

这种计算哈希值方法的原理时啥？不太明白，我学过的hush值运算是在网络安全中，用来验证信息的完整性的。

Scikit-Learn（机器学习的框架之一）提供了一些功能，以各种方式将数据集分割成多个子集。（最简单的函数是train\_test\_split()，它的功能与函数split\_train\_test()基本相同，但有两个附加特性。  
首先，有一个random\_state参数允许您设置随机生成器种子。其次，你可以给它传递多个数据集相同的行数，它会在相同的索引上分割它们(这是非常有用的）



分类器选择：随机分级下降(SGD)分类器，使用Scikit-Learn的SGDClassifier类。这种分类器的优点是能够有效地处理非常大的数据集。

评估模型的一个好方法是使用交叉验证：

from sklearn.model\_selection import StratifiedKFold

from sklearn.base import clone

skfolds = StratifiedKFold(n\_splits=3, random\_state=42)

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))

备注：

在每次迭代中，代码创建分类器的一个克隆，在训练折叠上训练克隆，并在测试折叠上进行预测。然后计算正确预测的数量并输出正确预测的比率。

使用cross\_val\_score()函数来评估SGDClassifier模型，使用K-fold三次交叉验证。

>>> from sklearn.model\_selection import cross\_val\_score

>>> cross\_val\_score(sgd\_clf, X\_train, y\_train\_5, cv=3, scoring="accuracy")

array([0.96355, 0.93795, 0.95615])

Confusion Matrix

用来评估分类器性能

如何评估？方法和标准是什么？

from sklearn.model\_selection import cross\_val\_predict

y\_train\_pred = cross\_val\_predict(sgd\_clf, X\_train, y\_train\_5, cv=3)

它返回的不是评估分数，而是每次测试折叠的预估值。

混和矩阵中的每一行代表一个实际的类，而每一列代表一个预测的类。P91

TP是真阳性的数量，FP是假阳性的数量。

（TP、True Positive   真阳性：预测为正，实际也为正

FP、False Positive  假阳性：预测为正，实际为负

FN、False Negative 假阴性：预测与负、实际为正

TN、True Negative 真阴性：预测为负、实际也为负。

）

\*F1得分是精度和召回率的调和平均值，计算F1 score，只需调用f1\_score()函数:

>>> from sklearn.metrics import f1\_score

>>> f1\_score(y\_train\_5, y\_train\_pred)

Precision/Recall Trade-of（P93）

理解例子：

想象一下, 你的女朋友在过去的十年里, 每年都给你生日惊喜… 然而今年,

She: Sweetie, do you remember all your birthday surprises from me?

You: O\_\_O …

The simple question makes your life in danger.

为了活下去, 你需要回忆 (Recall) 那些年的生日惊喜. 所以, 得到 Recall 公式:

比如你10件事中你想起了7件, 那么 $Recall = 7/10 = 0.7$

以上例子默认你想起的都是正确的, 但是你可能会记错, 比如她并没有送过你机械键盘, 是你自己买的, 或者你想靠运气多瞎猜几个, 那么就得到了 Precision 公式, 表示你的准确率:

例如你想起了8件事, 但是只有4件事正确的, 这时 $Precision = 4 /8 = 0.5$

作为你的女朋友, 她不希望看到:

你记得不对又不全 (Low Precision, Low Recall): 不需要解释了吧

你记得对但是不全 (High Precision, Low Recall): 你很诚实, 一说一个准, 然而不超过3件就说不下去了

你记得全但是不对 (Low Precision, High Recall): 听起来有点绕, 但其实就是你是一个不老实的人, 说了一堆常见的生日惊喜, 然后猜中了不少, 当然猜错的更多

记得对又记得全自然最好, 但毕竟人无完人, 于是她就得权衡你的用心程度和诚实度, 也就是所谓的 Precision & Recall Trade-off 了。

The receiver operating characteristic (ROC)曲线是另一个常用的工具与二进制分类器。

要绘制ROC曲线，首先使用roc\_curve()函数计算不同阈值的TPR和FPR：

from sklearn.metrics import roc\_curve

fpr, tpr, thresholds = roc\_curve(y\_train\_5, y\_scores)

然后可以使用Matplotlib对TPR绘制FPR：

def plot\_roc\_curve(fpr, tpr, label=None):

plt.plot(fpr, tpr, linewidth=2, label=label)

plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal

[...] # Add axis labels and grid

plot\_roc\_curve(fpr, tpr)

plt.show()

\*这同样需要权衡:召回率(TPR)越高，误报率就越多。

**Multiclass Classification**

1. 创建一个系统的一种方法，可以将数字图像分成10个类(从0到  
   9)是训练10个二进制分类器，每个数字一个(0检波器，1检波器，2检波器，等等)。然后，当你想要对一个图像进行分类时，你从每个分类器那里得到该图像的决策分数，然后选择出分类器的类别——即得分最高的类别。这称为“一人对众”(one-versus-the-rest, OvR)策略(也称为“一人对众”)。
2. 另一种策略是为每一对数字训练一个二进制分类器:一个用来区分0和1，另一个用来区分0和2s，另一个用来区分1和2s，以此类推。  
   这就是所谓的一对一(OvO)战略。如果有N个类，则需要训练N×(N - 1) / 2个分类器。对于MNIST问题，这意味着要训练45个二进制分类器!当您想要对一个图像进行分类时，您必须通过所有45个分类器运行该图像，以查看哪个类赢得最多的决斗。

