```
never_5_clf = Never5Classifier()
cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
###cross_val_score 안에 자동으로 fit 매소드와 .predict 객체변수를 불러오는 듯
```

```
#결과
array([0.909  , 0.90745, 0.9125 ])
```

무려 정확도가 90%이군요... 이런 식으로 클래스별 샘플 개수가 많이 차이나는 데이터셋을 분류할 때, 교차 검증은 효과적이지 못합니다.

3.3.2 오차 행렬

이렇게 샘플의 개수가 많이 차이나는 경우 교차 검증말고 다른 지표를 사용해야 합니다. 그 중 하나가 **오차 행렬**을 조사하는 것입니다.
일단 우리 모델이 우리의 훈련 세트를 어떻게 예측했는지 살펴보겠습니다.

```
from sklearn.model_selection import cross_val_predict

y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)
## cv=3이니깐 데이터 셋을 1,2,3으로 나누고
## 1,2로 3을 예측 & 1,3으로 2를 예측 $ 2,3으로 1을 예측하고
## 우측에 있는 예측 값들을 반환함
```

이 예측을 바탕으로 오차 행렬을 만들어 봅시다.

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_train_5, y_train_pred)
```

```
#결과
array([[52972,  1607],
       [  989,  4432]], dtype=int64)
```

이 오차 행렬을 분석해보면,

- 실제로 5가 아닌 것을 모델이 5가 아니라고 말한 개수가 52972
- 실제로 5가 아닌 것을 모델이 5라고 말한 개수가 1607
- 실제로 5인 것을 모델이 5가 아니라고 말한 개수가 989
- 실제로 5인 것을 모델이 5라고 말한 개수가 4432

라는 뜻입니다. 오차 행렬을 말로 설명하려면 어려우니 예제부터 봤습니다. 표로 한 번 더 보시죠.

	모델	예측	
실제		음성	양성
실제	음성	**TN** 52972개 ex) 7,8,9,4,1,2	**FP** 1607개 ex) 5 처럼 생긴 다른 숫자
	양성	**FN** 989개 ex) 5처럼 안 생긴 5	**TP** 4432개 ex) 5,5,5

- 행: 실제 클래스, 열: 예측 클래스
- TN 진짜 음성(음성, 양성은 예측을 기준으로 말합니다.)
- FP 거짓 양성

- FN 거짓 음성
- TP 진짜 양성

오차행렬에 대한 감이 오시나요? 이 오차 행렬을 보면 이 분류기가 제대로 분석한 것 얼마나 되는 지 알 수 있습니다. 하지만 아직 어떻게 교차 검증의 한계를 뛰어넘을지 알려드리지 않았습니다. 교차 검증의 한계를 뛰어넘을 수 있는 새로운 지표를 정의하겠습니다.

- 정밀도 = $\frac{TP}{(TP+FP)}$
- 재현율 = $\frac{TP}{(TP+FN)}$

이런 비율을 정의함으로서 저희는 각 클래스에 객체수에 상관없이 모델의 성능을 측정할 수 있습니다.

3.3.3 정밀도와 재현율

이 지표들이 어떤 차이를 갖는지는 다음 섹션에서 관찰하겠습니다.
사이킷런에 정밀도와 재현율을 구하는 함수가 있습니다.

```
from sklearn.metrics import precision_score, recall_score

print (precision_score(y_train_5, y_train_pred))
print (recall_score(y_train_5, y_train_pred))
```

```
# 결과
0.7338963404537175
0.8175613355469471
```

또 다른 지표가 있는데 F1 점수라고 합니다.

- $F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$

정밀도와 재현율이 비슷한 분류기에서는 F1 점수가 높습니다. 하지만 상황에 따라 정밀도가 중요한 상황과 재현율이 중요한 상황이 있습니다.
사이킷런에 F1 점수를 구하는 함수가 있습니다.

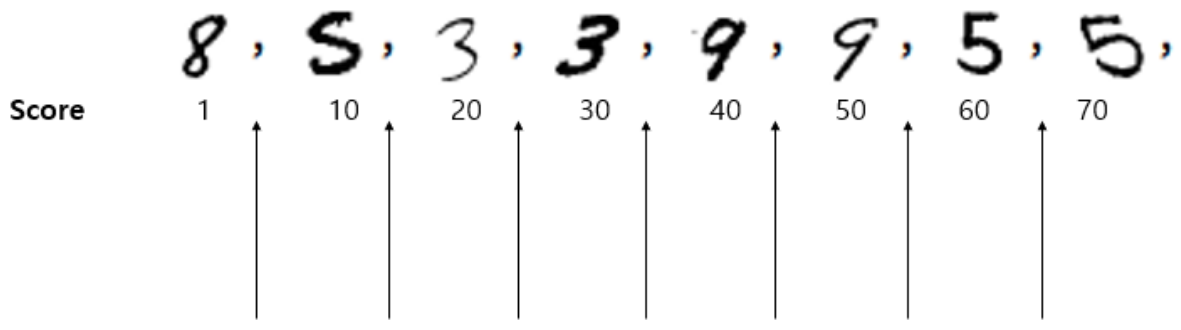
```
from sklearn.metrics import f1_score

