Chapter 3 분류

MNIST

머신러닝의 hello world 데이터 셋..10만개의 이미지로 구성되어있음! openml에서 다운로드 받아야 한다. 픽셀 데이터이며 정수 값이 들어가 있다.

In [1]:
```
# mnist dataset openml에서 내려받기
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)        # as_frame = True (데이터 프레임으로 받겠다~ 지금은 아니니 frame = null)
                                                    #mnist_784라는 아이디를 부여하며 버전1에 해당하는 것을 받겠다.

# keys 조회 (딕셔너리 스타일로 값을 가진 bunch 스타일 객체)
mnist.keys()
```

Out[1]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_names', 'DESCR', 'details', 'url'])
```

사이킷 런에서 읽어들인 데이터 셋들은 비슷한 딕셔너리 구조를 가지고 있다.

- 데이터셋을 설명하는 DESCR키
- 샘플이 하나의 행, 특성이 하나의 열로 구성된 배열을 가진 DATA 키
- 레이블배열을 담은 TARGET 키

In [14]:
```
mnist["url"]   # 얘 데려온 url
```

Out[14]:
```
'https://www.openml.org/d/554'
```

In [2]:
```
# 배열 살피기 (2차원)
# X와 y에 data와 target값을 받고 X먼저 조회하기

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

# 행이 70000개고 열이 784개이다~
# 70000개 샘플이 있고 784개 특성이 있구나~(픽셀이 28*28이라..)
```

Out[2]:
```
(70000, 784)
```

In [3]:
```
y.shape

# 이미지가 70000개 있고 이미지의 특성이 784개 있다는 뜻이다.
# 이미지의 픽셀이 28*28픽셀이기 때문이다.
```

Out[3]:
```
(70000,)
```

In [15]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt

some_digit = X[0]          # 이미지 한 놈을 some_digit에다 넣어주고...(얘가 numpy인데 바꿔줘야지)
some_digit_image = some_digit.reshape(28, 28)     # 샘플의 특성 벡터를 추출해 28*28 배열로 크기 바꾸기

plt.imshow(some_digit_image, cmap = "binary")     # imshow로 조회하기  # binary(보기 편하게 흑백값 반전)
plt.axis("off")
plt.show()
```

In [5]:
```
y[0]

# y이 첫번째 레이블(클래스 확인~)
# 실제 데이터 특성도 5 (레이블)
```

Out[5]: '5'

In [6]:
```
import numpy as np

y = y.astype(np.uint8)  # 정수 배열로 바꿔서 확인할 것이다
```

In [18]:
```
def plot_digit(data):
    image = data.reshape(28, 28)
    plt. imshow(image, cmap=mpl.cm.binary,
                interpolation="nearest")
    plt. axis("off")
```

In [22]:
```
# 숫자 그림을 위한 추가 함수
def plot_digits(instances, images_per_row=10, **options):
    size=28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size, size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size*n_empty)))
    for row in range(n_rows) :
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap=mpl.cm.binary, **options)
    plt.axis("off")
```

In [23]:
```
plt.figure(figsize=(9,9))
example_images = X[:100]
plot_digits(example_images, images_per_row=10)
plt.show()
```

In [24]:
```
y[0]  # integer로 잘 바뀌었군~
```

Out[24]:
```
5
```

In [7]:
```
# 훈련셋 확인해서 스플릿~
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]

# 훈련 세트는 이미 섞여 있기 때문에 모든 교차 검증 폴드가 비슷해진다.
# 어떤 학습 알고리즘은 훈련 샘플 순서에 민감해서 많은 비슷한 샘플이 있을 경우 성능이 나빠진다.
# 데이터셋을 섞으면 이런 문제를 방지할 수 있다.
# 그러나 주식 가격 같은 시계열 데이터는 오히려 섞지 않는 것이 나을 수 있다.

#SGDClassifier, SGDRegressor는 기본적으로 에포크(max_iter)마다 데이터를 다시 섞는다.
```

3.2 이진 분류기 훈련

숫자 '5' 가 있다면 '5-감지기'와 '5 아님' 두 개의 클래스를 구분할 수 있는 것이 이진 분류기의 예이다.

