Shanghai Advanced Institute of Finance
上海高级金融学院

量化俱乐部-机器学习-Classification

2019-08-04

Menu

1. KNN
2. MNIST
3. Training a Binary Classifier
4. Multiclass Classification
5. Error Analysis
6. Multilabel Classification
7. Performance Measures

自我介绍

张骁喆，2010年大连理工大学软件学院本科毕业，高金FMBA 2017PTE。9年计算机工作经验，熟悉开发，测试，运维，项目管理，产品设计各环节。
曾就职eBay，唯品会，2016加入SAP，现担任SAP Cloud部门开发团队主管。
2017开始接触量化投资和python，目前毕业论文研究方向使用机器学习在A股进行量化投资。
量化投资/机器学习咨询培训/项目管理产品管理。

KNN

1. Cover和Hart在1968年提出了最初的邻近算法
2. 分类算法(Classification)
3. Instance-based learning, lazy learning

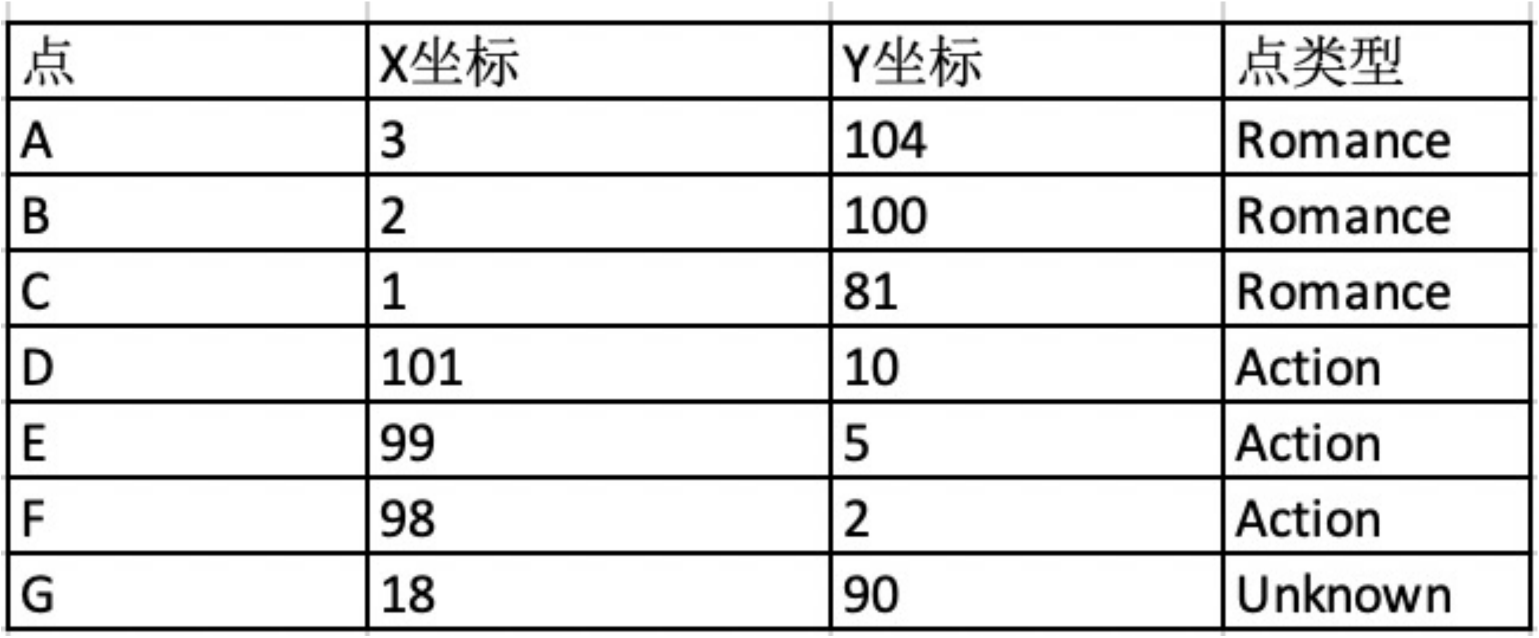
KNN

例子：

Movie	打斗次数	接吻次数	电影类型
California Man	3	104	Romance
He's Not Really into Dudes	2	100	Romance
Beautiful Woman	1	81	Romance
Kevin Longblade	101	10	Action
Robo Slayer 3000	99	5	Action
Amped II	98	2	Action
未知	18	90	Unknown

KNN

未知电影属于什么类型：

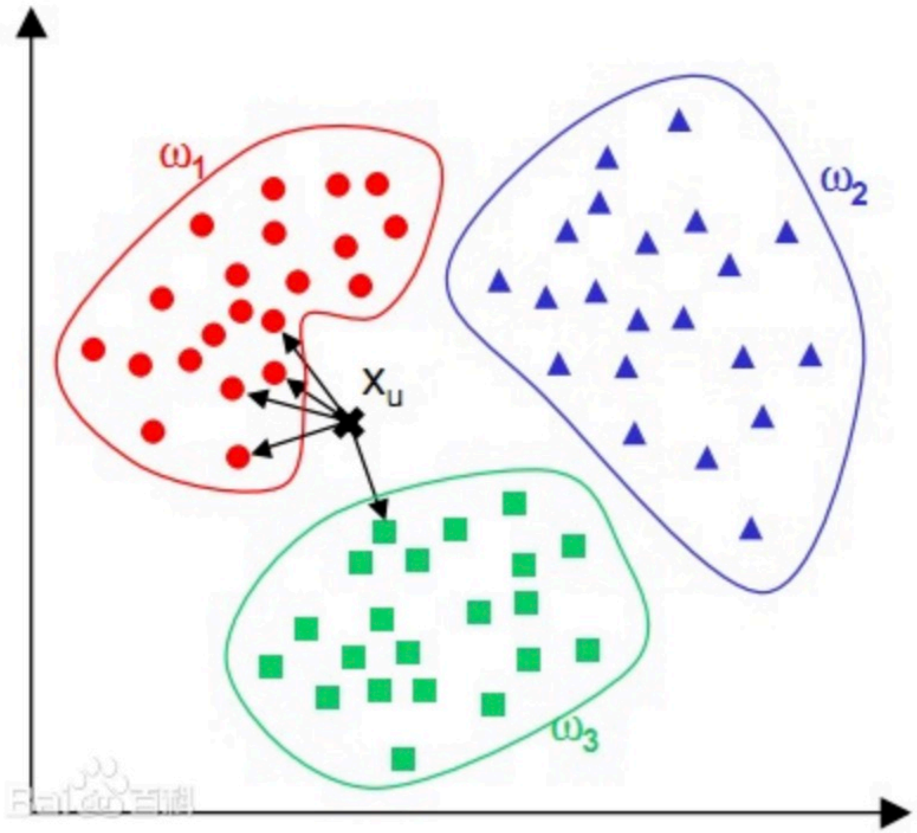


点	X坐标	Y坐标	点类型
A	3	104	Romance
B	2	100	Romance
C	1	81	Romance
D	101	10	Action
E	99	5	Action
F	98	2	Action
G	18	90	Unknown

KNN

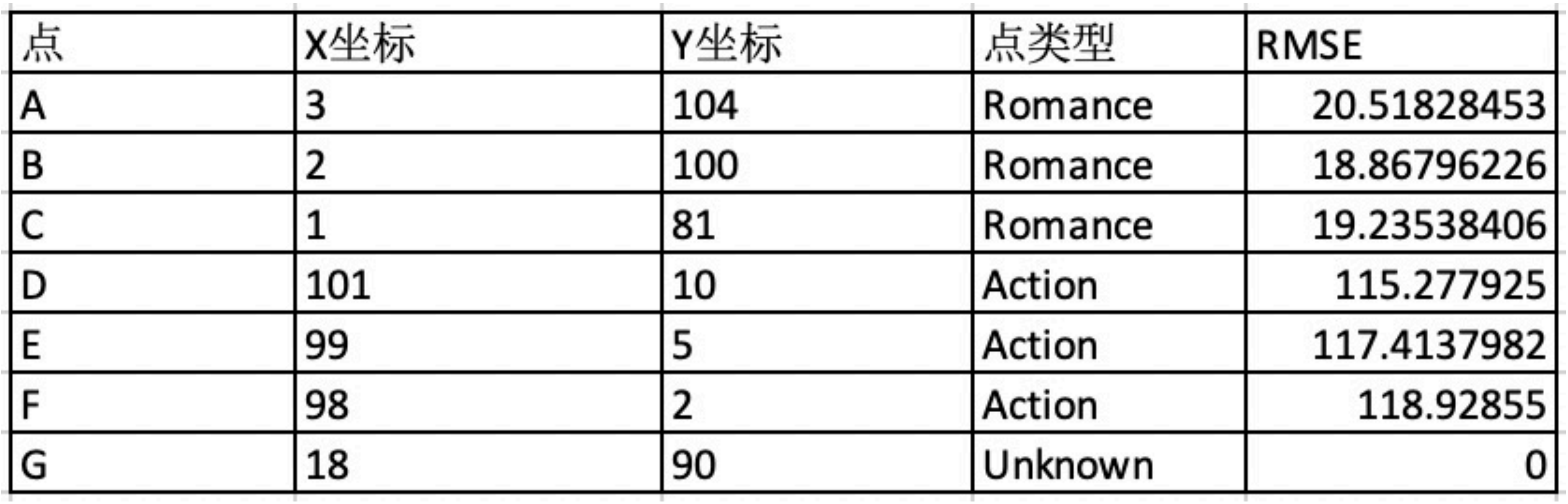
算法详述：
1. 为了判断未知实例的类别，以所有已知类别的实例作为参照。
2. 选择参数K
3. 计算未知实例与所有已知实例的距离
4. 根据大数规则(majority-voting), 让未知实例归类为K个最邻近样本中最多数的类别

细节：
关于K
关于距离的衡量方法l2 form, Euclidean Distance



KNN

K=3：
SQRT(POWER((H9-$H3),2) + POWER(($I$9-$I3),2))
距离最近三个点A，B，C



点	X坐标	Y坐标	点类型	RMSE
A	3	104	Romance	20.51828453
B	2	100	Romance	18.86796226
C	1	81	Romance	19.23538406
D	101	10	Action	115.277925
E	99	5	Action	117.4137982
F	98	2	Action	118.92855
G	18	90	Unknown	0