f1_score(y_train_5, y_train_pred)
```

```
#결과
0.7734729493891798
```

3.3.4 정밀도/재현율 트레이드 오프

정밀도와 재현율에 대해 자세히 살펴보죠. 모델을 훈련 시키면 모델은 각 데이터에게 점수를 줍니다. 밑에 그림을 보면 각 점수가 1, 10, 20, 30, 40, 50, 60, 70 입니다. 이때 이 점수가 결정 임곗값(threshold)을 넘으면 양성, 넘지 않으면 음성입니다.



Threshold	5	15	25	35	45	55	65
Precision	3/7	2/6	2/5	2/4	2/3	2/2	2/2
Recall	3/3	2/3	2/3	2/3	2/3	2/3	1/3

여기서 중요한게 결정 임곗값에 따라 정밀도와 재현율이 달라집니다. 그림에서 볼 수 있듯이 임곗값이 올라갈 수록 정밀도는 커지고 재현율은 줄어듭니다.

가장 기초적으로는 임곗값이 올라가면 FP가 감소하고 FN이 증가합니다.

이해가 쉽도록 몇 가지 성질을 말해보겠습니다.

- 임곗값 증가 »> 정밀도 증가, 재현율 감소
- 임곗값 감소 »> 재현율 감소, 정밀도 증가

- FP 감소 & FN 증가 »> 정밀도 증가
- FP 증가 & FN 감소 »> 재현율 증가

좀 더 나아가서 말로 풀어보겠습니다.

- 정밀도가 높다 - 거짓을 잘 구분해낸다. 참을 놓칠 수 있다.
- 재현율이 높다 - 거짓을 잘 구분하지 못한다. 참을 놓치지 않는다.

- 정밀도가 높다 - 참이라고 말한 것 중에 참인 것이 많다.
- 정밀도가 높다 - 거짓인 것 중 거짓이라고 말한 것이 많다.

- 재현율이 높다 - 참인 것 중 참이라고 말한 것이 많다.
- 재현율이 높다 - 거짓이라고 말한 것 중 거짓인 것이 많다.

결과적으로 두 지표는 트레이드 오프 관계입니다. 상황에 따라 정밀도가 중요한 상황과 재현율이 중요한 상황이 있습니다.

1. 암환자를 구별할 때, 임곗값을 낮춰서 재현율을 높이는 것이 좋습니다. 왜냐하면 실제로 암에 걸리지는 않은 환자가 있을 수는 있지만 암에 걸린 환자는 확실히 치료를 시도할 수 있으니까요.
2. 판사가 재판을 할 때, 임곗값을 높여서 정밀도를 높이는 것이 좋습니다. 왜냐하면 무죄추정의 원칙에 의해서 무고한 사람이 감옥에 가면 안되기 때문입니다.

적절한 임곗값을 구하기 위해 모든 샘플의 점수를 구해봅시다.

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
##어떤 식으로 scoring을 계산하는지는 모릅니다.
##나중에 4단원에서 안나오면 다시 공부할 것입니다.
```

이 점수로 precision_recall_curve() 함수를 사용해서 가능한 임곗값에 대해 정밀도와 재현율 그래프를 그릴 수 있습니다.

```
from sklearn.metrics import precision_recall_curve

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

여기서 제 사적인 궁금증이 생겼었습니다. 일단 그림을 보시죠.