In [25]:
```
y_train_5 = (y_train == 5)       # 5만 true이고 나머지는 false (이진분류기 만들기~)
y_test_5 = (y_test == 5)
```

SGDClassifier

SGD : 확률적 경사 하강법 (Stochastic Gradient Descent)

- 매우 큰 데이터셋도 효율적으로 처리한다.
- 한번에 하나씩 훈련 샘플을 독립적으로 처리한다.
- 온라인 학습에 잘 들어맞는다.

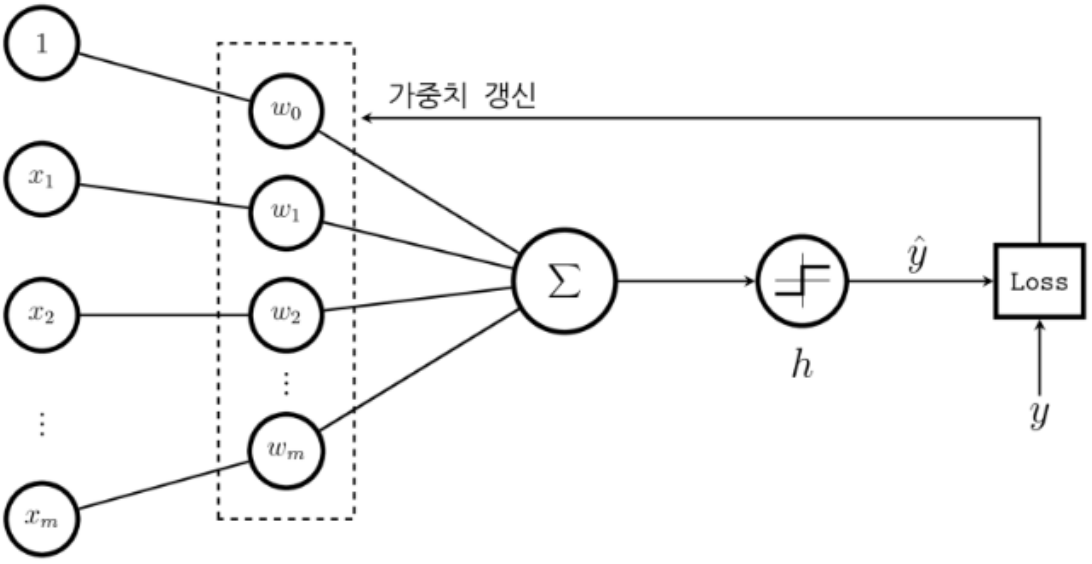
In [26]:
```
# SGDClassifier 만들고 적용시켜보기
# 이거 쓰면서 여러가지 분류 모델 만들 수 있땅

from sklearn. linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state = 42)      # sgd_clf는 SGDClassifier로 만들고
sgd_clf.fit(X_train, y_train_5)                 # .fit 적용해서 X_train, y_train_5를 사용하는 함수로 만들어줘

# 아까 만든 분류기 집어넣었다
# SGDClassifier는 훈련하는데에 무작위성을 사용한다. 동일한 결과를 재현하고 싶다면 random_state매개변수를 지정한다.
```

Out[26]:
```
SGDClassifier(random_state=42)
```

In [10]:
```
sgd_clf.predict([some_digit])

# 아까 some_digit는 1차원 배열인데 기본적으로 함수가 2차원 배열로 돌아가므로 [리스트] 로 감싸준다.
# 아까 샘플 넣고 돌리면 true라고 잘 예측 함
# true 값이 나오므로 분류기는 이 값이 5를 나타낸다고 추측한 것!
```

Out[10]:
```
array([ True])
```

퍼셉트론



퍼셉트론(perceptron)은 가장 오래되고 단순한 형태의 판별함수기반 분류모형 중 하나이다. 퍼셉트론은 입력 x=(1,x1,⋯,xm) 에 대해 1 또는 -1의 값을 가지는 y 를 출력하는 비선형 함수이다. 1을 포함하는 입력 요소 xi 에 대해 가중치 wi 를 곱한 값 a=wTx 을 활성화값(activations)이라고 하며 이 값이 판별함수의 역할을 한다.