KNN

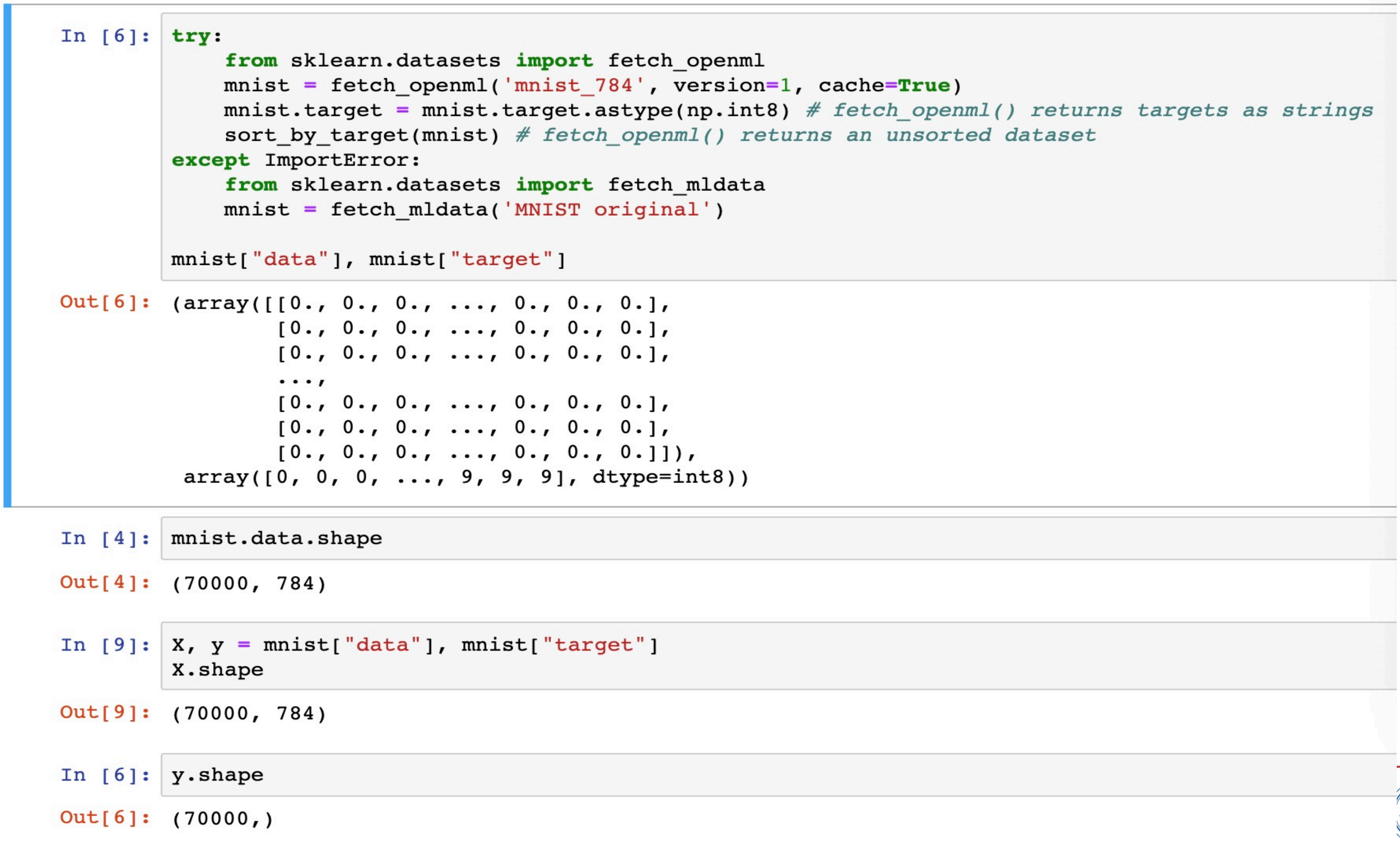
算法优点：
简单
易于理解
容易实现
通过对K的选择可具备丢噪音数据的健壮性

算法缺点：
需要大量空间存储所有已知实例
算法复杂度高(需要比较所有已知实例与未分类实例距离)
当样本分布不平衡时，比如其中一类样本过大，新的未知实例容易被归类为这个主导样本分类。

改进版本：
考虑距离，根据距离加上权重。

MNIST

Scikit-Learn provides many helper functions to download popular datasets. MNIST is one of them

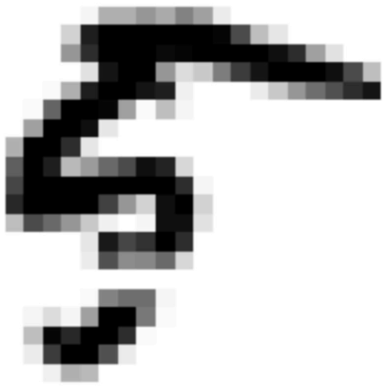


```
In [6]: try:
            from sklearn.datasets import fetch_openml
            mnist = fetch_openml('mnist_784', version=1, cache=True)
            mnist.target = mnist.target.astype(np.int8) # fetch_openml() returns targets as strings
            sort_by_target(mnist) # fetch_openml() returns an unsorted dataset
        except ImportError:
            from sklearn.datasets import fetch_mldata
            mnist = fetch_mldata('MNIST original')

        mnist["data"], mnist["target"]

Out[6]: (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.]]),
         array([0, 0, 0, ..., 9, 9, 9], dtype=int8))

In [4]: mnist.data.shape

Out[4]: (70000, 784)

In [9]: X, y = mnist["data"], mnist["target"]
        X.shape

Out[9]: (70000, 784)

In [6]: y.shape

Out[6]: (70000,)
```

MNIST

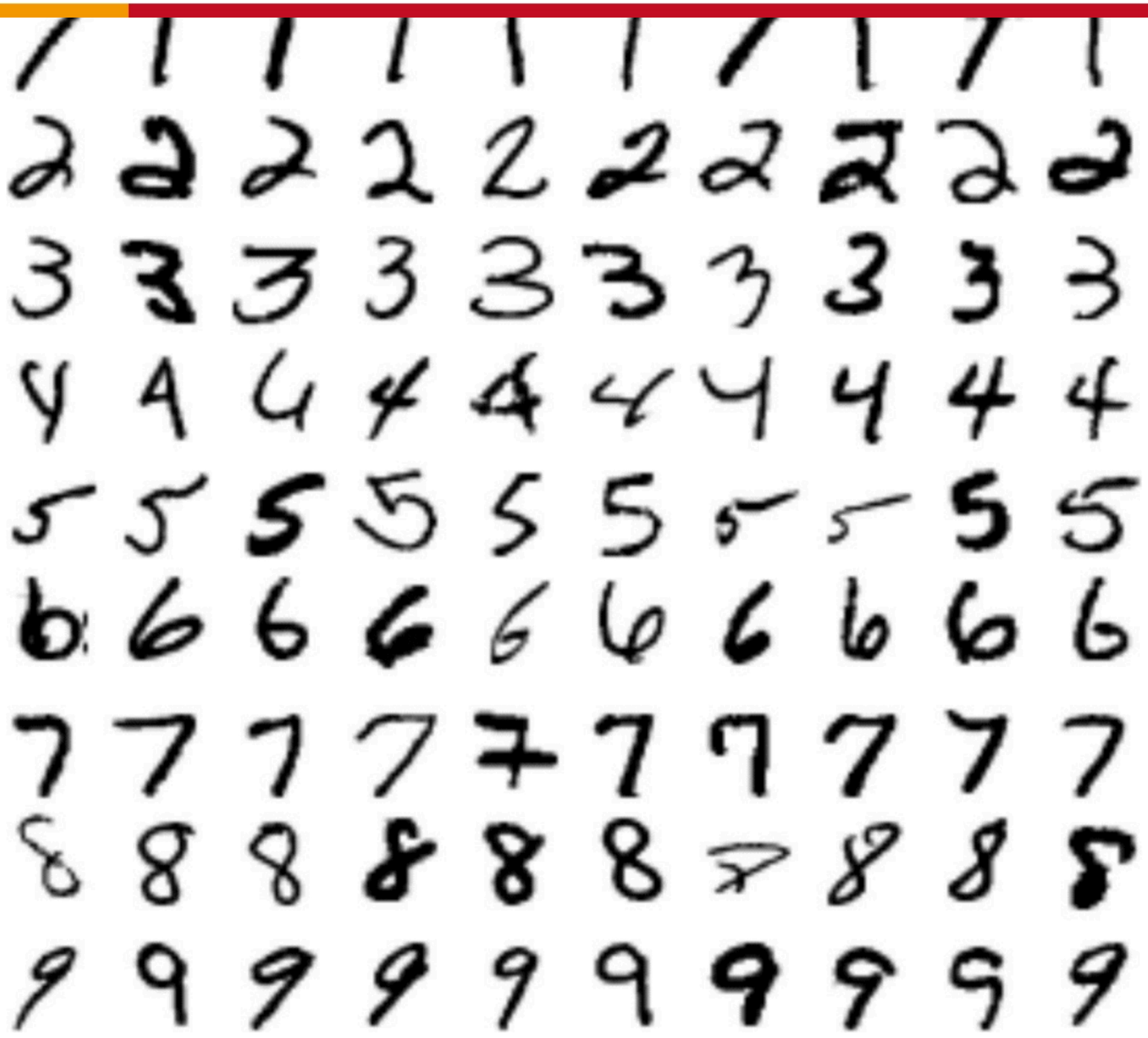
```
In [10]: some_digit = X[32221]
         some_digit_image = some_digit.reshape(28, 28)
         plt.imshow(some_digit_image, cmap = mpl.cm.binary,
                    interpolation="nearest")
         plt.axis("off")

         save_fig("some_digit_plot")
         plt.show()
```

```
Saving figure some_digit_plot
```



MNIST

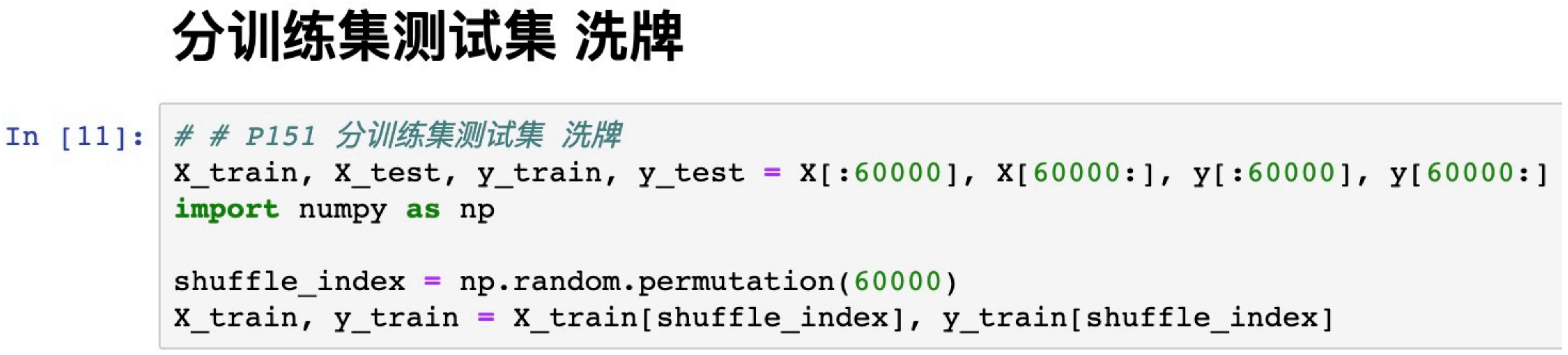
Shanghai Advanced Institute of Finance 上海高级金融学院



MNIST

The MNIST dataset is actually already split into a training set (the first 60,000 images) and a test set (the last 10,000 images)

Let’ s also shuffle the training set; this will guarantee that all cross-validation folds will be similar (you don’ t want one fold to be missing some digits). Moreover, some learning algorithms are sensitive to the order of the training instances, and they perform poorly if they get many similar instances in a row.