**Error Analysis （103）**

首先，看看混淆矩阵。您需要使用cross\_val\_predict()函数进行预测，然后调用confusion\_matrix()函数（之前有学到）

CH3 Training Models

针对线性回归模型的训练方法：

线性回归使用SVD

梯度下降法  
梯度下降是一种通用的优化算法，能够找到最优解决广泛的问题。梯度下降的一般思想是迭代地调整参数，以使代价函数最小化。（变体：批GD，迷你分批GD，和随机GD。）

（梯度下降的一个重要参数是步长，由学习速率超参数决定。如果学习率太小，那么算法将不得不经过多次迭代来收敛，这将花费很长时间；另一方面，如果学习率太高，你可能会跳过峰值，最终到达另一边，甚至可能比以前更高。这可能导致算法发散，数值越来越大，无法找到好的解决方案）

Fortunately, the MSE cost function for a Linear Regression model happens to be a

convex function, which means that if you pick any two points on the curve, the line

segment joining them never crosses the curve. This implies that there are no local

minima, just one global minimum. It is also a continuous function with a slope that

never changes abruptly.3 These two facts have a great consequence: Gradient Descent

is guaranteed to approach arbitrarily close the global minimum (if you wait long

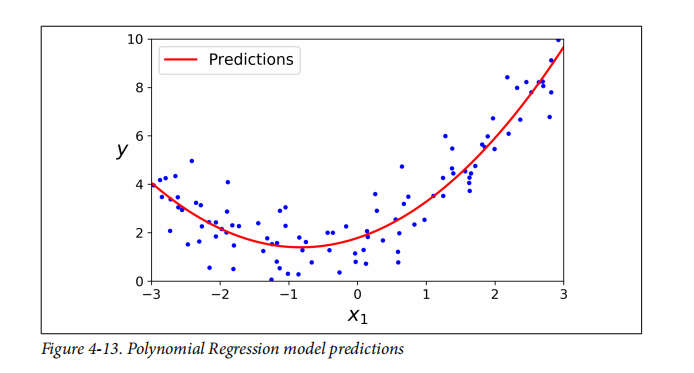
enough and if the learning rate is not too high).

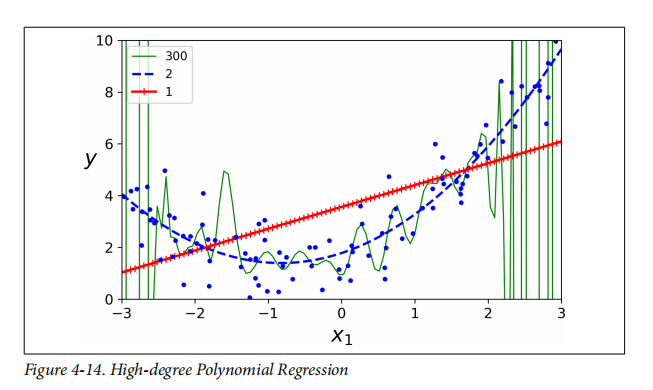
**Mini-batch Gradient Descent**

**Polynomial Regression**

当数据比直线更复杂的情况，使用Scikit-Learn的Poly nomialFeatures类来转换我们的训练数据，在训练集中添加每个特征的平方(二阶多项式)作为一个新的特征(在这种情况下只有一个特征):

**Learning Curves**

对比



模型泛化误差：

偏见  
这部分的泛化误差是由错误的假设造成的，例如，当数据实际上是二次的时候，却认为它是线性的。一个高偏差的模型很可能不适合训练数据  
方差  
这部分是由于模型对训练数据中的小变化过于敏感。一个有许多自由度的模型(例如一个高度的pol - yial模型)很可能有高的方差，从而使训练数据过拟合。  
不可约的错误  
这一部分是由于数据本身的噪声。减少这部分错误的唯一方法是清理数据

减少模型过度拟合的方法：

1. Regularized Linear Models
2. Ridge Regression
3. Lasso Regression

套索回归的一个重要特征是，它倾向于消除最不重要特征的权重。，将其设置为零)

1. Elastic Net

Elastic Net是Ridge Regression和套索回归的中间地带。正则化项是Ridge和Lasso正则化项的简单混合，可以控制混合比例r。当r = 0时，弹性网等价于Ridge，当r = 1时，相当于Lasso回归

\*不需要任何正则化的时候，可以选择普通的线性回归

1. Early Stopping

一种非常不同的方法来规范迭代学习算法，如梯度  
下降是在验证错误达到最小值时停止训练。这就是所谓的提前停止。

1. Logistic Regression（0和1）
2. Estimating Probabilities
3. Training and Cost Function
4. Decision Boundaries