a=wTx

판별 함수 값이 활성화함수(activation function) h(a)를 지나면 분류 결과를 나타내는 출력 y^가 생성된다.

y^=h(wTx)

퍼셉트론의 활성화 함수는 부호함수(sign function) 또는 단위계단함수(Heaviside step function)라고 부르는 함수이다.

h(a)={(−1,1),(a<0,a≥0)

퍼셉트론 손실함수

x, y = 독립변수, 종속변수 w = 가중치(예측 오차 최소화하는) L = 전체 예측 오차 (가중치값에 따라 달라지는)

다음과 같이 N 개의 학습용 데이터가 있다고 하자.

(x1,y1),(x2,y2),…,(xi,yi),…,(xN,yN)

퍼셉트론은 독립변수 x 로부터 종속변수 y 를 예측하는 예측 모형이므로 모든 학습 데이터에 대해 예측 오차를 최소화하는 가중치 w 를 계산해야 한다. 가중치 w 에 따라 달라지는 전체 예측 오차 L 는 i 번째 개별 데이터에 대한 손실함수 Li(y^i,yi) 의 합으로 표현할 수 있다.

$$L = \sum_{i=1}^{N} L_i(y_i, \hat{y}_i)$$

손실 Li(yi,y^i) 는 실제값 y 와 예측값 y^ 의 차이를 나타내는 함수이다. 회귀 분석에서는 L(y^,y)=−(y−y^)2 과 같은 손실함수를 많이 사용하였지만 퍼셉트론의 경우에는 다음과 같은 손실 함수를 사용한다. 이를 제로-원 손실함수(zero-one loss function)이라고 한다.

$$L_i(y_i, \hat{y}_i) = \max(0, -y_i \hat{y}_i)$$

제로-원 손실함수 Li 은 y^ 과 y 가 같으면 0이고 다르면 1이다. 다음처럼 서술할 수도 있다.

$$L_i(\hat{y}) = \begin{cases} \frac{1}{2}(\text{sgn}(-\hat{y}) + 1) & \text{if } y = 1 \\ \frac{1}{2}(\text{sgn}(\hat{y}) + 1) & \text{if } y = -1 \end{cases}$$

그런데 제로-원 손실함수를 쓰면 y^(x) 가 x 에 대한 계단형 함수이므로 대부분의 영역에서 기울기가 0이 되어 미분값으로부터 최소점의 위치를 구할 수 없다. 따라서 퍼셉트론에서는 y^ 대신 활성화값 wTx 를 손실함수로 사용한다.

$$L_P(w) = -\sum_{i \in M} y_i \cdot w^T x_i$$

이를 퍼셉트론 손실함수(perceptron loss function) 또는 0-힌지 손실함수(zero-hinge loss function)라고 한다. 여기에서 손실값은 오분류된 표본에 대해서만 계산한다는 점에 주의하라. 이 때는 y 와 sgn(y^) 값이 다르면 오분류된 것이다.

$$\begin{aligned} \text{sgn}\hat{y} = y \rightarrow & \quad \text{right classficiation} \\ \text{sgn}\hat{y} \neq y \rightarrow & \quad \text{misclassification} \end{aligned}$$

https://datascienceschool.net/view-notebook/342b8e2ecf7a4911a727e6fe97f4ab6b/ (https://datascienceschool.net/view-notebook/342b8e2ecf7a4911a727e6fe97f4ab6b/)

퍼셉트론 손실함수는 다음처럼 표기할 수도 있다.

$$L_{P,i}(\hat{y}) = \begin{cases} -\frac{1}{2} w^T x \left(\text{sgn}(-\hat{y}) + 1\right) & \text{if } y = 1 \\ \frac{1}{2} w^T x \left(\text{sgn}(\hat{y}) + 1\right) & \text{if } y = -1 \end{cases}$$

SGD

SGD(Stochastic Gradient Descent) 방법은 손실함수 자체가 아니라 손실함수의 기댓값을 최소하는 방법이다.