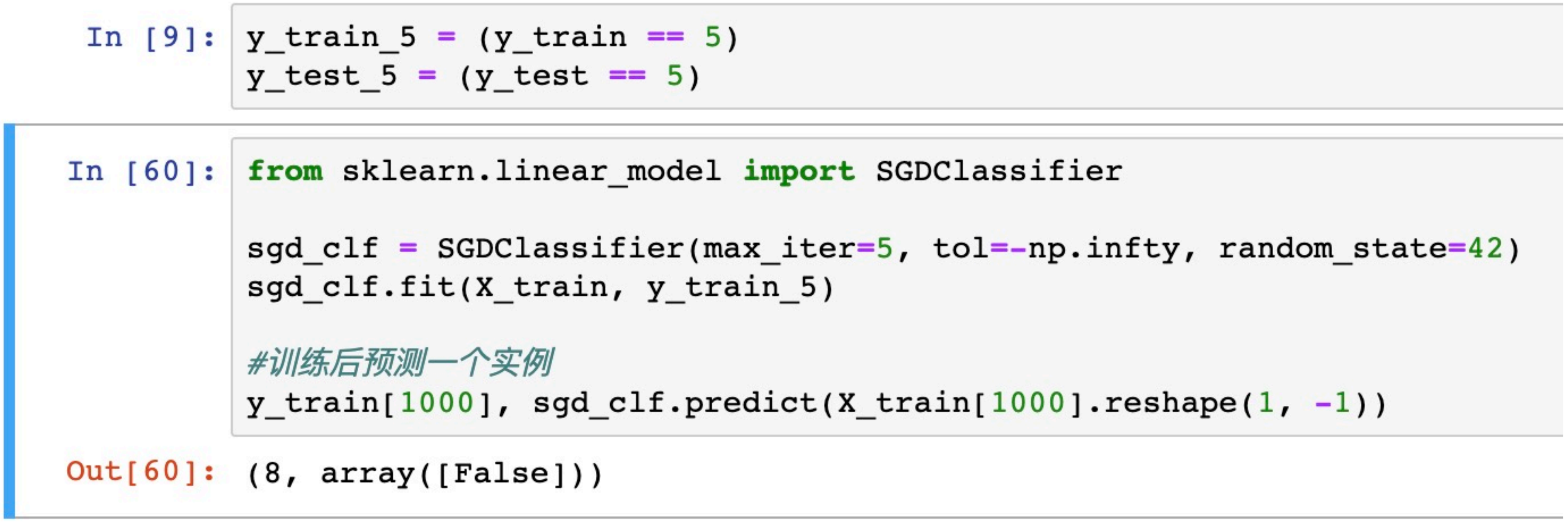


分训练集测试集 洗牌

```
In [11]: # # P151 分训练集测试集 洗牌
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
import numpy as np

shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

Training a binary classifier

Let’ s simplify the problem for now and only try to identify one digit — for example, the number 5. This “5-detector” will be an example of a binary classifier, capable of distinguishing between just two classes, 5 and not-5.



```
In [9]:  y_train_5 = (y_train == 5)
          y_test_5 = (y_test == 5)

In [60]:  from sklearn.linear_model import SGDClassifier

          sgd_clf = SGDClassifier(max_iter=5, tol=-np.infty, random_state=42)
          sgd_clf.fit(X_train, y_train_5)

          #训练后预测一个实例
          y_train[1000], sgd_clf.predict(X_train[1000].reshape(1, -1))

Out[60]:  (8, array([False]))
```

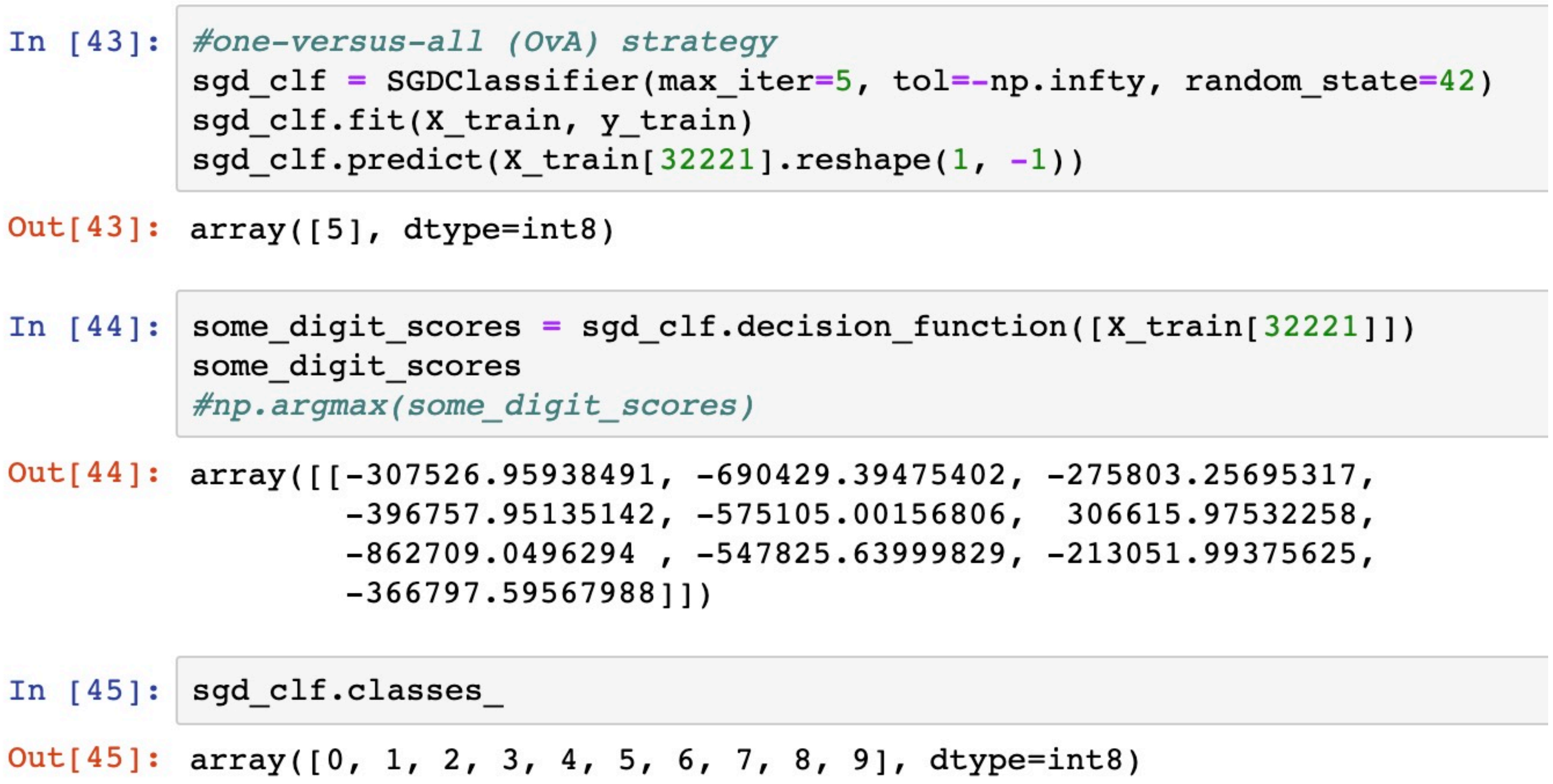
Multi-class classification

Whereas binary classifiers distinguish between two classes, multiclass classifiers (also called multinomial classifiers) can distinguish between more than two classes.

handling multiple classes directly： Random Forest classifiers； naive Bayes classifiers
Binary Classifier: SVM classifier, SGD classifier.   OVO, OVA strategies

Multi-class classification

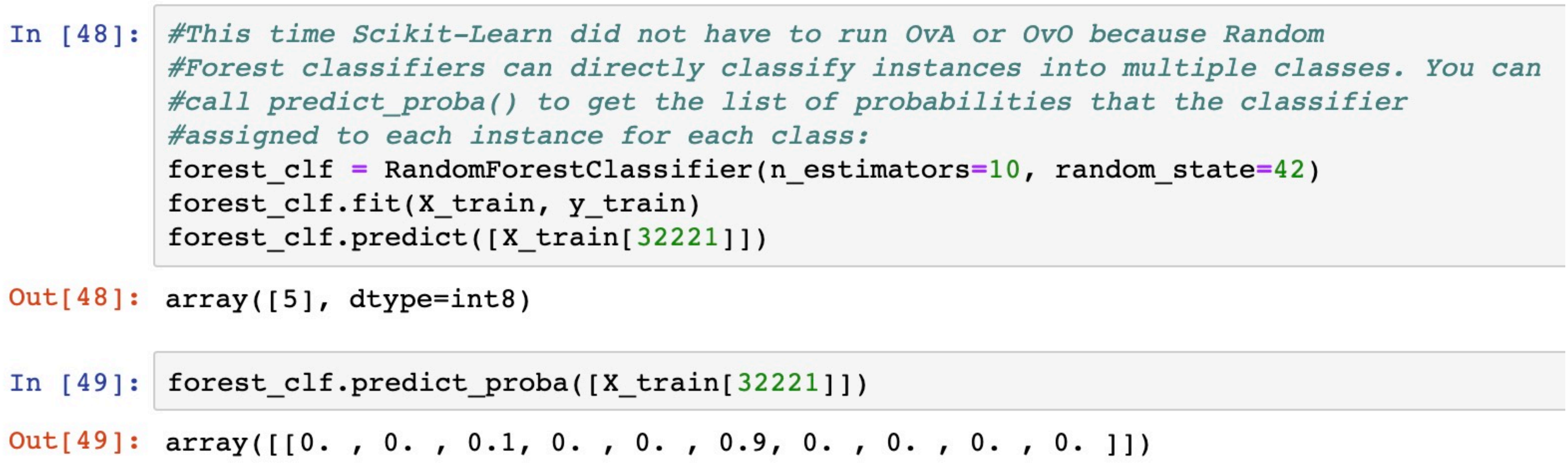
Scikit-Learn detects when you try to use a binary classification algorithm for a multiclass classification task, and it automatically runs OvA.



```
In [43]: #one-versus-all (OvA) strategy
         sgd_clf = SGDClassifier(max_iter=5, tol=-np.infty, random_state=42)
         sgd_clf.fit(X_train, y_train)
         sgd_clf.predict(X_train[32221].reshape(1, -1))

Out[43]: array([5], dtype=int8)

In [44]: some_digit_scores = sgd_clf.decision_function([X_train[32221]])
         some_digit_scores
         #np.argmax(some_digit_scores)

Out[44]: array([[-307526.95938491, -690429.39475402, -275803.25695317,
                 -396757.95135142, -575105.00156806,  306615.97532258,
                 -862709.0496294 , -547825.63999829, -213051.99375625,
                 -366797.59567988]])

In [45]: sgd_clf.classes_

Out[45]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int8)
```

Multi-class classification

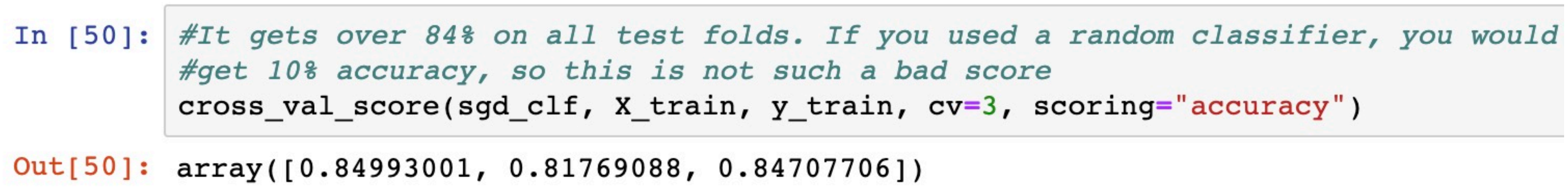
Training a RandomForestClassifier(handling multiple classes directly) is just as easy.



