbət sinfin ehtimalından istifadə edə bilərik ki, bu da problemsız işləyəcək. [RandomForestClassifier](#)-i çarraz yoxlama istifadə edərək təlim etmək və onun hər bir təsvir üçün sinif ehtimallarını proqnozlaşdırmasını təmin etmək üçün [cross\\_val\\_predict\(\)](#) funksiyasını aşağıdakı kimi çağırıa bilərik:

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

Gəlin təlim toplusundakı ilk iki təsvir üçün sinif ehtimallarına baxaq:

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

Model proqnozlaşdırır ki, birinci təsvir 89% ehtimalla müsbətdir, və ikinci təsvirin 99% ehtimalla mənfi olduğunu proqnozlaşdırır. Hər bir təsvir ya müsbət, ya da mənfi olduğundan, hər bir sətirdəki ehtimalların cəmi 100%-ə bərabərdir.

### XƏBƏRDARLIQ

Bunlar faktiki ehtimallar deyil, təxmini ehtimallardır. Məsələn, əger modelin 50% ilə 60% arasında təxmini ehtimalla müsbət kimi təsnif etdiyi bütün təsvirlərə baxsanız, onların təxminən 94%-i həqiqətən də müsbətdir. Beləliklə, bu halda modelin təxmini ehtimalları həddindən artıq aşağı olub — lakin modellər həmçinin həddindən artıq "özündən əmin" də ola bilərlər. `sklearn.calibration` paketi təxmini ehtimalları düzənləmək və onları faktiki ehtimallara daha da yaxınlaşdırmaq üçün alətlərə sahibdir. Daha ətraflı məlumat üçün bu [fəslin iş dəftərindəki](#) əlavə materiallar bölməsinə baxın.

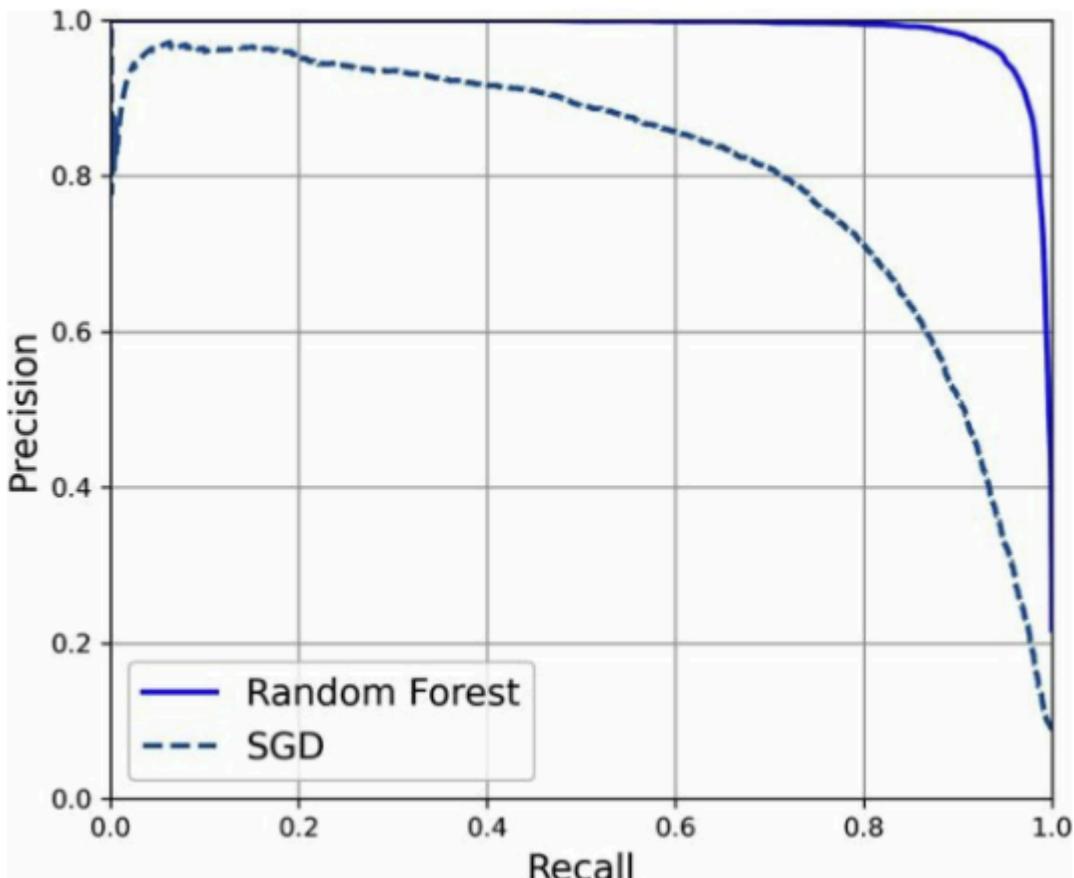
### [fəslin iş dəftəri](#)

İkinci sütun müsbət sinif üçün təxmini ehtimalları saxlayır, buna görə də gəlin onları `precision_recall_curve()` funksiyasına ötürək:

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

İndi PR əyrisini qrafikləməyə hazırlıq. Onların necə müqayisə olunduğunu görmək üçün ilk PR əyrisini də əlavə etmək faydalıdır ([Şəkil 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()
```



Şəkil 3-8. PR əyrilərinin müqayisəsi: Təsadüfi Meşələr təsnifatçısı SGD təsnifatçısından daha üstündür, çünki onun PR əyri yuxarı sağ künçə daha yaxındır və o, daha böyük AUC göstəricisinə malikdir.

Şəkil 3-8-də gördükünüz kimi, `RandomForestClassifier`-in PR əyri `SGDClassifier`-inkindən qat-qat yaxşı görünür: o, yuxarı sağ künçə daha çox yaxınlaşır. Onun F\_1 xalı və ROC AUC xalı da əhəmiyyətli dərəcədə daha yaxşıdır:

```
>>> y_train_pred_forest = y_probas_forest[:, 1] >= 0.5 # positive proba ≥ 50%
>>> f1_score(y_train_5, y_pred_forest)
0.9242275142688446
>>> roc_auc_score(y_train_5, y_scores_forest)
0.9983436731328145
```

İndi dəqiqlik və əhatəlilik (recall) xallarını ölçməyə çalışın: təxminən 99.1% dəqiqlikvə 86.6% əhatəlilik (recall) əldə etməliyik. Heç də pis deyil!

Artıq ikili təsnifatçıları necə təlim edəcəyinizi, məsələniz üçün uyğun ölçünü (metrikanı) necə seçəcəyinizi, çarbaz yoxlama

(cross-validation) istifadə edərək təsnifatçılarını necə qiymtləndirəcəyinizi, ehtiyaclarınıza uyğun dəqiqlik/əhatəlilik kompromisini necə seçəcəyinizi və müxtəlif modelləri müqayisə etmək üçün bir neçə ölçü və əyridən necə istifadə edəcəyinizi bilirsiniz.

Siz artıq sadəcə 5-ləri deyil, daha çoxunu aşkar etməyə çalışmağa hazırlısınız.

## Çoxsinifli Təsnifat

İkili təsnifatçılar iki sinfi bir-birindən ayırdığı halda, çoxsinifli təsnifatçılar (həmçinin çoxhədli təsnifatçılar da adlanır) ikidən çox sinfi bir-birindən ayırd edə bilər.

Bəzi Scikit-Learn təsnifatçıları (məsələn, [LogisticRegression](#), [RandomForestClassifier](#) və [GaussianNB](#)) çoxlu siniflərlə doğrudan işləmək qabiliyyətinə malikdir. Digərləri isə yalnız ikili təsnifatçılardır (məsələn, [SGDClassifier](#) və [SVC](#)). Lakin, çoxlu sayda ikili təsnifatçı ilə çoxsinifli təsnifatı həyata keçirmək üçün istifadə edə biləcəyiniz müxtəlif strategiyalar mövcuddur.

Rəqəm təsvirlərini 10 sinfə (0-dan 9-a qədər) təsnif edə bilən bir sistem yaratmağın bir yolu, hər rəqəm üçün bir ədəd olmaqla 10 ikili təsnifatçı təlim etməkdir (bir 0-detektoru, bir 1-detektoru, bir 2-detektoru və s.). Sonra bir təsviri təsnif etmək istədiyiniz zaman, həmin təsvir üçün hər bir təsnifatçıdan qərar xalını (decision score) alırsınız və təsnifatçısı ən yüksək xalı verən sinfi seçirsiniz. Bu, "*biri-qalanlara-qarşı*" (OvR) və ya bəzən "*biri-hamıya-qarşı*" (OvA) strategiyası adlanır.

Başqa bir strategiya, hər bir rəqəm cütlüyü üçün ikili təsnifatçı təlim etməkdir: biri 0-ları və 1-ləri ayırd etmək üçün, digəri 0-ları və 2-ləri ayırd etmək üçün, başqa biri 1-lər və 2-lər üçün və s. Bu, "*birin-birə-qarşı*" (OvO) strategiyası adlanır. Əgər  $N$  sinif varsa, sizin  $N \times (N - 1)/2$  sayda təsnifatçı təlim etməyiniz lazımdır. MNIST məsələsi üçün bu, 45 ikili təsnifatçı təlim etmək deməkdir! Bir təsviri təsnif etmək istədikdə, siz təsviri bütün 45 təsnifatçıdan keçirməli və hansı sinfin ən çox duel qazandığına baxmalısınız. OvO-nun əsas üstünlüyü odur ki, hər bir təsnifatçının yalnız ayırd etməli olduğu iki sinfi ehtiva edən təlim toplusu hissəsi üzərində təlim edilməsinə ehtiyac var.

Bəzi alqoritmlər (məsələn, dəstək vektor maşınları (SVM) təsnifatçıları) təlim toplusunun ölçüsü ilə zəif miqyaslanır. Bu alqoritmlər üçün OvO-ya üstünlük verilir, çünkü kiçik təlim topluları

Üzərində çox sayda təsnifatçı təlim etmək, böyük təlim topluları üzərində az sayda təsnifatçı təlim etməkdən daha sürətlidir. Lakin, əksər ikili təsnifat alqoritmləri üçün OvR-ə üstünlük verilir.

Scikit-Learn, siz çoxsinifli təsnifat məsələsi üçün ikili təsnifat alqoritmindən istifadə etməyə çalışdığınıza aşkarlayır və alqoritmdən asılı olaraq avtomatik olaraq OvR və ya OvO-nu işə salır. Gəlin bunu `sklearn.svm.SVC` sinfini istifadə edərək dəstək vektor maşını təsnifatçısı ilə sınayaq (bax: [Fəsil 5](#)). Biz yalnız ilk 2,000 təsvir üzərində təlim keçəcəyik, əks halda bu, çox uzun çəkəcək:

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

Biz SVC-ni "5-ə-qarşı-qalanlar" hədəf sinifləri (`y_train_5`) əvəzinə, 0-dan 9-a qədər olan ilkin (orijinal) hədəf sinifləri (`y_train`) üzərində təlim etdik. 10 sinif (yəni 2-dən çox) olduğu üçün Scikit-Learn OvO strategiyasından istifadə etdi və 45 ikili təsnifatçı təlim etdi. İndi gəlin bir təsvir üzərində proqnoz verək:

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

Bu doğrudur! Bu kod əslində 45 proqnoz verdi (hər sinif cütlüyü üçün bir ədəd) və ən çox duel qazanan sinfi seçdi. Əgər `decision_function()` metodunu çağırısanız, görəcəksiniz ki, o, hər nümunə üçün 10 xal qaytarır: hər sinif üçün bir xal. Hər bir sinif, təsnifatçı xallarına əsaslanaraq, qazanılmış duellərin sayına bərabər bir xal alır; bərabərlik halını pozmaq (break ties) üçün bu xala kiçik bir düzəliş (maksimum  $\pm 0.33$ ) əlavə edilir və ya çıxılır:

```
>>> some_digit_scores = svm_clf.decision_function([some_digit])  
>>> some_digit_scores.round(2)  
array([[ 3.79,  0.73,  6.06,  8.3 , -0.29,  9.3 ,  1.75,  2.77,  7.21,  
       4.82]])
```

Ən yüksək xal həqiqətən də 5-ci sınıfə uyğun gələn 9.3 xalıdır:

```
>>> class_id = some_digit_scores.argmax()  
>>> class_id  
5
```

Təsnifatçı təlim edildikdə, o, hədəf siniflərin siyahısını dəyərə görə sıralanmış şəkildə öz `classes_` atributunda saxlayır. MNIST nümunəsində `classes_` massivindəki hər bir sınıfın indeksi əlverişli bir şəkildə elə sınıfın özü ilə üst-üstə düşür (məsələn, 5-ci indeksdə olan sınıf elə '5' sınıfı olur), lakin ümumi halda bəxtiniz belə gətirməyəcək; sınıf etiketini bu şəkildə axtarmalı olacaqsınız:

```
>>> svm_clf.classes_  
array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'], dtype=object)  
>>> svm_clf.classes_[class_id]  
'5'
```

Əgər Scikit-Learn-ü "birin-birə-qarşı" (one-versus-one) və ya "birin-qalanlara-qarşı" (one-versus-the-rest) strategiyasını istifadə etməyə məcbur etmək istəyirsizsə, `OneVsOneClassifier` və ya `OneVsRestClassifier` sınıflarından istifadə edə bilərsiniz.

Sadəcə bir nümunə (instance) yaradın və onun konstrukturuna bir təsnifatçı ötürün (bu, hətta ikili təsnifatçı olmaya da bilər). Məsələn, bu kod `SVC`-yə əsaslanan və OvR strategiyasını istifadə edən çoxsinifli bir təsnifatçı yaradır:

```
from sklearn.multiclass import OneVsRestClassifier  
  
ovr_clf = OneVsRestClassifier(SVC(random_state=42))  
ovr_clf.fit(X_train[:2000], y_train[:2000])
```

Gəlin bir proqnoz verək və təlim edilmiş təsnifatçıların sayını yoxlayaq:

```
>>> ovr_clf.predict([some_digit])  
array(['5'], dtype='<U1')  
>>> len(ovr_clf.estimators_)  
10
```

`SGDClassifier`-i çoxsinifli məlumat toplusu üzərində təlim etmək və onunla proqnoz vermək də eyni dərəcədə asandır:

```
>>> sgd_clf = SGDClassifier(random_state=42)
>>> sgd_clf.fit(X_train, y_train)
>>> sgd_clf.predict([some_digit])
array(['3'], dtype='<U1')
```

Proqnoz xətaları baş verir! Hansı ki elə bizim də başımıza gəldi. Bu dəfə Scikit-Learn pərdə arxasında OvR strategiyasından (birin-qalanlara-qarşı) istifadə etdi: 10 sinif olduğu üçün 10 ikili təsnifatçı təlim etdi. `decision_function()` metodu indi hər sinif üçün bir dəyər qaytarır. Gəlin SGD təsnifatçısının hər sinfə təyin etdiyi xallara baxaq:

```
>>> sgd_clf.decision_function([some_digit]).round()
array([[ -31893., -34420., -9531.,  1824., -22320., -1386., -26189.,
       -16148., -4604., -12051.]])
```

Görə bilərsiniz ki, təsnifatçı öz proqnozuna o qədər də əmin deyil: demək olar ki, bütün xallar çox mənfidir, lakin sinif 3-ün xalı +1,824-dür, sinif 5 isə -1,386 xal ilə ondan çox da geri qalmır. Əlbəttə, siz bu təsnifatçını birdən çox təsvir üzərində qiymətləndirmək istəyəcəksiniz. Hər sinifdə təxminən eyni sayda təsvir olduğundan, doğruluq (accuracy) ölçüsü kifayət qədər əlverişlidir. Həmişə olduğu kimi, modeli qiymətləndirmək üçün `cross_val_score()` funksiyasından istifadə edə bilərsiniz:

```
>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([0.87365, 0.85835, 0.8689])
```

O, bütün test qatlarında (test folds) 85.8%-dən yuxarı nəticə əldə edir. Əgər təsadüfi təsnifatçı istifadə etsəydiniz, 10% doğruluq əldə edərdiniz, buna görə də bu o qədər də pis bir nəticə deyil, lakin siz hələ də daha yaxşı nəticə göstərə bilərsiniz. Sadəcə olaraq girişlərin miqyaslandırılması (**Fəsil 2**-də müzakirə edildiyi kimi) dəqiqliyi 89.1%-dən yuxarı qaldırır:

```
>>> from sklearn.preprocessing import StandardScaler  
>>> scaler = StandardScaler()  
>>> X_train_scaled = scaler.fit_transform(X_train.astype("float64"))  
>>> cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")  
array([0.8983, 0.891 , 0.9018])
```

## Xəta Təhlili (Error Analysis)

Əgər bu, real bir layihə olsaydı, siz indi maşın öyrənməsi layihənizin yoxlama siyahısındaki (bax: **Əlavə A**) addımları izləyərdiniz. Siz məlumatların hazırlanması (data preparation) seçimlərini araşdırar, çoxsaylı modelləri sınaqdan keçirər, ən yaxşalarını qısa siyahıya alar, onların hiperparametrlərini **GridSearchCV** istifadə edərək incə tənzimləmələr edər və mümkün qədər çox şeyi avtomatlaşdırardınız. Burada, biz fərz edəcəyik ki, siz ümidverici bir model tapmısınız və onu təkmilləşdirmək yollarını tapmaq istəyirsiniz. Bunu etməyin bir yolu, onun buraxdığı xətaların növlərini təhlil etməkdir.

Əvvəlcə, xəta matrisinə (confusion matrix) baxın. Bunun üçün, ilk növbədə **cross\_val\_predict()** funksiyasından istifadə edərək proqnozlar etməlisiniz; sonra, eynilə əvvəl etdiyiniz kimi, etiketləri və proqnozları **confusion\_matrix()** funksiyasına ötürə bilərsiniz. Lakin, indi 2 sinif əvəzinə 10 sinif olduğu üçün, xəta matrisi xeyli çox rəqəm ehtiva edəcək və onu oxumaq çətin ola bilər.

Xəta matrisinin rəngli bir diaqramını təhlil etmək daha asandır. Belə bir diaqramı qurmaq (çəkmək) üçün **ConfusionMatrixDisplay.from\_predictions()** funksiyasından bu şəkildə istifadə edin:

```
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()
```

Bu, **Şəkil 3-9**-dakı sol diaqramı yaradır. Bu xəta matrisi olduqca yaxşı görünür: təsvirlərin eksəriyyəti əsas diaqonal üzərindədir ki, bu da onların düzgün təsnif edildiyi deməkdir. Diqqət yetirin ki, 5-ci sətir və 5-ci sütundakı diaqonal üzərindəki xana digər rəqəmlərdən bir qədər daha tünd görünür. Bu, ona görə ola bilər ki, ya model 5-lər üzərində daha çox xəta buraxıb, ya da məlumat toplusunda 5-lərin sayı digər rəqəmlərdən daha azdır. Məhz buna görə də xəta matrisini, hər bir dəyəri müvafiq sinifdəki təsvirlərin ümumi sayına bölməklə (yəni, sətrin cəminə bölməklə) normallaşdırmaq vacibdir.

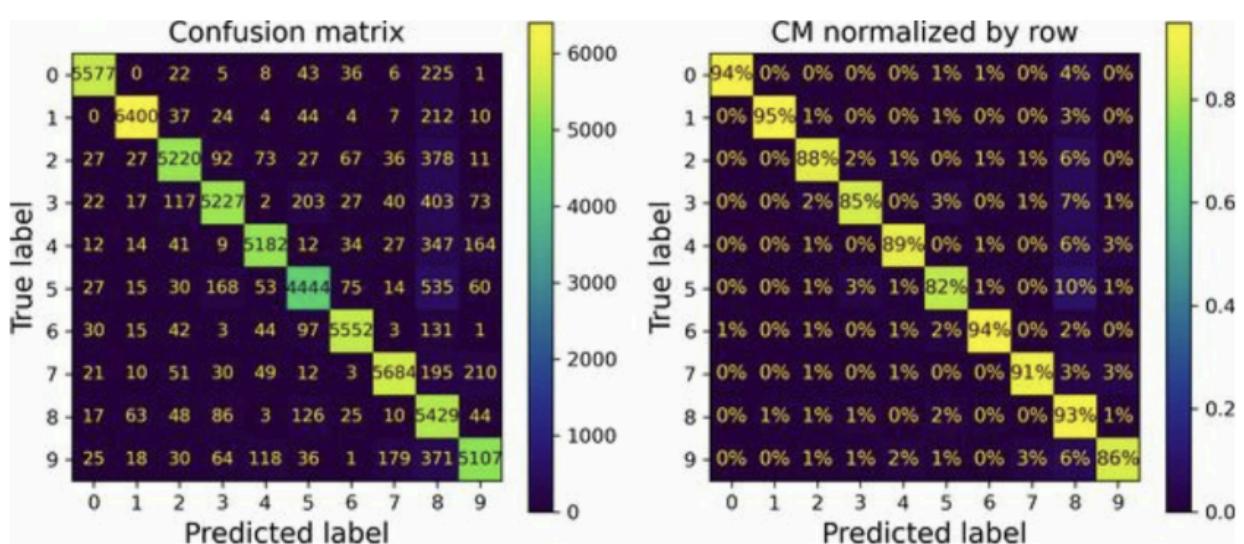
Bunu sadəcə `normalize="true"` təyin etməklə etmək olar. Biz həmçinin faizləri kəsr hissəsiz (ondalıq rəqəmsiz) göstərmək üçün `values_format=".0%"` arqumentini də təyin edə bilərik.

Aşağıdakı kod [Şəkil 3-9](#)-dakı sağ tərəfdəki diaqramı yaradır:

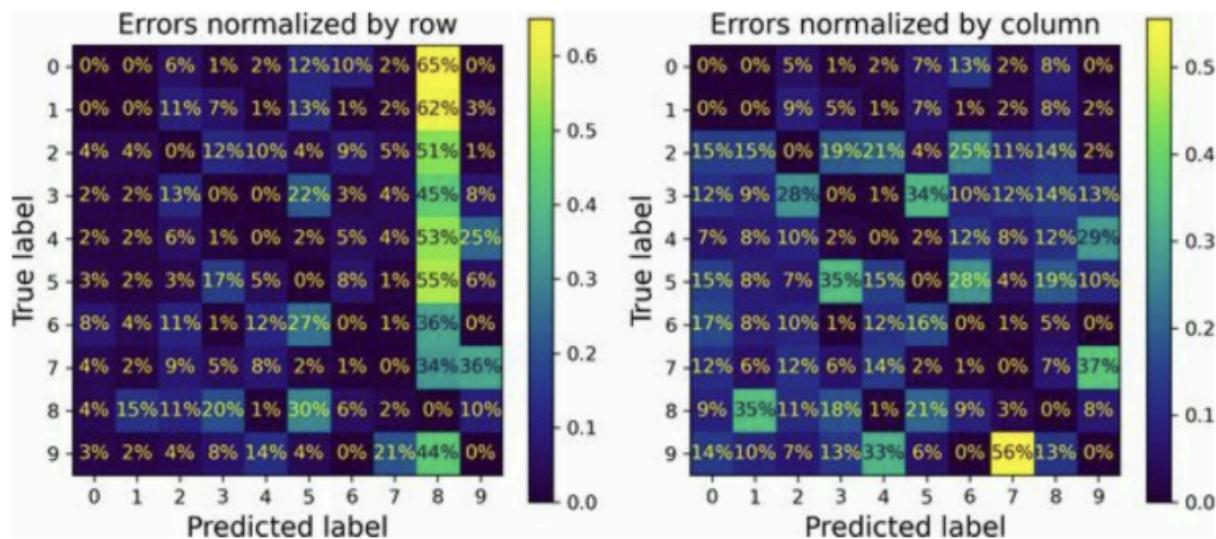
```
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred,
                                         normalize="true", values_format=".0%")
plt.show()
```

İndi biz asanlıqla görə bilərik ki, 5 rəqəminin təsvirlərinin yalnız 82%-i düzgün təsnif edilib. Modelin 5 rəqəminin təsvirləri ilə bağlı etdiyi ən çox yayılmış xəta, onları 8 kimi səhv təsnif etməsi olub: bu, bütün 5-lərin 10%-i üçün baş verib. Lakin 8-lərin yalnız 2%-i 5 kimi səhv təsnif edilib; xəta matrisləri (confusion matrices) adətən simmetrik olmur! Əgər diqqətlə baxsanız, bir çox rəqəmin 8 kimi səhv təsnif edildiyini görərsiniz, lakin bu, həmin diaqramdan dərhal açıq-aydın görünmür. Əgər xətaları daha çox nəzərə çarpan etmək istəyirsinizsə, düzgün proqnozlara sıfır çəki (zero weight) verməyə cəhd edə bilərsiniz. Aşağıdakı kod məhz bunu edir və [Şəkil 3-10](#)-dakı sol diaqramı yaradır:

```
sample_weight = (y_train_pred != y_train)
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred,
                                         sample_weight=sample_weight,
                                         normalize="true", values_format=".0%")
plt.show()
```



[Şəkil 3-9. Xəta matrisi \(solda\) və sətrə görə normallaşdırılmış eyni CM \(sağda\)](#)



*Şəkil 3-10. Yalnız xətaların olduğu xəta matriisi; sətrə görə (solda) və sütuna görə (sağda) normallaşdırılmışdır.*

İndi təsnifatçının etdiyi xəta növlərini daha aydın görə bilərsiniz. 8-ci sinif üçün olan sütun indi həqiqətən də parlaqdır ki, bu da bir çox təsvirin 8 kimi səhv təsnif edildiyini təsdiqləyir. Əslində, bu, demək olar ki, bütün siniflər üçün ən çox rast gəlinən səhv təsnifatdır.

Lakin bu diaqramdakı fazləri necə şərh etdiyinizə diqqət edin: unutmayın ki, biz düzgün proqnozları xaric etmişik. Məsələn, 7-ci sətir, 9-cu sütundakı 36% o demək deyil ki, bütün 7 rəqəmi təsvirlərinin 36%-i 9 kimi səhv təsnif edilib. Bu o deməkdir ki, modelin 7 rəqəmi təsvirlərində etdiyi xətaların 36%-i onların 9 kimi səhv təsnif edilməsindən ibarət olub. Reallıqda isə, **Şəkil 3-9**-dakı sağ diaqramda görə biləcəyiniz kimi, 7 rəqəmi təsvirlərinin cəmi 3%-i 9 kimi səhv təsnif edilmişdi.

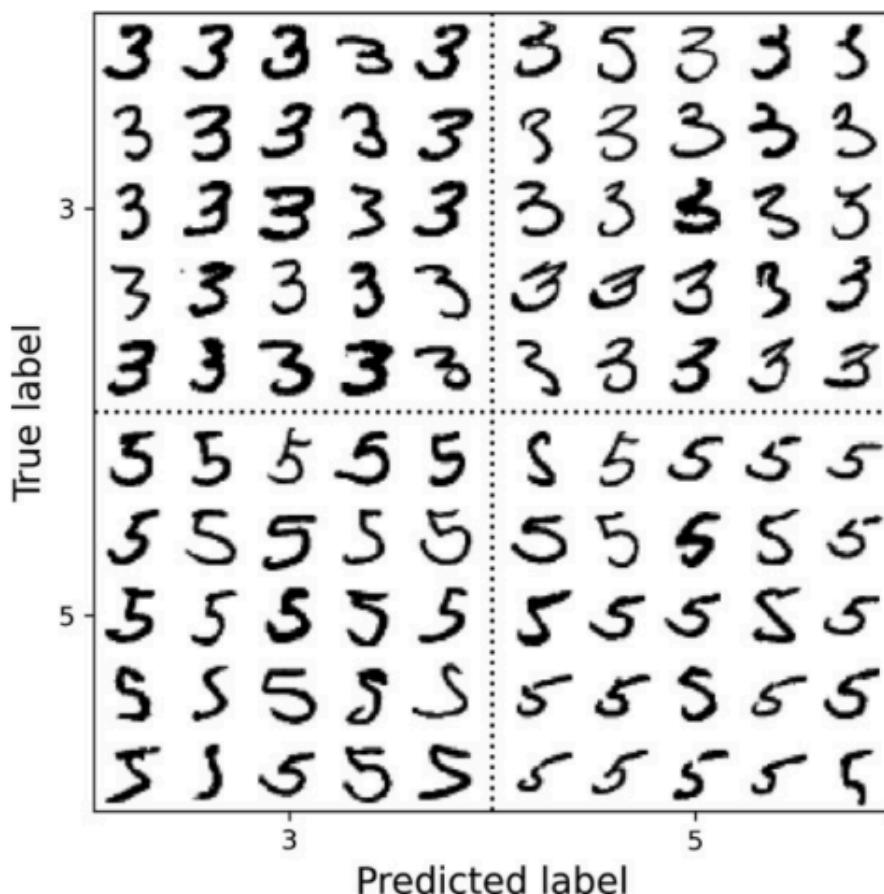
Xəta matrişini sətrə görə deyil, sütuna görə normallaşdırmaq da mümkünündür: əgər `normalize="pred"` təyin etsəniz, **Şəkil 3-10**-dakı sağ diaqramı əldə edərsiniz. Məsələn, görə bilərsiniz ki, səhv təsnif edilmiş 7-lərin 56%-i əslində 9-dur.

Xəta matrişini təhlil etmək çox vaxt sizə təsnifatçınızı təkmilləşdirmək yolları haqqında ipucular verir. Bu qrafiklərə baxdıqda, elə görünür ki, səyləriniz yalançı 8-ləri (false 8s) azaltmağa yönəldilməlidir. Məsələn, 8-ə oxşayan (amma 8 olmayan) rəqəmlər üçün daha çox təlim məlumatı toplamağa cəhd edə bilərsiniz ki, təsnifatçı onları həqiqi 8-lərdən ayırd etməyi

öyrənə bilsin. Yaxud, təsnifatçıya kömək edəcək yeni xüsusiyyətlər yarada (engineer new features) bilərsiniz – məsələn, qapalı ilgəklərin (loops) sayını hesablamaq üçün bir alqoritm yazmaq (məsələn, 8-də iki, 6-da bir, 5-də heç yoxdur). Və ya, qapalı ilgəklər kimi bəzi nümunələri (patterns) daha çox nəzərə çarpan etmək üçün təsvirləri öncədən emal edə (preprocess) bilərsiniz (məsələn, Scikit-Image, Pillow və ya OpenCV istifadə edərək).

Fərdi xətaları təhlil etmək də təsnifatçınızın nə etdiyini və niyə uğursuz olduğunu dərindən anlamaq üçün yaxşı bir yol ola bilər. Məsələn, gəlin 3-lər və 5-lərin nümunələrini xəta matrisi üslubunda qrafikdə quraq (**Şəkil 3-11**):

```
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
```



Şəkil 3-11. Xəta matrisi kimi təşkil edilmiş bəzi 3 və 5 təsvirləri

Gördüyünüz kimi, təsnifatçının səhv etdiyi (yəni aşağı sol və yuxarı sağ bloklardakı) bəzi rəqəmlər o qədər pis yazılıb ki, hətta bir insan da onları təsnif etməkdə çətinlik çəkərdi. Lakin, səhv təsnif edilmiş təsvirlərin əksəriyyəti bizə açıq-aşkar səhv'lər kimi görünür.

Təsnifatçının niyə belə səhv'lər etdiyini başa düşmək çətin ola bilər, lakin unutmayın ki, insan beyni fantastik bir nümunə tanıma sistemidir (pattern recognition system) və bizim görmə sistemimiz hər hansı bir məlumat hətta bizim şüurumuza çatmadan əvvəl çoxlu mürəkkəb ilkin emal həyata keçirir. Ona görə də, bu tapşırığın sadə görünməsi onun həqiqətən sadə olduğu demək deyil.

Xatırlayın ki, biz sadə bir **SGDClassifier** istifadə etdik ki, bu da sadəcə bir xətti modeldir: onun etdiyi tək şey hər bir sinif üçün hər pikselə bir çəki təyin etməkdir və o, yeni bir təsvir gördükdə, hər bir sinif üçün bir xal almaq məqsədilə sadəcə ağırlıq verilmiş piksel intensivliklərini cəmləyir. 3-lər və 5-lər yalnız bir neçə pikselə görə fərqləndiyi üçün bu model onları asanlıqla səhv salacaq.

3-lər və 5-lər arasındaki əsas fərq, üst xətti aşağı qövsə birləşdirən kiçik xəttin mövqeyidir. Əgər siz 3-ü birləşmə yeri bir qədər sola sürüşmiş şəkildə çəksəniz, təsnifatçı onu 5 kimi təsnif edə bilər və əksinə. Başqa sözlə, bu təsnifatçı təsvirin sürüşdürülməsinə (shifting) və fırlanmasına (rotation) qarşı olduqca həssasdır.

3/5 qarışığılığını azaltmağın bir yolu, onların yaxşı mərkəzləşdirildiyindən və həddən artıq fırlanmadığından əmin olmaq üçün təsvirləri öncədən emal etməkdir. Lakin bu, asan olmaya bilər, çünkü bu, hər bir təsvirin düzgün fırlanmasını proqnozlaşdırmağı tələb edir.

Daha sadə bir yanaşma, təlim toplusunu təlim təsvirlərinin bir qədər sürüşdürülmüş və fırladılmış variantları ilə zənginləşdirməkdən ibarətdir. Bu, modeli belə dəyişikliklərə qarşı daha dözümlü (tolerant) olmayı öyrənməyə məcbur edəcək. Buna məlumatların artırılması (*data augmentation*) deyilir (bunu **Fəsil 14**-də əhatə edəcəyik; həmçinin bu fəslin sonundakı 2-ci tapşırığa baxın).

## Çoxetiketli Təsnifat

İndiyə qədər hər bir nümunə (instance) həmişə yalnız bir sinfə təyin edilmişdi. Lakin bəzi hallarda siz təsnifatçılarınızın hər bir nümunə üçün bir neçə sinif qaytarmasını istəyə bilərsiniz.

Bir üz tanıma təsnifatçısını nəzərdən keçirək: əgər o, eyni şəkildə bir neçə insanı tanıyırsa, nə etməlidir? O, tanıldığı hər bir şəxs üçün bir etiket (tag) təyin etməlidir. Tutaq ki, təsnifatçı üç üzü tanımaq üçün təlim edilib: Alisa, Bob və Çarli. O zaman təsnifatçıya Alisa və Çarlinin şəkli göstərildikdə, o, **[True, False, True]** (yəni "Alisa var, Bob yoxdur, Çarli var") nəticəsini çıxışa verməlidir.

Çoxlu ikili etiketləri çıxışa verən belə bir təsnifat sistemi çoxetiketli təsnifat sistemi adlanır.

Biz hələlik üz tanıma mövzusunun dərinliyinə getməyəcəyik, lakin gəlin sadəcə əyani nümunə məqsədilə daha sadə bir misala baxaq:

```
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)
```

Bu kod hər bir rəqəm təsviri üçün iki hədəf etiketini ehtiva edən bir **y\_multilabel** massivi yaradır: birincisi rəqəmin böyük (7, 8 və ya 9) olub-olmadığını, ikincisi isə onun tək olub-olmadığını göstərir.

Daha sonra kod, çoxetiketli təsnifatı dəstəkləyən (bütün təsnifatçılar bunu dəstəkləmir) bir **KNeighborsClassifier** nümunəsi yaradır və bu modeli çoxlu hədəflər massivindən istifadə edərək təlim edir. İndi siz proqnoz verə bilərsiniz, diqqət yetirin ki, o, iki etiket qaytarır:

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

Və o, bunu düzgün tapır! 5 rəqəmi həqiqətən də böyük deyil (**False**) və təkdir (**True**).

Çoxetiketli təsnifatçını qiymətləndirməyin bir çox yolu var və doğru ölçünü (metrikanı) seçmək həqiqətən də sizin layihənizdən asılıdır. Yanaşmalardan biri hər bir fərdi etiket üçün  $F_1$  xalını (və ya əvvəl müzakirə edilən hər hansı digər ikili təsnifatçı ölçüsünü) ölçmək, sonra isə sadəcə orta xalı hesablamaqdır.

Aşağıdakı kod bütün etiketlər üzrə orta  $F_1$  xalını hesablayır:

```
>>> 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
```

Bu yanaşma bütün etiketlərin bərabər əhəmiyyətli olduğunu fərz edir, lakin bu, hər zaman belə olmaya bilər.

Xüsusilə, əgər sizdə Alisanın şəkilləri Bob və ya Çarlıdən daha çoxdursa, siz təsnifatçının Alisanın şəkilləri üzrə olan xalına daha çox ağırlıq vermək istəyə bilərsiniz. Sadə seçimlərdən biri hər bir etiketə onun dəstəyinə (həmin hədəf etiketinə malik nümunələrin sayına) bərabər bir çəki verməkdir. Bunu etmək üçün **f1\_score()** funksiyasını çağırarkən sadəcə **average="weighted"** təyin edin.

Əgər **SVC** kimi çoxetiketli təsnifatı birbaşa dəstəkləməyən bir təsnifatçıdan istifadə etmək istəyirsinizsə, mümkün strategiyalardan biri hər etiket üçün bir model təlim etməkdir. Lakin bu strategiya etiketlər arasındakı asılılıqları tutmaqda çətinlik çəkə bilər.

Məsələn, böyük rəqəmin (7, 8 və ya 9) tək olması ehtimalı cüt olması ehtimalından iki dəfə çoxdur, lakin "tək" etiketi üçün olan təsnifatçı "böyük" etiketi üçün olan təsnifatçının nə proqnozlaşdırıldığını bilmir.

Bu problemi həll etmək üçün modellər bir zəncir (chain) şəklində təşkil edilə bilər: model proqnoz verərkən, giriş xüsusiyyətlərinə əlavə olaraq, zəncirdə ondan əvvəl gələn modellərin bütün proqnozlardan istifadə edir.

Şad xəbər odur ki, Scikit-Learn-də məhz bunu edən **ChainClassifier** adlı sinif mövcuddur! Susmaya görə (standart

olaraq), o, təlim üçün həqiqi etiketlərdən istifadə edir və hər bir modelə, zəncirdəki mövqeyindən asılı olaraq müvafiq etiketləri ötürür.

Lakin əgər `cv` hiperparametrini təyin etsəniz, o, təlim toplusundakı hər bir nümunə üçün hər bir təlim edilmiş modeldən "təmiz" (önündən kənar) proqnozlar almaq məqsədilə çarraz yoxlamadan istifadə edəcək. Sonra bu proqnozlar zəncirdə daha sonra gələn bütün modelləri təlim etmək üçün istifadə olunacaq.

Budur, çarraz yoxlama strategiyasından istifadə edərək bir `ChainClassifier`-i necə yaratmaq və təlim etməyi göstərən nümunə. Əvvəl olduğu kimi, işləri sürətləndirmək üçün təlim toplusundakı sadəcə ilk 2,000 təsvirdən istifadə edəcəyik:

```
from sklearn.multioutput import ClassifierChain  
  
chain_clf = ClassifierChain(SVC(), cv=3, random_state=42)  
chain_clf.fit(X_train[:2000], y_multilabel[:2000])
```

İndi proqnozlar vermək üçün bu `ChainClassifier`-dən istifadə edə bilərik:

```
>>> chain_clf.predict([some_digit])  
array([[0., 1.]])
```

## Çoxçixışlı Təsnifat

Burada müzakirə edəcəyimiz sonuncu təsnifat növü çoxçixışlı-çoxsinifli təsnifat (və ya sadəcə çoxçixışlı təsnifat) adlanır.

Bu, hər bir etiketin çoxsinifli ola bildiyi (yəni, ikidən çox mümkün dəyərə malik ola bildiyi) çoxetiketli təsnifatın bir ümumiləşdirilməsidir.

Bunu əyani göstərmək üçün, gəlin təsvirlərdən küyü (noise) təmizləyən bir sistem quraq. O, giriş olaraq küylü rəqəm təsvirini qəbul edəcək və (ümid edirik ki) çıxışa, eynilə MNIST təsvirləri kimi, piksel intensivliklərinin massivi şəklində təmsil olunan təmiz rəqəm təsvirini verəcək. Diqqət yetirin ki, təsnifatçının çıxışı çoxetiketlidir (hər piksel üçün bir etiket) və hər bir etiket çoxlu dəyərlərə malik ola bilər (piksel intensivliyi 0-dan 255-ə qədər dəyişir). Beləliklə, bu, bir çoxçixışlı təsnifat sisteminə nümunədir.

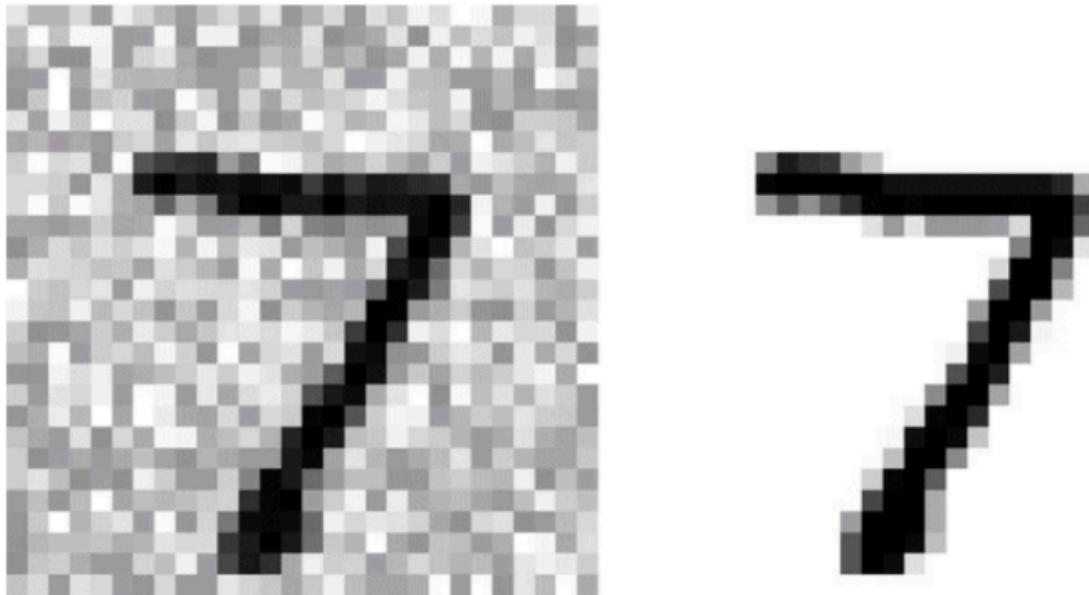
### QEYD

Təsnifat və regressiya arasındaki sərhəd bəzən bulanıq olur, məsələn, bu nümunədə olduğu kimi. Mübahisə etmək olar ki, piksel intensivliyini proqnozlaşdırmaq təsnifatdan daha çox regressiyaya yaxındır. Üstəlik, çoxçixışlı sistemlər təsnifat məsələləri ilə məhdudlaşdır; siz hətta hər bir nümunə üçün həm sinif etiketləri, həm də dəyər etiketləri daxil olmaqla, çoxlu etiket qaytaran bir sistemə sahib ola bilərsiniz.

Gəlin MNIST təsvirlərini götürüb, NumPy-ın `randint()` funksiyası vasitəsilə onların piksel intensivliklərinə küy (noise) əlavə etməklə təlim və test toplularını yaratmaqla başlayaq. Hədəf təsvirlər isə orijinal təsvirlər olacaq:

```
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
```

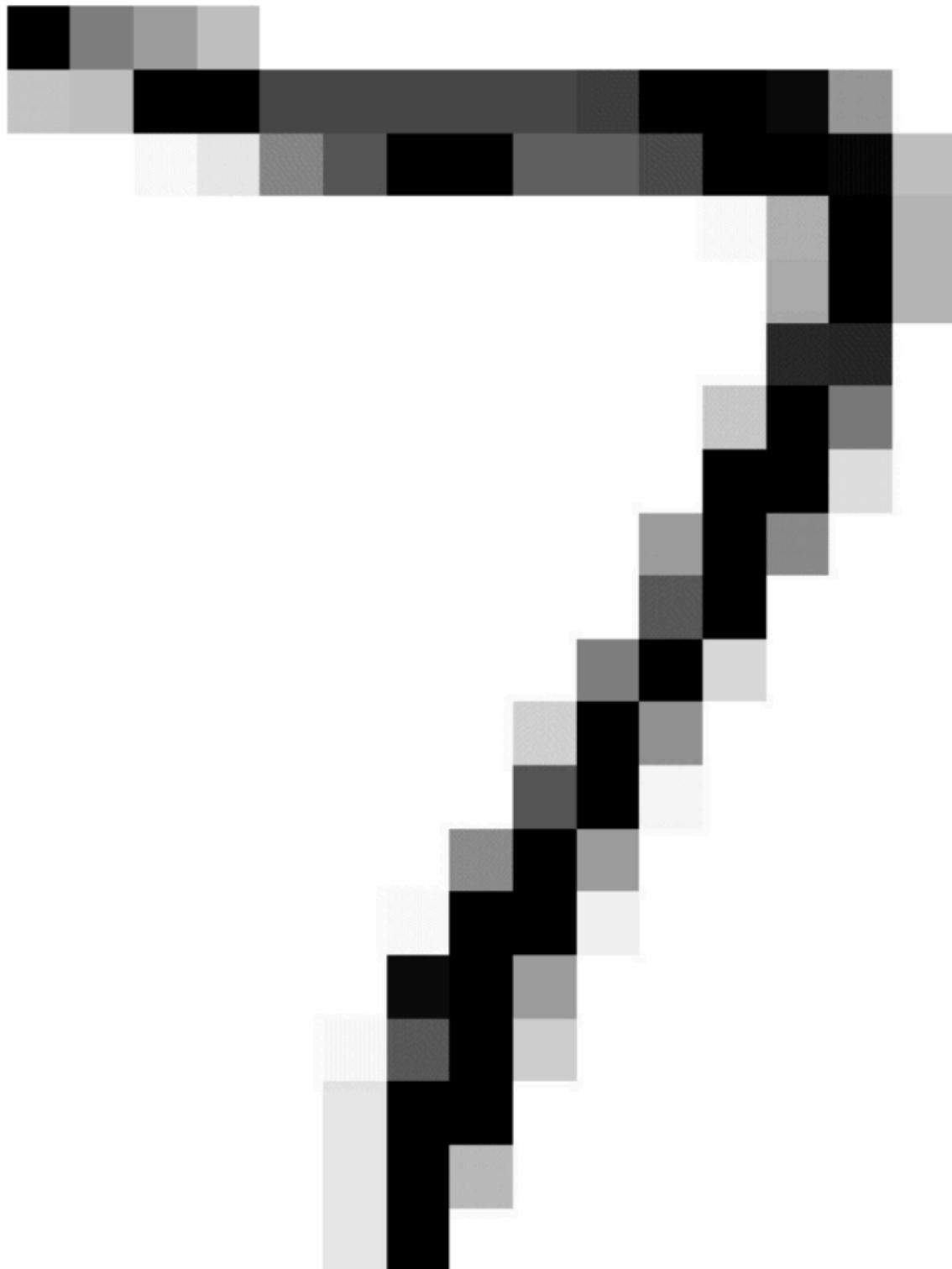
Gəlin test toplusundan olan ilk təsvirə bir nəzər salaq ([Şəkil 3-12](#)). Bəli, biz test məlumatlarına gizlicə baxırıq (*snooping*), “heç etik deyil”.



Şəkil 3-12. Küylü təsvir (solda) və təmiz hədəf təsvir (sağda).

Soldakı küylü giriş təsviri, sağdakı isə təmiz hədəf təsviridir. İndi gəlin təsnifatçını təlim edək və bu təsviri təmizləməsini təmin edək ([Şəkil 3-13](#)):