3.3 성능 측정

분류기 평가는 회귀모델보다 어렵다. 사용할 수 있는 성능 지표가 많으니 여유롭게 둘러보자

3.3.1 교차 검증을 사용한 정확도 측정

교차 검증 구현

사이킷런이 제공하는 기능보다 교차 검증 과정을 더 많이 제어해야 할 필요가 있다. 이때는 교차 검증 기능을 직접 구현하면 된다. 다음 코드는 사이킷 런의 cross_val_score() 함수와 비슷한 작업을 수행하고 동일한 결과를 출력한다. 잘 보자

In [11]:
```
from sklearn.model_selection import StratifiedKFold  # 이친구는 클래스별 비율이 유지되도록 폴드를 만들기 위해 계층적 샘플링을 수행한다.
from sklearn.base import clone

skfolds = StratifiedKFold(n_splits = 3, random_state = 42)  # 훈련 세트를 3개의 폴드로 나누자
# cross_val_score(gd_clf, X_train, y_train_5, cv=skfolds)  # 이렇게 하면 skf사용해서 3겹 교차검증 사용 가능

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))
```

```
C:\Users\KIM\AppData\Roaming\Python\Python36\site-packages\sklearn\model_selection\_split.py:297: FutureWarning: Setting a random_state has no effect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its default (None), or set shuffle=True.
  FutureWarning

0.95035
0.96035
0.9604
```

In [13]:
```
# cross_val_score() 함수로 폴드 3개인 k겹 교차 검증 사용해 SGDClassifier 모델 평가하기

from sklearn.model_selection import cross_val_score
cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")  #accuracy 정확도를 보겠다고 한건데 디폴트값


# 다 95%가 넘는 점수 ㄷㄷ 정확도가 95%라는 소리다
# 근데 전체 샘플 중 5가 아닌게 90%이므로 해당 점수는 딱히 좋은게 아니다
```

Out[13]: array([0.95035, 0.96035, 0.9604])

In [27]:
```
from sklearn.base import BaseEstimator

class Never5Classifier(BaseEstimator):
    def fit(self, X, y=None):
        return self
    def predict(self, X):
        return np.zeros((len(X), 1), dtype=bool)
```

In [28]:
```
never_5_clf = Never5Classifier()
cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")

#정확도만으로 모델의 좋고 나쁨을 파악하기는 어렵다(불균형한 데이터셋)
#5가 너무 적으니깐..
# 무조건 5 아님' 으로 예측하면 90% 이상의 정확도가 나온다는 이유
```

Out[28]:
```
array([0.91125, 0.90855, 0.90915])
```

3.3.2 오차행렬

클래스A의 샘플이 클래스 B로 잘못 분류된 횟수를 세는 방법이다.

숫자 5가 3으로 잘못 분류된 횟수를 알고 싶다면 오차 행렬의 5행 3열을 조회하는 방식

오차 행렬을 만들려면..

- 실제 타깃과 비교할 수 있도록 예측값을 먼저 만든다
- 테스트 세트로 예측 만들 수 있어도 여기서 사용하지 않는다
- cross_val_predict() 함수는 사용 가능하다

In [32]:
```
from sklearn.model_selection import cross_val_predict

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

cross_val_predict

이 함수는 cross_val_score함수처럼 k겹 교차검증을 진행하나 평가 점수를 반환하지는 않고 각 테스트 폴드의 예측을 반환한다. 훈련 세트의 모든 샘플에 대해 깨끗한 예측을 얻게 되는 것.(훈련 동안 보지 못한 데이터에 대해 예측)

In [33]:
```
# confusion_matrix() 함수를 이용해 오차 행렬을 만들자 타깃클래스(y_train_5)와 예측 클래스(y_train_pred) 넣기