```
In [40]: y_scores.shape
Out[40]: (60000,)
```

```
In [43]: print (type(precisions), type(recalls), type(thresholds))
         print (len(precisions), len(recalls), len(thresholds))
         print (np.sort(y_scores))
         print (np.sort(thresholds))
         print (len(np.unique(y_scores)))
         ###Q1. 왜 thresholds는 하나가 적을까?
         ###Q2. y_scores와 thresholds는 왜 차이가 날까?
         ####A1. 밑에서
         #### A2. TF의 변화가 있을 때까지 threshold가 나오지 않음
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
59696 59696 59695
[-2971577.63808833 -2805454.32621884 -2657357.5057543  ...
  949804.17166689   967758.32875719  1075693.60395262]
[-1652198.87312432 -1649262.25561156 -1648466.62976267 ...
  949804.17166689   967758.32875719  1075693.60395262]
60000
```

Q2는 쉽게 알 수 있지만, Q1은 왜인지 알기가 어려웠습니다. 하지만 열심히 탐구해서 정답 비슷한걸 알게 된 것 같아서 여기에 적겠습니다. ㅎㅎ

threshold가 매우 크다면

1. FP에 있는 샘플이 0에 가깝다.
2. score가 가장 높은 것이 FP 혹은 TP에 있을 것(여기서 만약 임곗값이 score가 가장 큰 객체의 score보다 커지면 정밀도의 분모가 0이 됩니다. 그래서 threshold의 마지막 숫자는 score 중 두 번째로 큰 수이고 마지막 precision과 recall 값은 1과 0입니다).
    - 2-1 정밀도
        - FP에 있다면 0/(0+1) = 0
        - TP에 있다면 1/(1+0) = 1
    - 2-2 재현율
        - FP에 있다면 0/(0+5의 개수) = 0
        - TP에 있다면 1/(1+5의 개수) >= 0

threshold가 매우 작다면

1. FP에 있는 샘플이 0에 가깝다.