```
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()
```



Şəkil 3-13. Təmizlənmiş təsvir.

Hədəfə kifayət qədər yaxın görünür!

Bununla da təsnifat (classification) üzrə səyahətimiz yekunlaşır.

Siz artıq təsnifat məsələləri üçün yaxşı ölçüləri (metrikaları) seçməyi, uyğun dəqiqlik/əhatəlilik kompromisini təyin etməyi, təsnifatçıları müqayisə etməyi və daha ümumi desək, müxtəlif tapşırıqlar üçün yaxşı təsnifat sistemləri qurmağı bilirsiniz.

Növbəti fəsillərdə isə, istifadə etdiyiniz bütün bu maşın öyrənməsi modellərinin əslində necə işlədiyini öyrənəcəksiniz. Sizə uğurlar ...

## Tapşırıqlar

1. MNIST məlumat toplusu üçün test toplusunda 97%-dən çox doğruluq (*accuracy*) əldə edən bir təsnifatçı qurmağa çalışın.
  - *İpucu:* `KNeighborsClassifier` bu tapşırıq üçün olduqca yaxşı işləyir; sadəcə yaxşı hiperparametr dəyərlərini tapmalısınız (`weights` və `n_neighbors` hiperparametrləri üzrə "grid search" (tor axtarışı) etməyi sınayın).
2. MNIST təsvirini istənilən istiqamətə (sola, sağa, yuxarı və ya aşağı) bir piksel sürüsdürə bilən funksiya yazın. Sonra, təlim toplusundakı hər bir təsvir üçün dörd sürüsdürülmüş surət (hər istiqamət üçün bir ədəd) yaradın və onları təlim toplusuna əlavə edin. Nəhayət, ən yaxşı modelinizi bu genişləndirilmiş təlim toplusu üzərində təlim edin və test toplusunda onun dəqiqliyini ölçün.
  - Modelinizin indi daha yaxşı nəticə göstərdiyini müşahidə etməlisiniz! Təlim toplusunun bu şəkildə süni böyüdülməsi texnikasına məlumatların artırılması (*data augmentation*) və ya təlim toplusunun genişləndirilməsi deyilir.
3. Titanik məlumat toplusunu ələ alın (həll etməyə çalışın). Başlamaq üçün Kaggle əla yerdir. Alternativ olaraq, məlumatları <https://homl.info/titanic.tgz> ünvanından yükləyə və Fəsil 2-də mənzil məlumatları üçün etdiyiniz kimi bu arxiv (*tarball*) aça bilərsiniz. Bu sizə `pandas.read_csv()` istifadə edərək yükləyə biləcəyiniz iki CSV faylı verəcək: `train.csv` və `test.csv`.
  - Məqsəd, digər sütunlara əsaslanaraq `Survived` (Sağ qaldı) sütununu proqnozlaşdırma bilən bir təsnifatçı təlim etməkdir.
4. Bir spam təsnifatçısı qurun (daha çətin tapşırıq):
  - a. Apache SpamAssassin-in ictimai məlumat toplularından spam və *ham* (spam olmayan) nümunələrini yükleyin.
  - b. Məlumat toplularını arxivdən çıxarın və məlumat formatı ilə tanış olun.
  - c. Məlumatları təlim toplusuna və test toplusuna bölün.

- d. Hər bir e-poçtu xüsusiyyət vektoruna çevirmək üçün məlumat hazırlığı payplaynı (ardıcılığı) yazın. Hazırlıq payplaynınız bir e-poçtu hər bir mümkün sözün mövcudluğunu və ya yoxluğunu göstərən (seyrək) bir vektorə çevirməlidir. Məsələn, bütün e-poçtlar yalnız dörd sözdən ("Salam", "necəsən", "haradasan", "sən") ibarət olsayıdı, "Salam sən Salam Salam sən" e-poçtu **[1, 0, 0, 1]** vektoruna çevrilərdi (yəni ["Salam" var, "necəsən" yoxdur, "haradasan" yoxdur, "sən" var]), və ya hər sözün rast gəlmə sayını saymağa üstünlük verirsinizsə **[3, 0, 0, 2]** kimi.
  - E-poçt başlıqlarını çıxarıb-çıxarmamaq, hər bir e-poçtu kiçik hərflərə çevirmək, durğu işarələrini silmək, bütün URL-ləri "URL" ilə əvəz etmək, bütün rəqəmləri "NUMBER" ilə əvəz etmək və ya hətta stemminq etmək (yəni söz sonluqlarını kəsmək; bunun üçün Python kitabxanaları mövcuddur) kimi addımları idarə etmək üçün hazırlıq payplaynizi hiperparametrlər əlavə etmək istəyə bilərsiniz.
- e. Nəhayət, bir neçə təsnifatçını sınayın və həm yüksək əhatəliliyi (*recall*), həm də yüksək presiyona malik əla bir spam təsnifatçısı qura bilib-bilmədiyinizi yoxlayın.

Bu tapşırıqların həlləri bu fəslin "notebook"unun (iş dəftərinin) sonunda, <https://homl.info/colab3> ünvanında mövcuddur.