from sklearn.metrics import confusion_matrix
confusion_matrix(y_train_5, y_train_pred)
```

Out[33]:
```
array([[53892,   687],
       [ 1891,  3530]], dtype=int64)
```

오차 행렬의 행은 실제 클래스를 나타내고, 열은 예측한 클래스를 나타낸다. 이 행렬의 첫 행은 '5가 아닌 숫자를 53892개를 5가 아닌 것으로 정확히 분류했고, 687개는 5가 아닌데 5라고 잘못 분류했다.

두 번째 행은 5인데 5아님으로 1891개로 잘못 분류했고, 나머지 3530개를 5인데 정확히 5라고 분류했다.

(찐음, 짭양) (짭음, 찐양)

In [34]:
```
y_train_perfect_predictions = y_train_5      #완벽한 분류기일 경우
confusion_matrix(y_train_5, y_train_perfect_predictions)
```

Out[34]:
```
array([[54579,     0],
       [    0,  5421]], dtype=int64)
```

정밀도란?

TP(찐양)/(TP+FP), 즉 양성이라고 '예측한' 놈들 중에 진짜 양성이었던 친구들의 비율 가지고 정밀도라~하는 것이다..

재현율이란?

TP/(TP+FN), 즉 진짜양성이랑 거짓음성(그니까 실제로 양성인) 친구들 = '실제로'값이 양성인 애들 중에서 찐으로 판명난 비율을 가지고 재현율이라~한다.

3.3.3 정밀도와 재현율

위의 친구들을 토대로 나가보자~ 이말이야~

In [35]:
```
# 정밀도
from sklearn.metrics import precision_score, recall_score

precision_score(y_train_5, y_train_pred)    #3530/(3530+687) 정밀도!
```

Out[35]: 0.8370879772350012

In [37]:
```
# 재현율

recall_score(y_train_5, y_train_pred)       #3530/(3530+1892)

# 아까 90퍼라 그러더니 이거 뭐냐
# 여튼 정밀도와 재현율 2개의 지표가 있는데 F1이라는 다른 것도 있다.
```

Out[37]: 0.6511713705958311

F1 점수~

TP / [TP+{(FN+FP)/2}] 라고 계산해도 되고,

2(정밀도 * 재현율) / (정밀도+재현율) 해도 된다

In [39]:
```
# F1점수

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

Out[39]: 0.7325171197343846

안타깝게도 정밀도와 재현율을 모두 높일 수는 없다..이런걸 정밀도/재현율 트레이트 오프라고 한다

3.3.4 정밀도/재현율 트레이드 오프

SGDClassifier, 이 분류기는 결정함수를 사용해서 각 샘플의 점수를 계산한다. 이 점수가 임곗값보다 크다면, 샘플은 양성 클래스에 할당하고, 임곗값보다 크지 않다면 음성 클래스에 할당한다.

사이킷 런에서 임곗값을 직접 지정할 수는 없지만, 예측에 사용한 점수는 또 확인할 수 있다!

분휴기의 predict() 대신 decision_function()메서드를 호출해서 각 샘플의 점수를 얻어보자. 그리고 이 점수를 기반으로 원하는 임곗값을 정해 예측을 만들 수 있따.

In [40]:
```
y_scores = sgd_clf.decision_function([some_digit])
y_scores
```

Out[40]:
```
array([2164.22030239])
```

In [41]:
```
threshold = 0
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred

# 아까 수가 2000이 넘는데 양수지? 0보다 크니까 true값이 나왔다
# 8000으로 높이면 false 나오지 뭐
```

Out[41]:
```
array([ True])
```

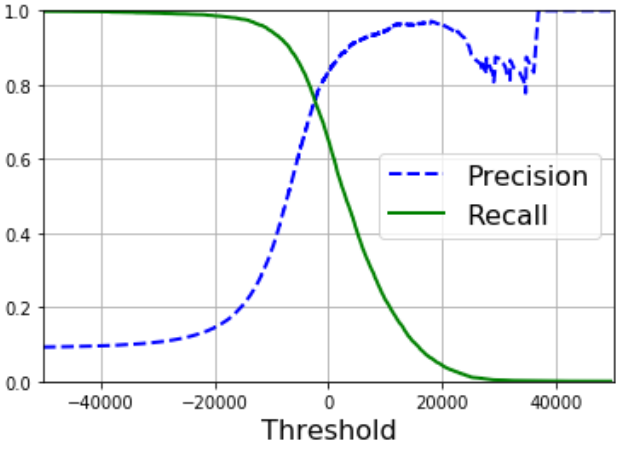
In [42]:
```
threshold = 8000
y_some_digit_pred = (y_scores > threshold)
y_some_digit_pred

# 당연히 8000이면 false 나오지.. ^^;;
# 임곗값을 높이면 재현율이 줄어드는거 보이니
# 적당한 임곗값은 어떻게 설정할까-> cross_val_predict로 모든 샘플 점수 구하기
```