```
In [48]: #This time Scikit-Learn did not have to run OvA or OvO because Random
         #Forest classifiers can directly classify instances into multiple classes. You can
         #call predict_proba() to get the list of probabilities that the classifier
         #assigned to each instance for each class:
         forest_clf = RandomForestClassifier(n_estimators=10, random_state=42)
         forest_clf.fit(X_train, y_train)
         forest_clf.predict([X_train[32221]])

Out[48]: array([5], dtype=int8)

In [49]: forest_clf.predict_proba([X_train[32221]])

Out[49]: array([[0. , 0. , 0.1, 0. , 0. , 0.9, 0. , 0. , 0. , 0. ]])
```

Multi-class classification

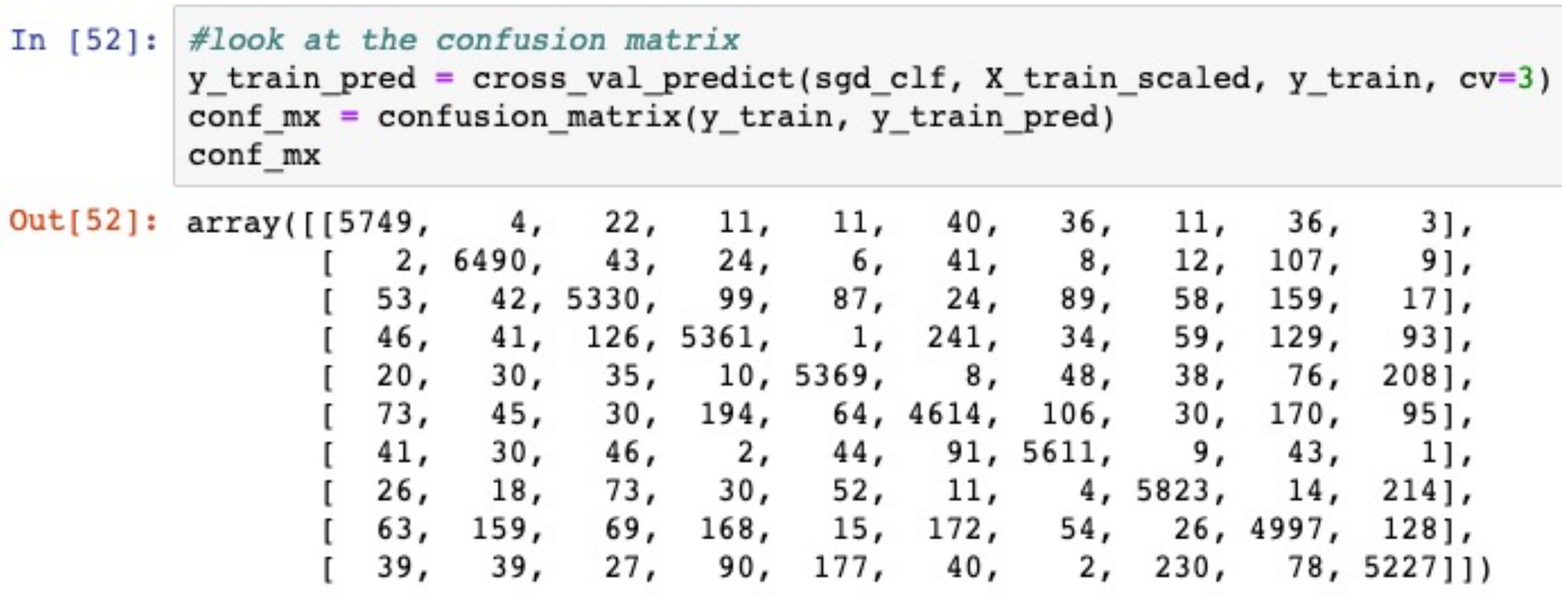
Now of course you want to evaluate these classifiers. As usual, you want to use cross-validation. Let’ s evaluate the SGDClassifier’ s accuracy using the cross_val_score() function



```
In [50]: #It gets over 84% on all test folds. If you used a random classifier, you would
         #get 10% accuracy, so this is not such a bad score
         cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")

Out[50]: array([0.84993001, 0.81769088, 0.84707706])
```

Error Analysis

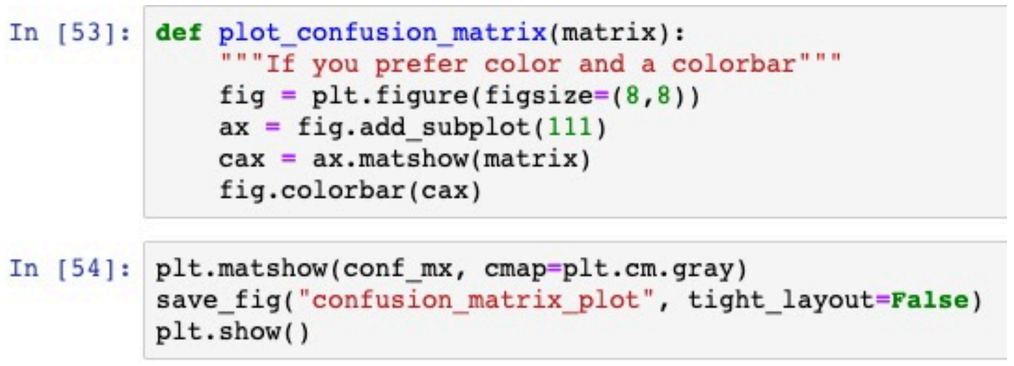
Let’ s take a look at the confusion matrix:



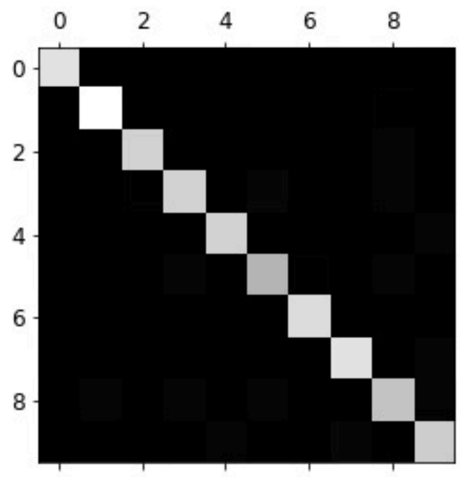
```
In [52]: #look at the confusion matrix
         y_train_pred = cross_val_predict(sgd_clf, X_train_scaled, y_train, cv=3)
         conf_mx = confusion_matrix(y_train, y_train_pred)
         conf_mx

Out[52]: array([[5749,    4,   22,   11,   11,   40,   36,   11,   36,    3],
                [   2, 6490,   43,   24,    6,   41,    8,   12,  107,    9],
                [  53,   42, 5330,   99,   87,   24,   89,   58,  159,   17],
                [  46,   41,  126, 5361,    1,  241,   34,   59,  129,   93],
                [  20,   30,   35,   10, 5369,    8,   48,   38,   76,  208],
                [  73,   45,   30,  194,   64, 4614,  106,   30,  170,   95],
                [  41,   30,   46,    2,   44,   91, 5611,    9,   43,    1],
                [  26,   18,   73,   30,   52,   11,    4, 5823,   14,  214],
                [  63,  159,   69,  168,   15,  172,   54,   26, 4997,  128],
                [  39,   39,   27,   90,  177,   40,    2,  230,   78, 5227]])
```

Error Analysis

using Matplotlib’ s matshow() function



```
In [53]: def plot_confusion_matrix(matrix):
             """If you prefer color and a colorbar"""
             fig = plt.figure(figsize=(8,8))
             ax = fig.add_subplot(111)
             cax = ax.matshow(matrix)
             fig.colorbar(cax)

In [54]: plt.matshow(conf_mx, cmap=plt.cm.gray)
         save_fig("confusion_matrix_plot", tight_layout=False)
         plt.show()

         Saving figure confusion_matrix_plot
```

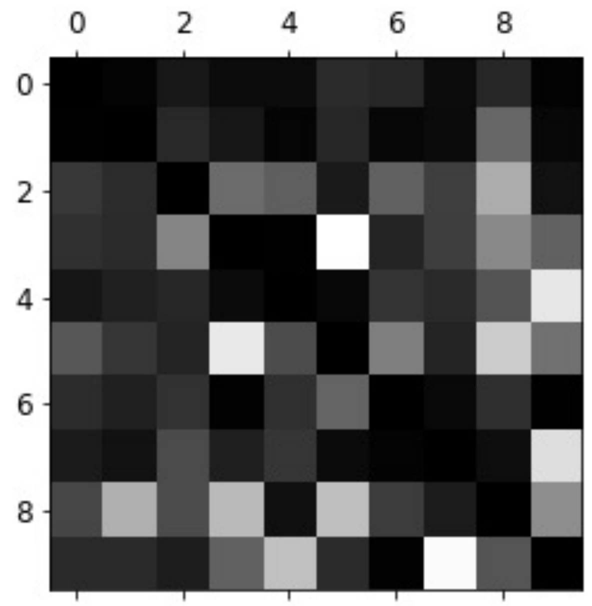


Error Analysis

Now let’ s fill the diagonal with zeros to keep only the errors

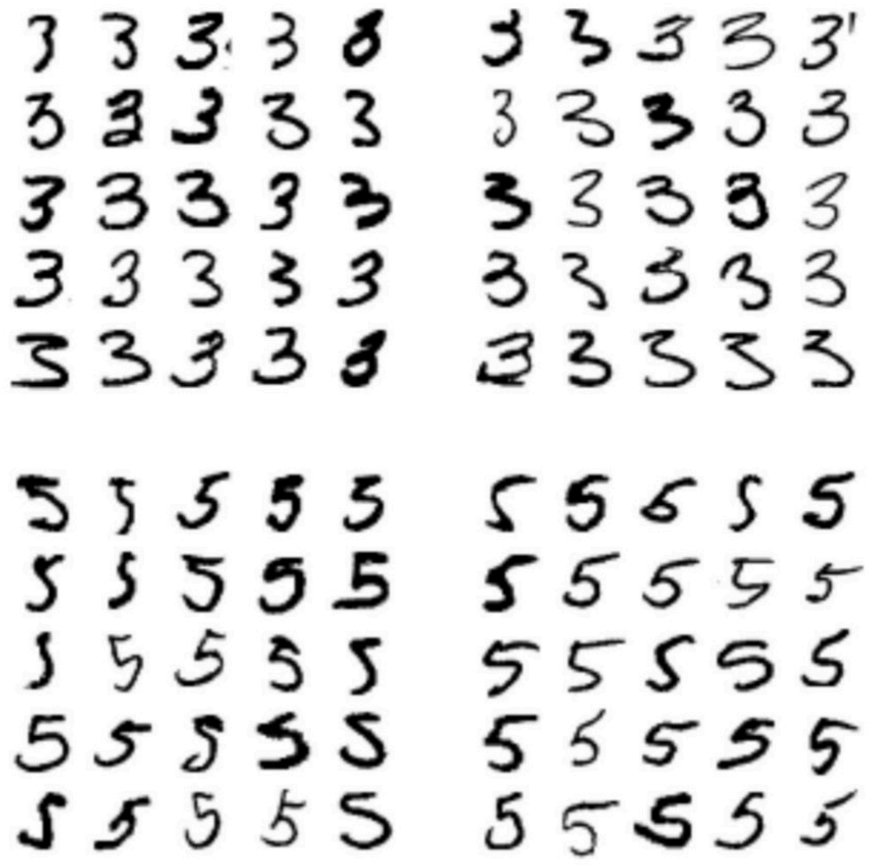
```
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
save_fig("confusion_matrix_errors_plot", tight_layout=False)
plt.show()
```