Out[42]:
```
array([False])
```

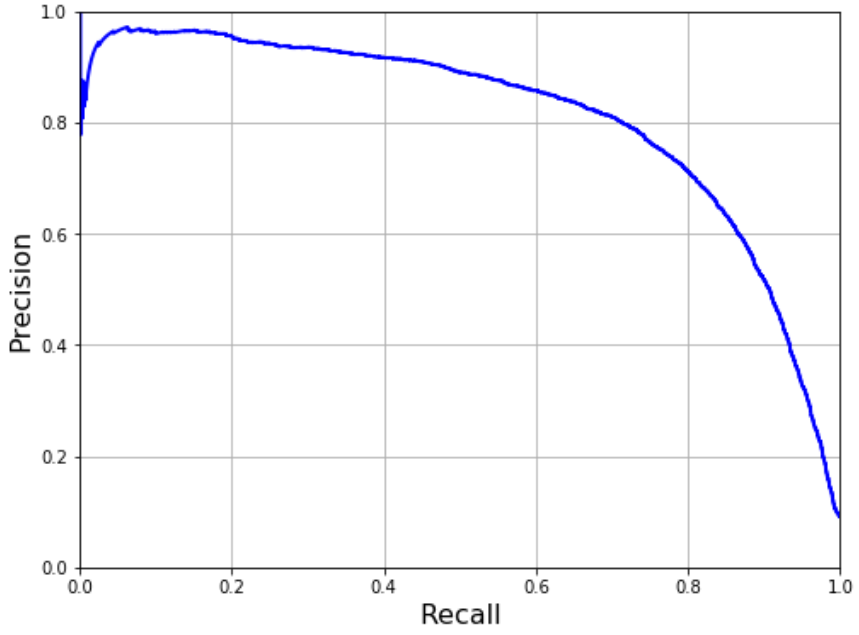
In [43]:
```
# cross_val_predict 로 모든 샘플 점수 구하기
# 이번엔, 예측결과가 아니라 결정 점수를 반환하도록!!

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

# 이 점수로 precision_recall_curve()
# 함수를 사용해 가능한 모든 임곗값에 대해 정밀도와 재현율을 계산할 수 있다규!
```

In [44]:
```
from sklearn.metrics import precision_recall_curve

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

In [51]:
```
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)
     # 임곗값을 표시하고 범례, 축 이름, 그리드를 추가합니다
    plt.legend(loc="center right", fontsize=16)
    plt.xlabel("Threshold", fontsize=16)
    plt.grid(True)
    plt.axis([-50000,50000,0,1])

plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```



In [62]:
```
# 정밀도 재현율만 뽑는 곡선

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.grid(True)

plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.show()

# from sklearn.metrics import average_precision_score
# average_precision_score(y_train_5, y_scores)
```



In [63]:
```
#argmax() 최댓값의 첫 번째 인덱스 반환
# 90보다 큰 정밀도일 때, 인덱스값을 얻고, thredshold에서의 임곗값을 찾자능

threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]
threshold_90_precision

# 이 점수면 90퍼 이상이구나아..재현율은 떨어지겠지만..^^
```

Out[63]: 3370.0194991439557

In [55]:
```
# 훈련 세트에 대한 예측

y_train_pred_90 = (y_scores >= threshold_90_precision)
```