```
Saving figure confusion_matrix_errors_plot
```



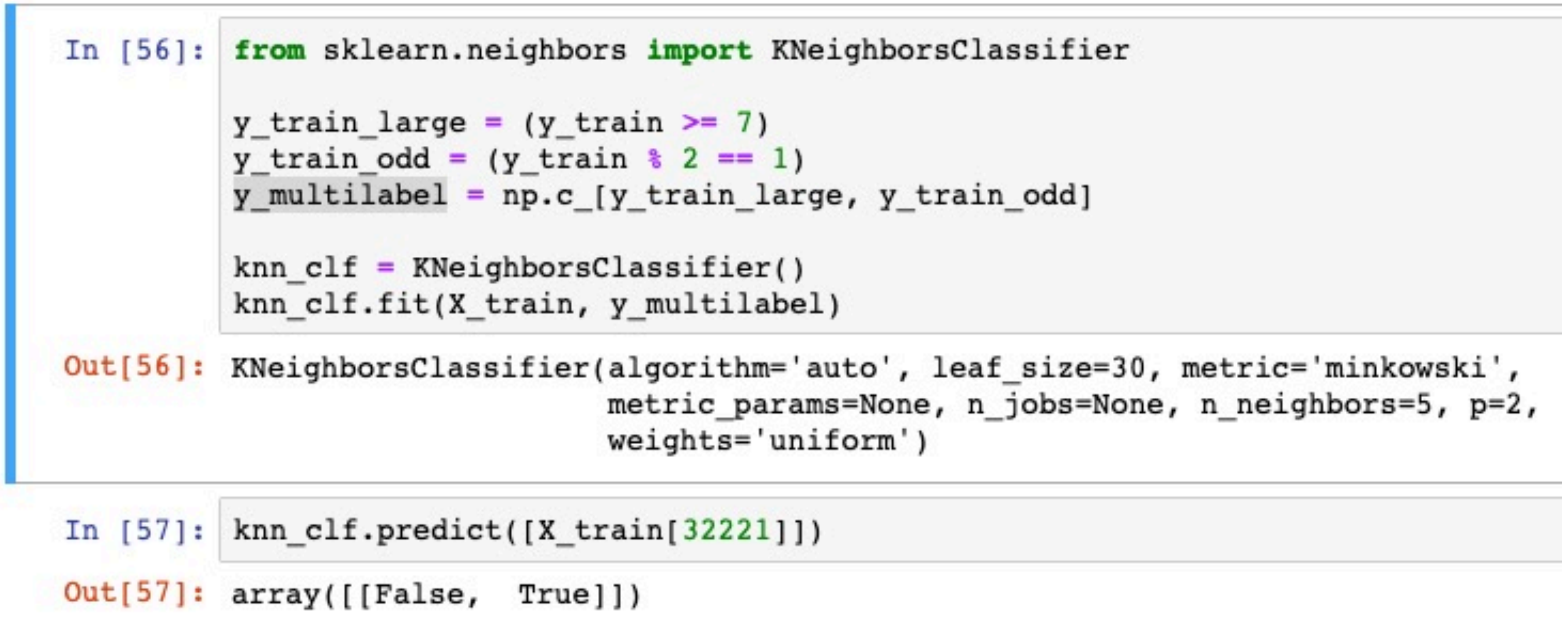
Error Analysis

Analyzing individual errors can also be a good way to gain insights on what your classifier is doing and why it is failing, but it is more difficult and time-consuming. For example, let’ s plot examples of 3s and 5s



Multilabel Classification

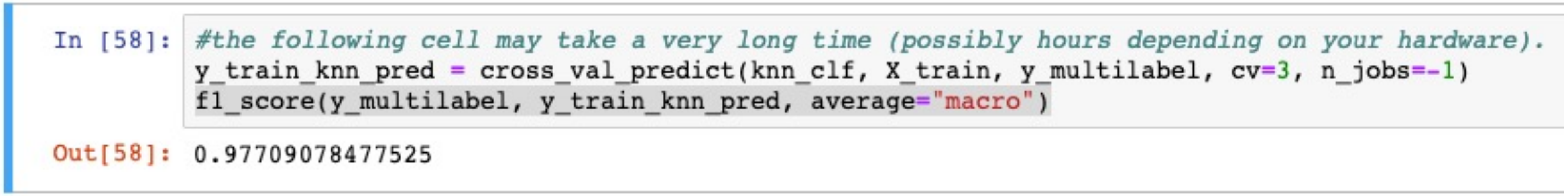
A Until now each instance has always been assigned to just one class. In some cases you may want your classifier to output multiple classes for each instance.



```
In [56]: from sklearn.neighbors import KNeighborsClassifier

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

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

Out[56]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                              weights='uniform')

In [57]: knn_clf.predict([X_train[32221]])

Out[57]: array([[False,  True]])
```

Multilabel Classification

There are many ways to evaluate a multilabel classifier, and selecting the right metric really depends on your project. For example, one approach is to measure the F1 score for each individual label.



```
In [58]: #the following cell may take a very long time (possibly hours depending on your hardware).
         y_train_knn_pred = cross_val_predict(knn_clf, X_train, y_multilabel, cv=3, n_jobs=-1)
         f1_score(y_multilabel, y_train_knn_pred, average="macro")

Out[58]: 0.977090784775 25
```

Performance Measures

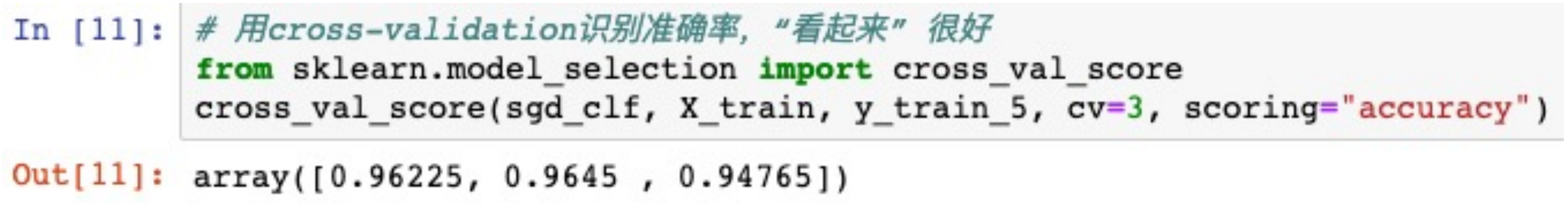
Evaluating a classifier is often significantly trickier than evaluating a regressor, so we will spend a large part of this chapter on this topic. There are many performance measures available, so grab another coffee and get ready to learn many new concepts.

Let’ s take the trip with a binary classifier “5-detector”

Performance Measures

Cross-Validation to evaluate a model:

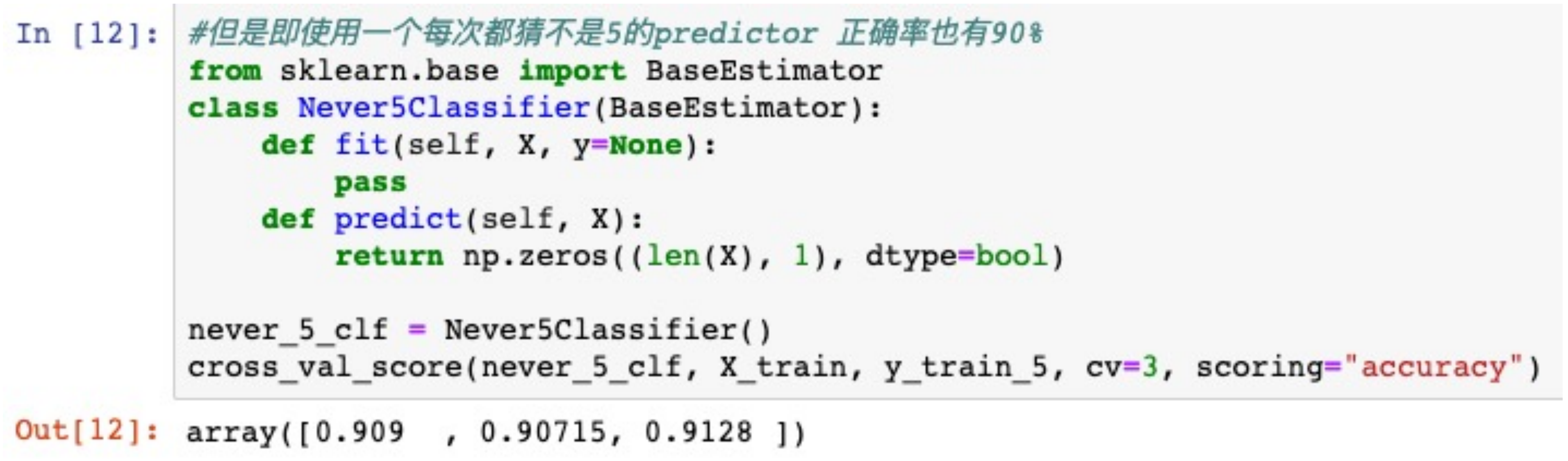
1. K-fold cross-validation means splitting the training set into K-folds
2. making predictions and evaluating them on each fold using a model trained on the remaining folds
3. Score = count_correct/count_instances



```
In [11]: # 用cross-validation识别准确率, “看起来” 很好
         from sklearn.model_selection import cross_val_score
         cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")

Out[11]: array([0.96225, 0.9645 , 0.94765])
```

Performance Measures

dumb classifier that just classifies every single image in the “not-5” class

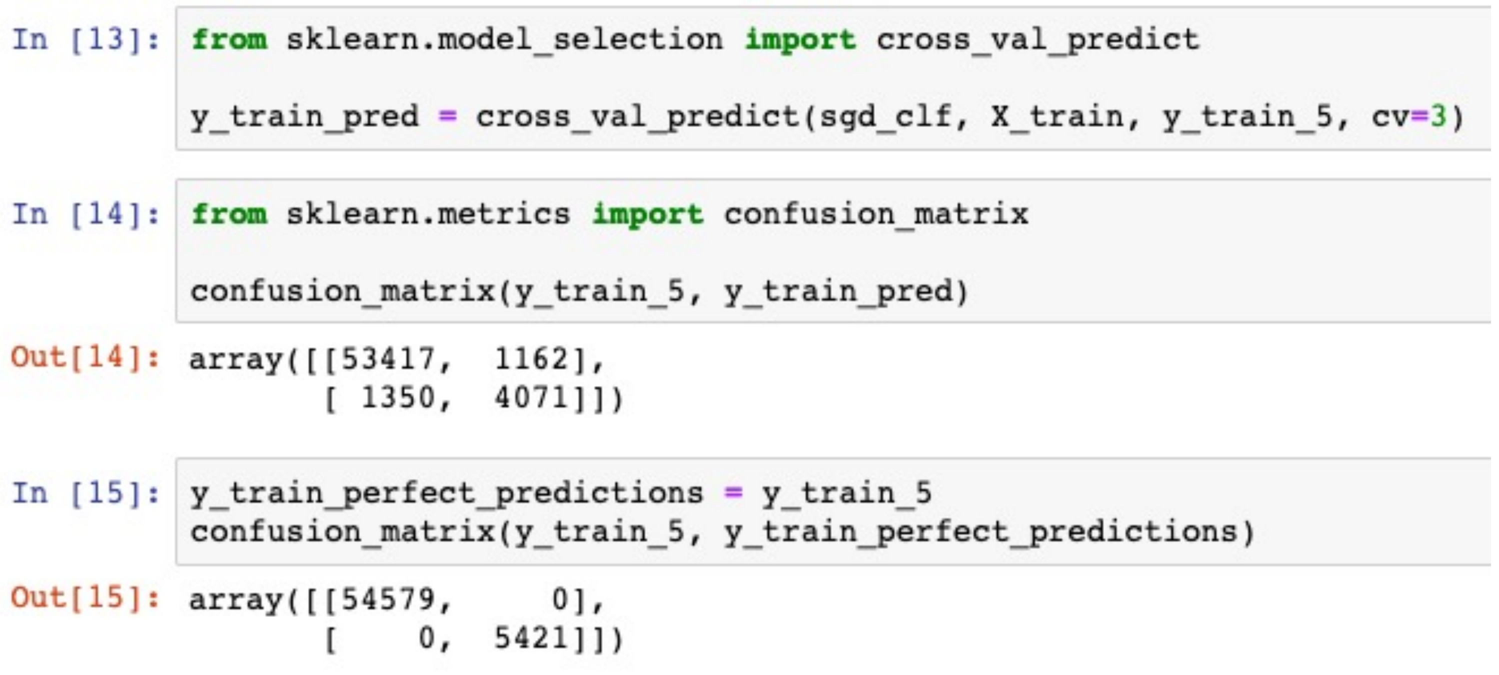


```
In [12]: #但是即使用一个每次都猜不是5的predictor 正确率也有90%
         from sklearn.base import BaseEstimator
         class Never5Classifier(BaseEstimator):
             def fit(self, X, y=None):
                 pass
             def predict(self, X):
                 return np.zeros((len(X), 1), dtype=bool)

         never_5_clf = Never5Classifier()
         cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
Out[12]: array([0.909  , 0.90715, 0.9128 ])
```

Performance Measures

Confusion Matrix: Each row in a confusion matrix represents an actual class, while each column represents a predicted class.

The first row of this matrix considers non-5 images (the negative class): 53,272 of them were correctly classified as non-5s (they are called true negatives)



```
In [13]: from sklearn.model_selection import cross_val_predict

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

In [14]: from sklearn.metrics import confusion_matrix

         confusion_matrix(y_train_5, y_train_pred)

Out[14]: array([[53417,  1162],
                [ 1350,  4071]])

In [15]: y_train_perfect_predictions = y_train_5
         confusion_matrix(y_train_5, y_train_perfect_predictions)

Out[15]: array([[54579,     0],
                [    0,  5421]])
```

Performance Measures–P&R

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.

Equation 3-2. Recall

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

Performance Measures–P&R

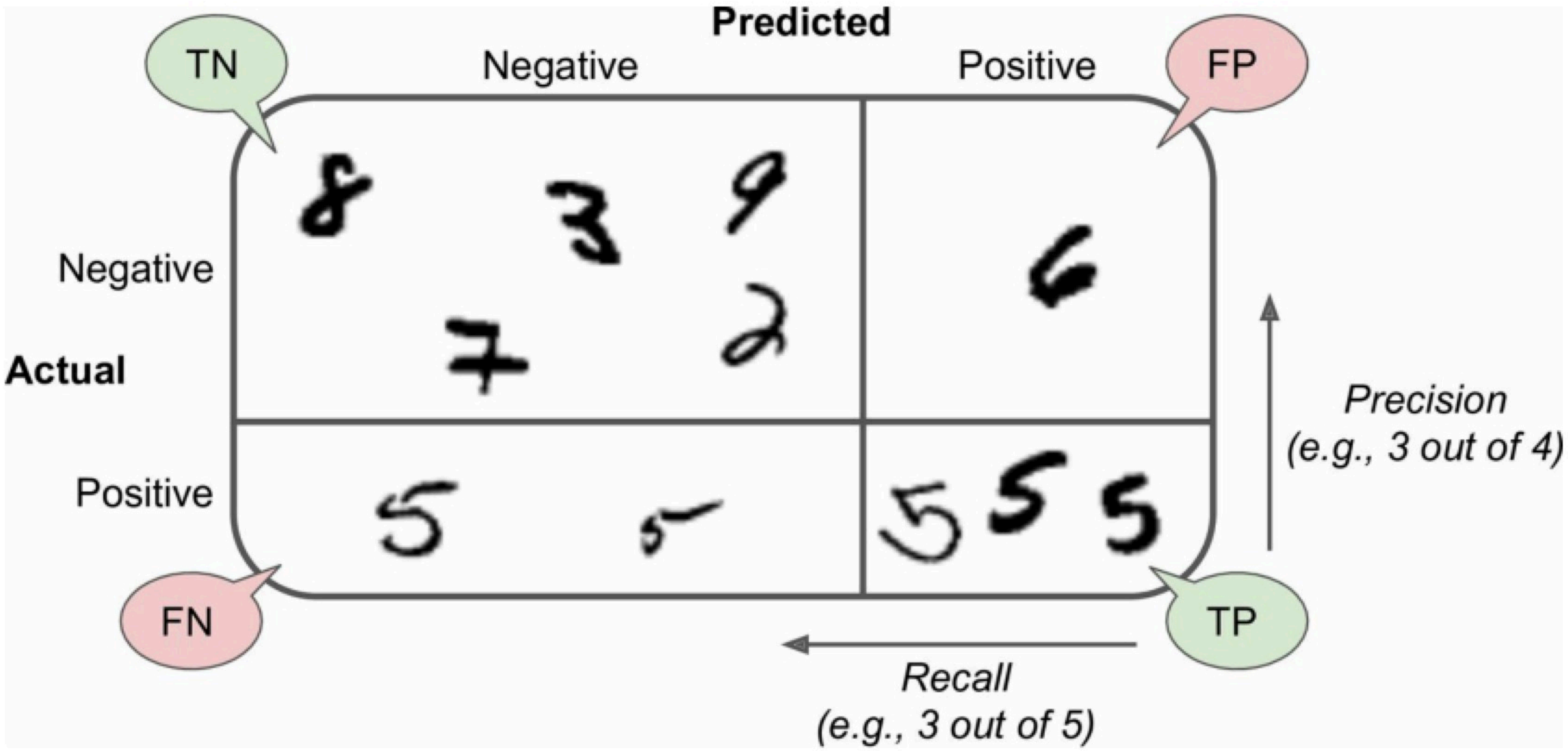
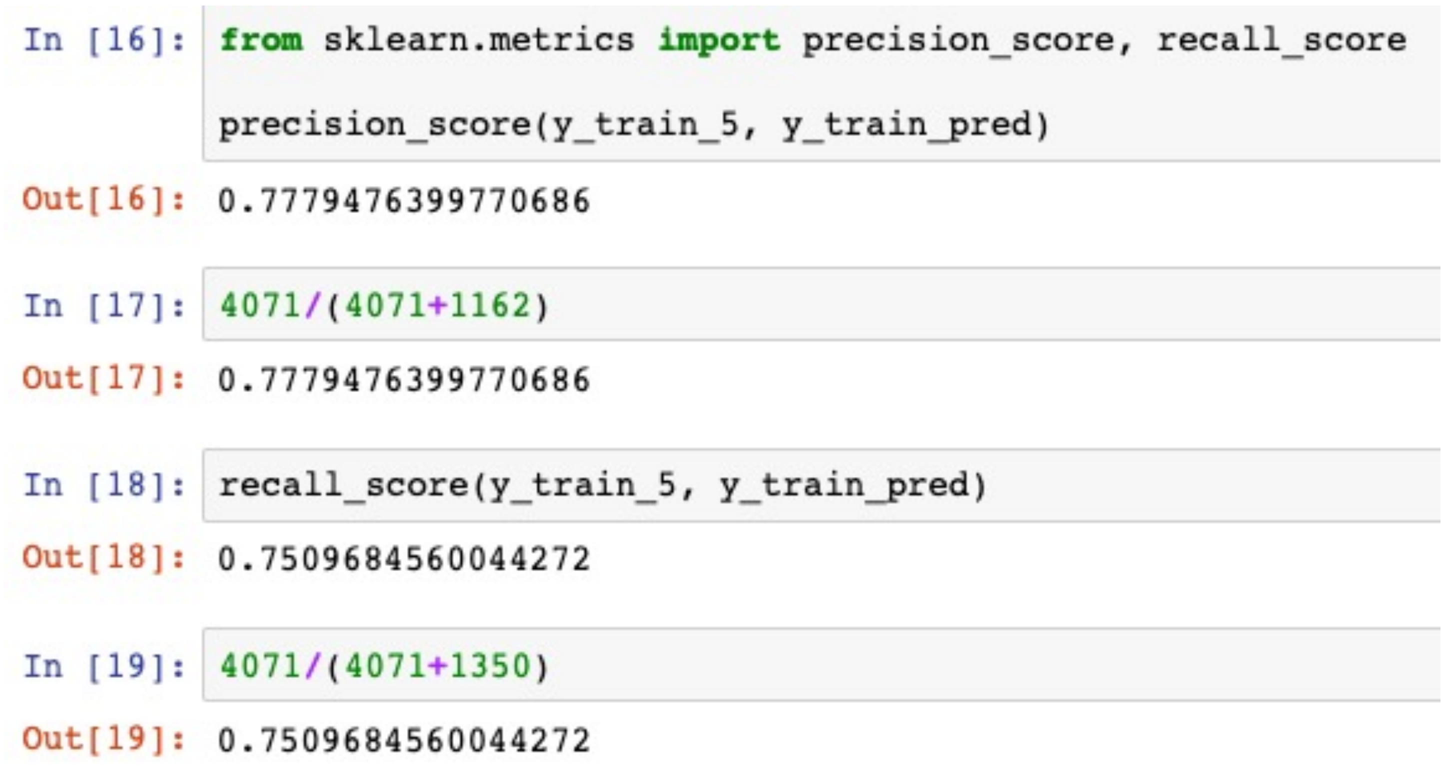


Figure 3-2. An illustrated confusion matrix

Performance Measures–P&R

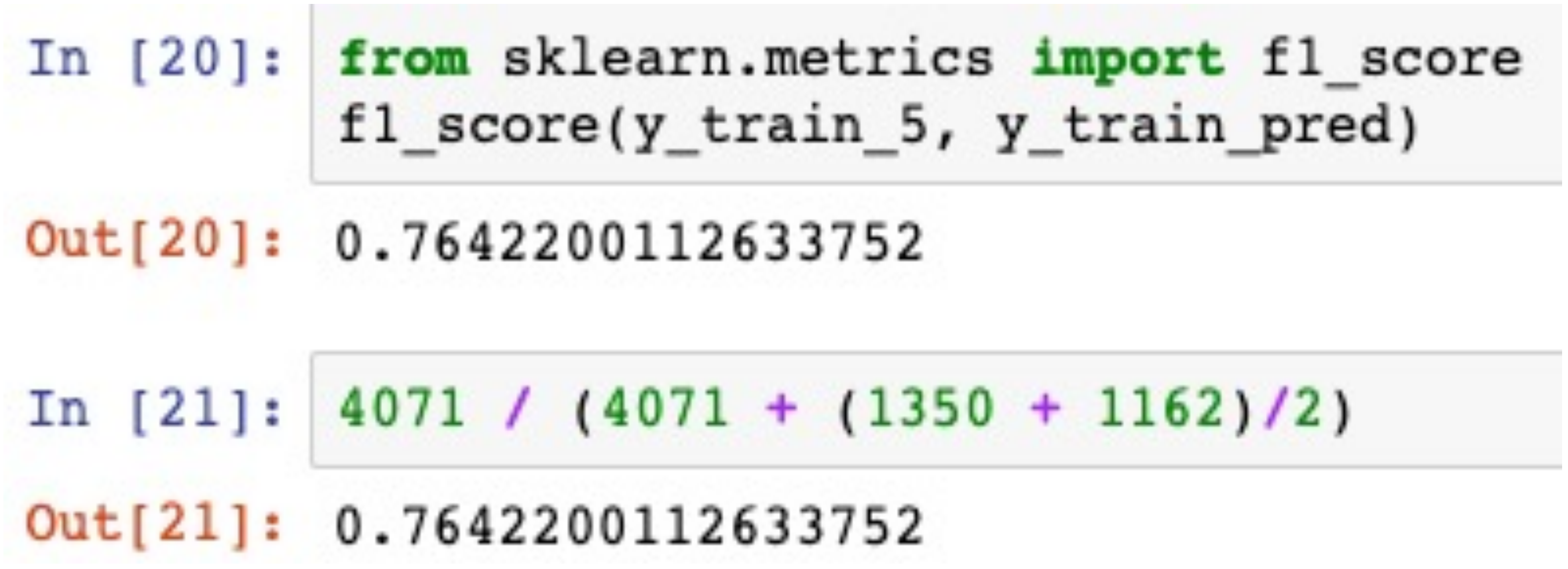
Scikit-Learn provides several functions to compute classifier metrics, including precision and recall