In [56]:
```
precision_score(y_train_5, y_train_pred_90)
```

Out[56]: 0.9000345901072293

In [57]:
```
recall_score(y_train_5, y_train_pred_90)
```

Out[57]: 0.4799852425751706

3.3.5 ROC 곡선

ROC receiver operating chracteristic 곡선도 이진 분류에서 널리 사용하는 도구다. 정밀도/재현율 곡선과 매우 비슷하나, ROC곡선은 정밀도에 대한 재현율 곡선이 아니고!! ㄱ

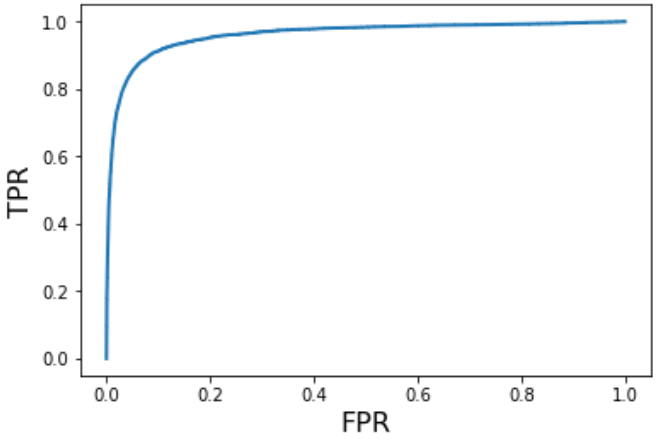
거짓양성비율(FPR 짭양비) 에 대한 진짜양성비율(TPR찐양비) 의 곡선이다.

TNR = 특이도

ROC = 민감도(재현율) 에 대한 1-특이도 그래프

In [64]:
```
# roc_curve 함수 사용해 임곗값에서 TPR, FPR 값 계산

from sklearn.metrics import roc_curve

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

In [67]:
```
def plot_roc_curve(fpr, tpr, label = None):
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plt=([0,1], [0,1], 'k--')
    plt.xlabel("FPR", fontsize=16)
    plt.ylabel("TPR", fontsize=16)
plot_roc_curve(fpr, tpr)
plt.show()
```



좋은 곡선은 우측상향 대각선에서 최대한 멀리 떨어져 있는 친구이다. 찐양율이 올라감에 따라 짭양율도 같이 증가를 해야 말이 맞는 것..

위 그래프의 곡선 아래 면적(AUC)을 측정하면 분류기들을 비교할 수 있다.

완벽한 분류기는 ROC의 AUC가 1이고, 완전한 랜덤분류기는 0.5이다.

사이킷런은 이 함수도 제공한다.

In [69]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_5, y_scores)
```

Out[69]: 0.9604938554008616

잠깐만~

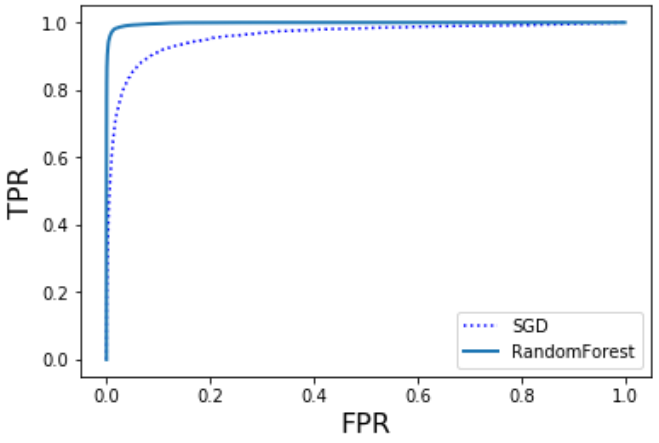
roc 곡선이 정밀도/재현율 곡선과 비슷하니까 뭘 사용해야 할지 모르겠다. 일반적으로는..

- 양성클래스가 드물거나 거짓음성(짭음)보다 거짓양성(짭양) 이 더 중요하면 PR
- 아니면 ROC 쓴다.