```
In [16]: from sklearn.metrics import precision_score, recall_score

         precision_score(y_train_5, y_train_pred)

Out[16]: 0.7779476399770686

In [17]: 4071/(4071+1162)

Out[17]: 0.7779476399770686

In [18]: recall_score(y_train_5, y_train_pred)

Out[18]: 0.7509684560044272

In [19]: 4071/(4071+1350)

Out[19]: 0.7509684560044272
```

Performance Measures

It is often convenient to combine precision and recall into a single metric called the F1 score, in particular if you need a simple way to compare two classifiers



```
In [20]: from sklearn.metrics import f1_score
         f1_score(y_train_5, y_train_pred)

Out[20]: 0.7642200112633752

In [21]: 4071 / (4071 + (1350 + 1162)/2)

Out[21]: 0.7642200112633752
```

Performance Measures

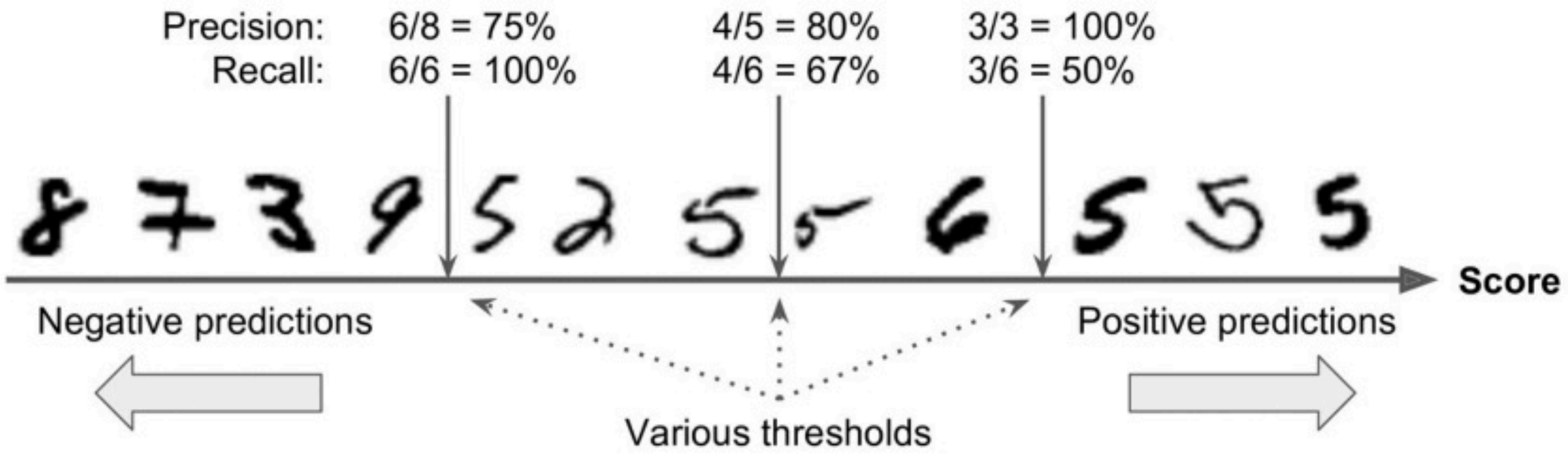
The F1 score favors classifiers that have similar precision and recall. This is not always what you want.

suppose you train a classifier to detect shoplifters on surveillance images: it is probably fine if your classifier has only 30% precision as long as it has 99% recall (sure, the security guards will get a few false alerts, but almost all shoplifters will get caught).

Unfortunately, you can’t have it both ways: increasing precision reduces recall, and vice versa. This is called the precision/recall tradeoff

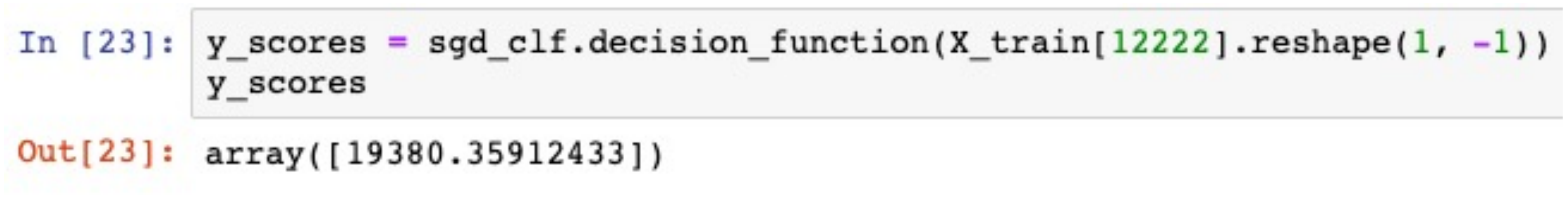
Performance Measures-tradeoff

To understand this tradeoff, let’ s look at how the SGDClassifier makes its classification decisions. For each instance, it computes a score based on a decision function, and if that score is greater than a threshold, it assigns the instance to the positive class, or else it assigns it to the negative class.



Performance Measures-tradeoff

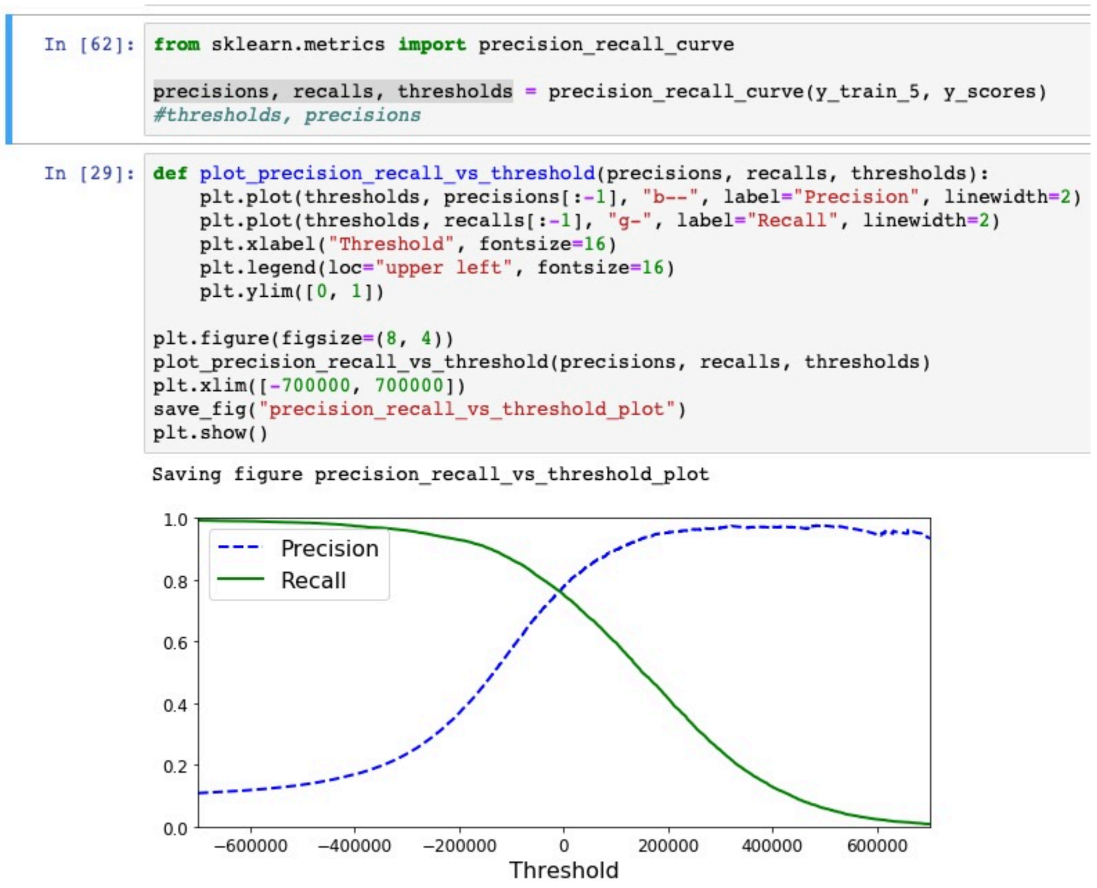
Scikit-Learn *decision_function()* method, which returns a score for each instance



```
In [23]: y_scores = sgd_clf.decision_function(X_train[12222].reshape(1, -1))
         y_scores

Out[23]: array([19380.35912433])
```

Performance Measures-tradeoff

get the scores of all instances in the training set
compute precision and recall for all possible thresholds using the precision_recall_curve()

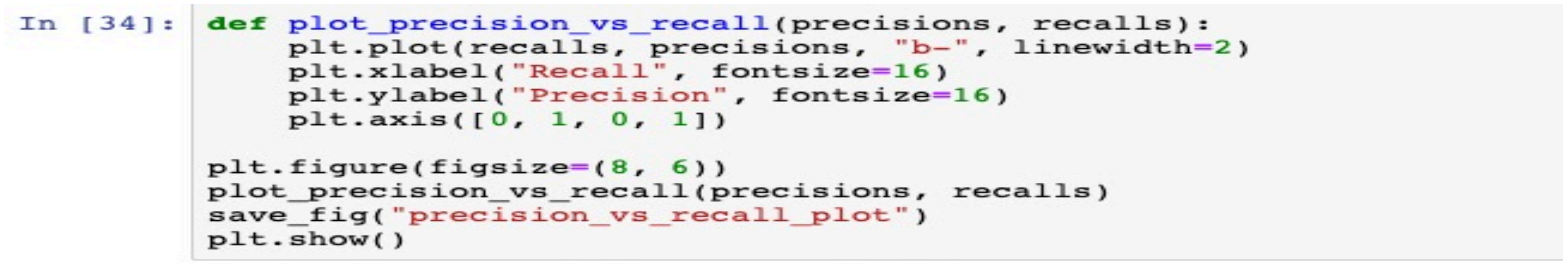


```
In [62]: from sklearn.metrics import precision_recall_curve

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

```
In [29]: def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
             plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
             plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
             plt.xlabel("Threshold", fontsize=16)
             plt.legend(loc="upper left", fontsize=16)
             plt.ylim([0, 1])

         plt.figure(figsize=(8, 4))
         plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
         plt.xlim([-700000, 700000])
         save_fig("precision_recall_vs_threshold_plot")
         plt.show()
```

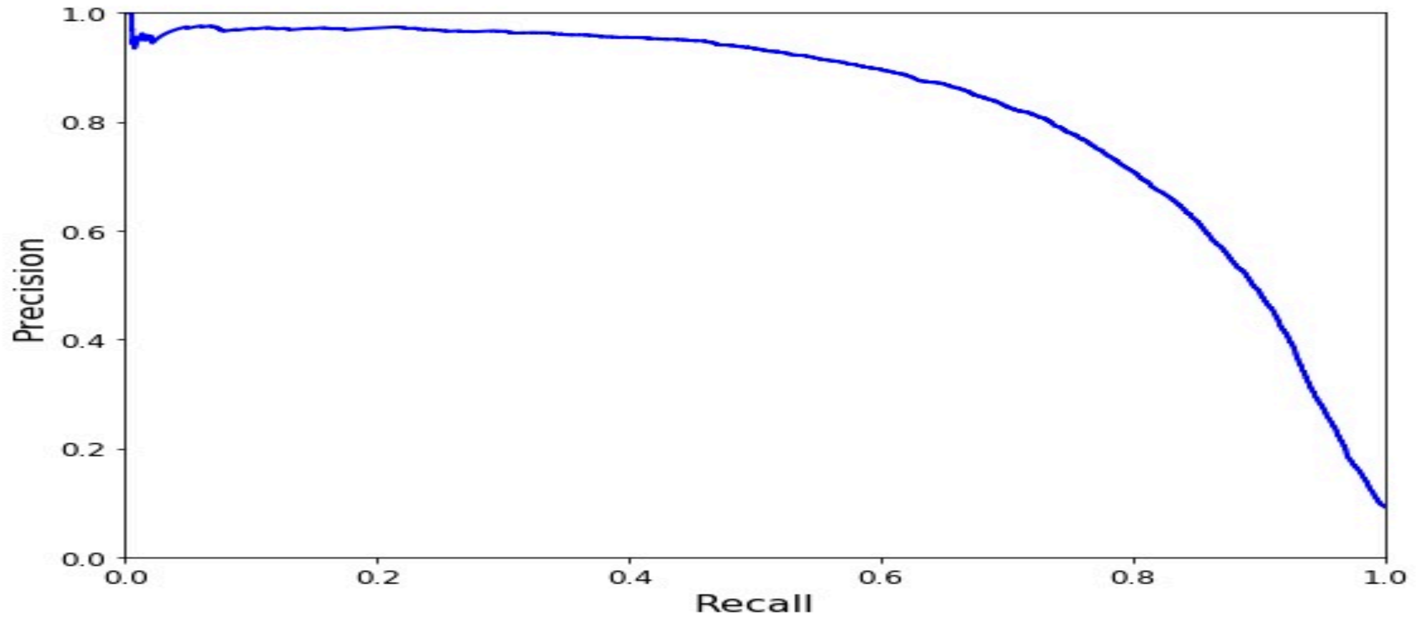
```
Saving figure precision_recall_vs_threshold_plot
```

Performance Measures-tradeoff

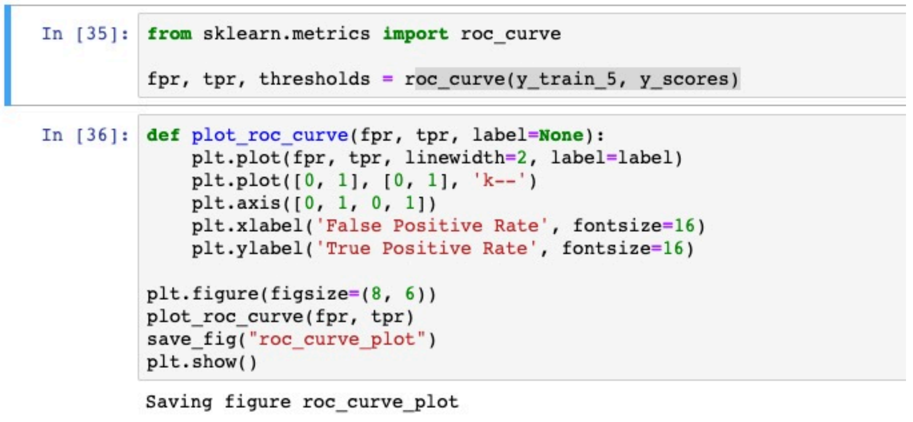


```
In [34]: def plot_precision_vs_recall(precisions, recalls):
             plt.plot(recalls, precisions, "b-", linewidth=2)
             plt.xlabel("Recall", fontsize=16)
             plt.ylabel("Precision", fontsize=16)
             plt.axis([0, 1, 0, 1])

         plt.figure(figsize=(8, 6))
         plot_precision_vs_recall(precisions, recalls)
         save_fig("precision_vs_recall_plot")
         plt.show()
```

```
Saving figure precision_vs_recall_plot
```



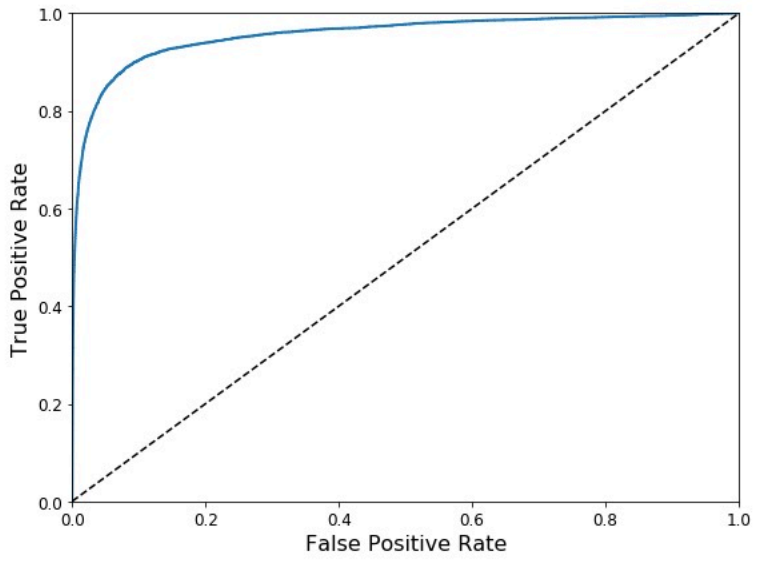
Performance Measures-ROC



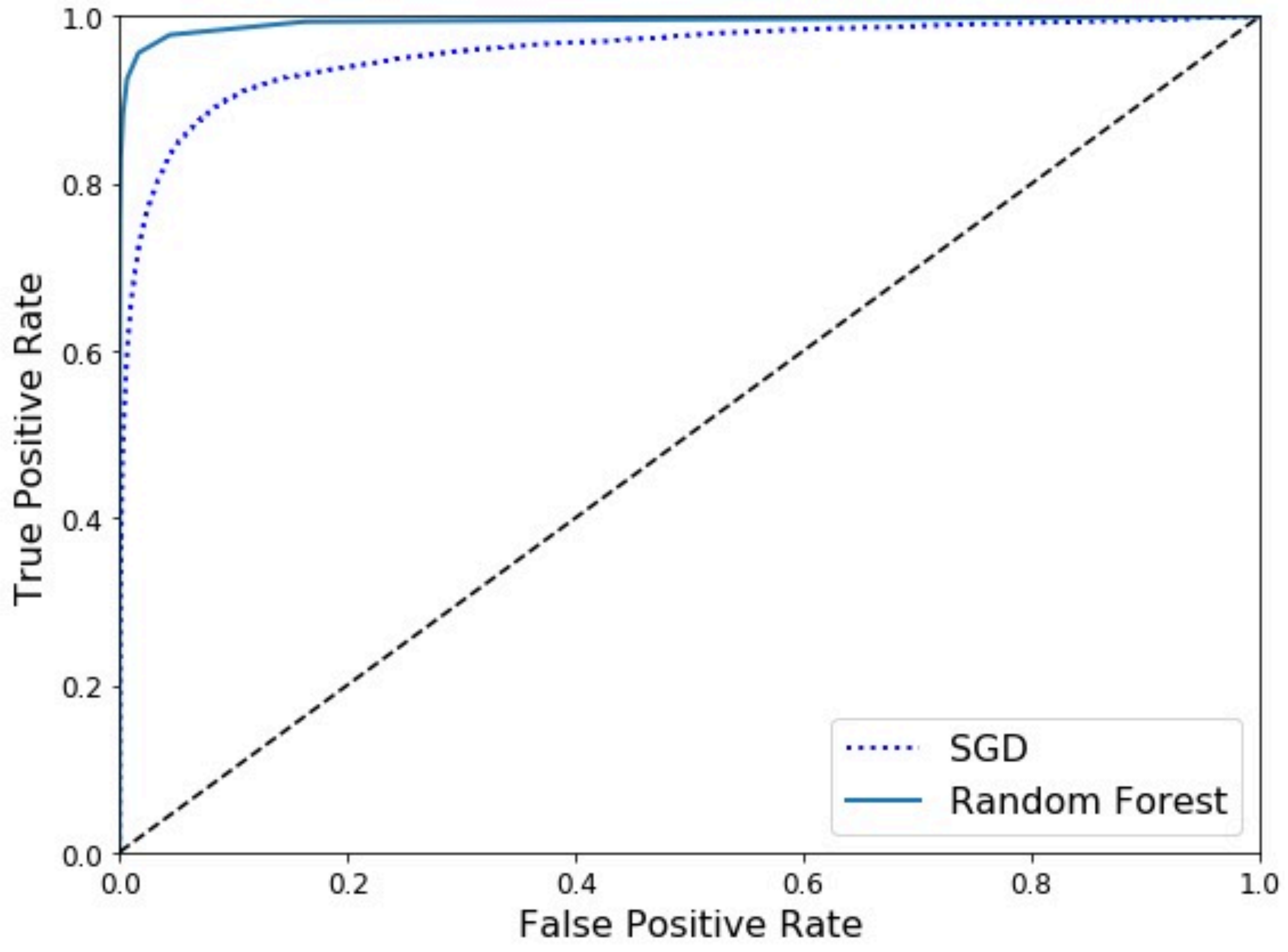
```
In [35]: from sklearn.metrics import roc_curve

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

```
In [36]: def plot_roc_curve(fpr, tpr, label=None):
             plt.plot(fpr, tpr, linewidth=2, label=label)
             plt.plot([0, 1], [0, 1], 'k--')
             plt.axis([0, 1, 0, 1])
             plt.xlabel('False Positive Rate', fontsize=16)
             plt.ylabel('True Positive Rate', fontsize=16)

         plt.figure(figsize=(8, 6))
         plot_roc_curve(fpr, tpr)
         save_fig("roc_curve_plot")
         plt.show()
```

```
Saving figure roc_curve_plot
```



Performance Measures-ROC



SAIF

Shanghai Advanced Institute of Finance

上海高级金融学院