In [70]:
```
from sklearn.ensemble import RandomForestClassifier

# predict_proba 메소드는 샘플이 행, 클래스가 열
# 샘플이 주어진 클래스에 속할 확률을 담을 배열 반환
# roc_curve() 함수는 레이블과 점수를 기대...근데 점수 대신 클래스 확률 전달 가능

forest_clf = RandomForestClassifier(random_state = 42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")
```

In [71]:
```
# 양성 클래스 확률을 점수로 사용해보자~

y_scores_forest = y_probas_forest[:, 1]      # 양성 클래스에 대한 확률 점수로 써라!
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5, y_scores_forest)
```

In [72]:
```
plt.plot(fpr, tpr, "b:", label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, "RandomForest")
plt.legend(loc="lower right")     # 우측 하단에 뭐가 뭔지 표시
plt.show()

# 랜덤포레스트 분류기가 더 낫구먼
# 헷헷
```



In [74]:
```
roc_auc_score(y_train_5, y_scores_forest)

# 점수도 좋구만!
```

Out[74]: 0.9983436731328145

3.4 다중 분류

이진 분류가 두 개의 클래스를 구분한다구? 다중 분류기는 둘 이상의 클래스를 구별한다!

일부 알고리즘은 여러 개의 클래스를 직접 처리할 수 있다. 다른 것은 이진 분류만 가능하지만..

하지만 이진 분류기를 여러개 사용해 다중 클래스를 분류하는 기법도 많다.

이렇게 이진 분류 여러개를 만들어서 활용할 때, 각 분류기의 결정 점수 중에서 가장 높은 것을 클래스로 선정하면 된다. 이를 OvR전략이라고 한다. (One versus the Rest) (OvA)

또 다른 전략은 0과1 구별, 0과2 구별, 1과 2 구별 같이 각 숫자 조합마다 이진 분류기를 훈련시킬 수 도 있다. 이를 OvO라고 한다. 클래스가 n개 있다고 치면 n(n+1)/2 개가 필요하다.

다중 클래스 분류 작업에 이진 분류 알고리즘 선택하면, 사이킷런이 알고리즘 따라 자동으로 OvR, OvO 실행한다.

In [75]:
```
from sklearn.svm import SVC
svm_clf = SVC()
svm_clf.fit(X_train, y_train)       # y_train
svm_clf.predict([some_digit])
```

Out[75]:
```
array([5], dtype=uint8)
```

In [76]:
```
# decision_function() 샘플당 10개의 점수 반환, 클래스 별 하나씩
# 가장 높은 점수가 클래스 5에 해당하는 것

some_digit_scores = svm_clf.decision_function([some_digit])
some_digit_scores
```

Out[76]:
```
array([[ 1.72501977,  2.72809088,  7.2510018 ,  8.3076379 , -0.31087254,
         9.3132482 ,  1.70975103,  2.76765202,  6.23049537,  4.84771048]])
```

In [77]:
```
np.argmax(some_digit_scores)
```

Out[77]:
```
5
```

In [78]:
```
svm_clf.classes_
```

Out[78]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

In [79]:
```
svm_clf.classes_[5]
```

Out[79]:
```
5
```

분류기가 훈련될 때 classes_속성에 타깃 클래스의 리스트를 값으로 정렬해 저장한다. 위 예제에서는 classes_배열에 있는 각 클래스의 인덱스가 클래스 값 자체와 같다. 근데 이런 경우는 드물다.

In [ ]:
```
from sklearn.multiclass import OneVsRestClassifier
# 사이킷런에서 OvO나 OvR을 사용하도록 강제하려면 이 걸 사용해

ovr_clf = OneVsRestClassifier(SVC())
ovr_clf.fit(X_train, y_train)
ovr_clf.predict([some_digit])
```

In [ ]:
```
len(ovr_clf.estimators_)
```

In [ ]:
```
sgd_clf.fit(X_trian, y_train)
sgd_clf.predict([some_digit])
```

In [ ]:
```
# decision_function() 은 클래스마다 하나의 값을 반환한다.

sgd_clf.decision_function([some_digit])
```

In [ ]:
```
cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.astype(np.float64))
cross_val_score(sgd_clf, X_train_scaled, y_train, cv=3, scoring="accuracy")
```