2. score가 가장 낮은 것이 TN 혹은 FN에 있을 것.
    - 2-1 정밀도
        - TN에 있다면 적당한 비율 (5의 개수/전체)
        - FN에 있다면 적당한 비율 (5의 개수/전체)
    - 2-2 재현율
        - TN에 있다면 5의 개수/(5의 개수 + 0)
        - FN에 있다면 5의 개수/(5의 개수 + 1)

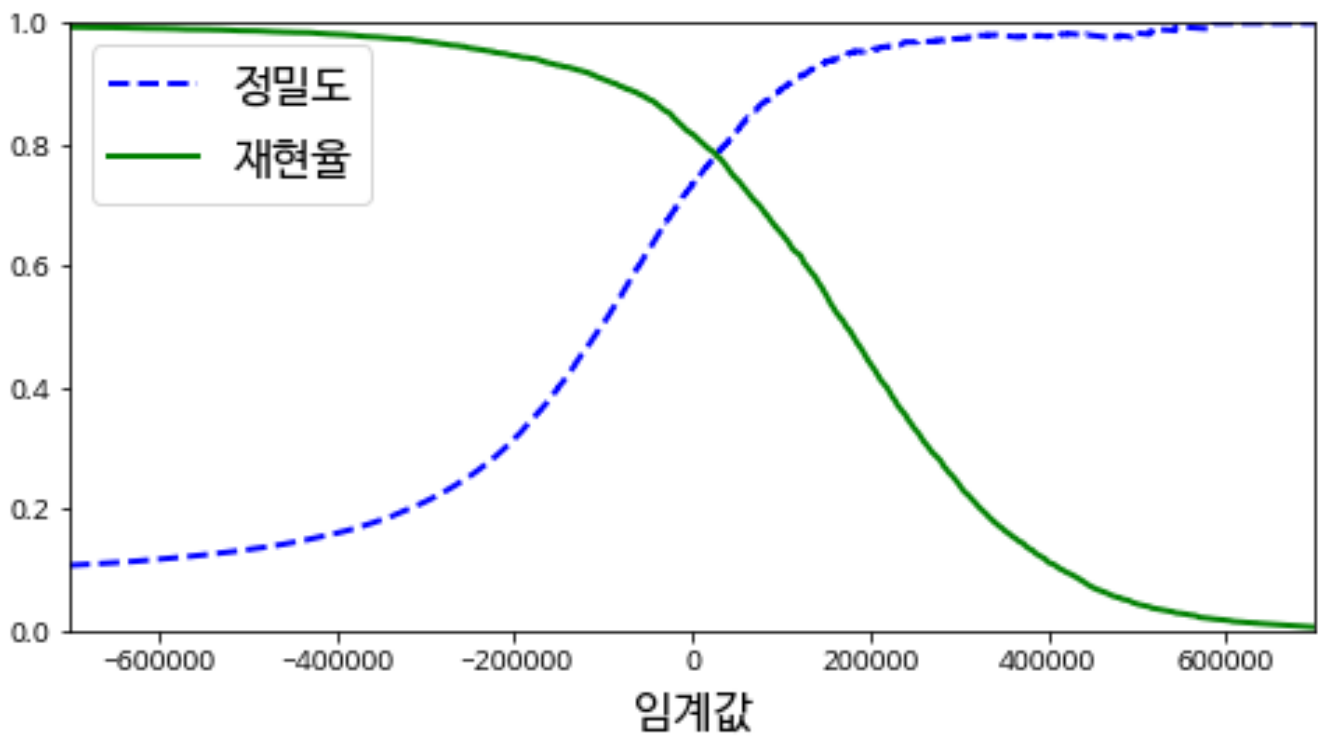
In [52]:
```
precisions[-1], recalls[-1]
```

Out[52]: (1.0, 0.0)

이제 맷플롭립을 이용해 정밀도와 재현율 함수를 그려보겠습니다.

In [44]:
```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="정밀도", linewidth=2)
    plt.plot(thresholds, recalls[:-1], "g-", label="재현율", linewidth=2)
    plt.xlabel("임계값", 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])
plt.show()
```

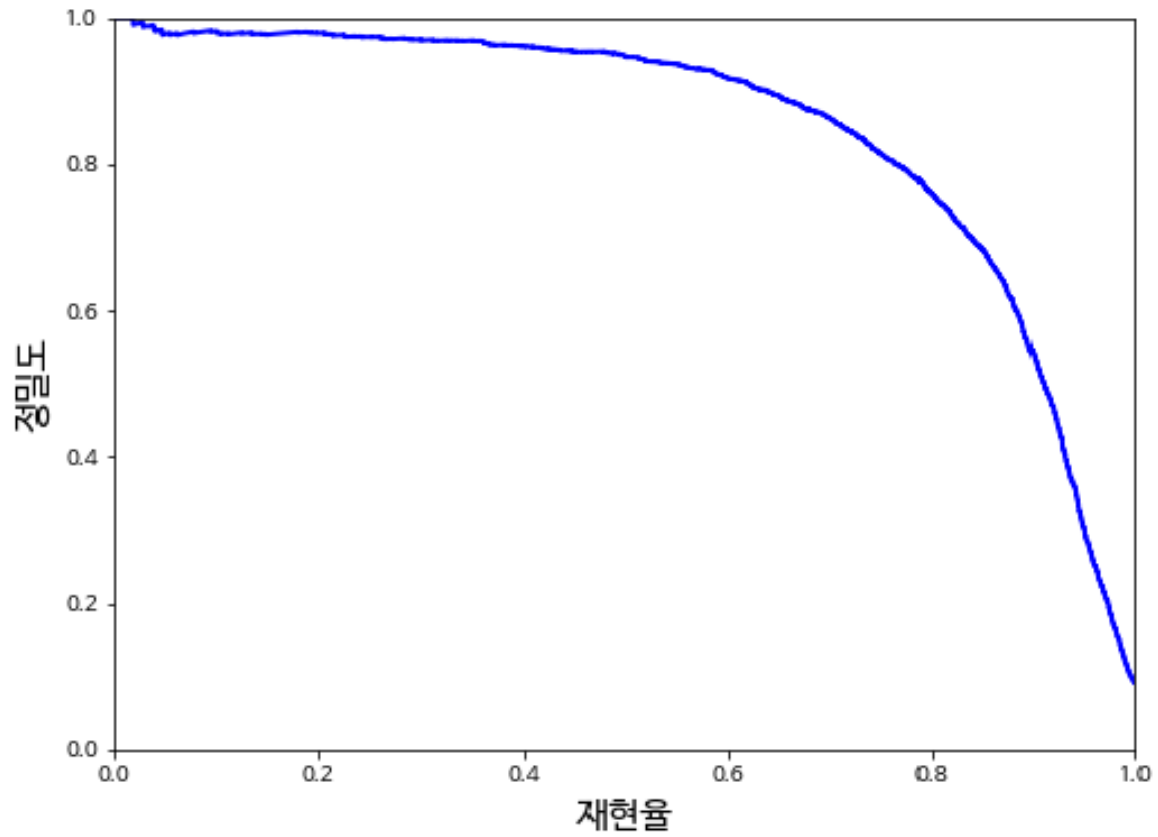


정밀도 곡선이 재현율 곡선보다 왜 더 울퉁불퉁한지도 위에 설명을 잘 이해하셨다면 이해하실 수 있습니다.

재현율에 대한 정밀도 곡선을 그리면 좋은 정밀도/재현율 트레이드오프를 선택할 수 있습니다.

In [45]:
```
def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("재현율", fontsize=16)
    plt.ylabel("정밀도", fontsize=16)
    plt.axis([0, 1, 0, 1])

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



만약 정밀도 90%가 목표라고 합시다.

```
y_train_pred_90 = (y_scores > 70000) ##대충보고... 임계값을 설정
print(precision_score(y_train_5, y_train_pred_90)) #정밀도 계산
print(recall_score(y_train_5, y_train_pred_90)) #재현율 계산
```

```
#결과
0.855198572066042
0.7070651171370596
```

우리는 분류기를 만들 때, 정밀도와 재현율을 상대적으로 비교해서 분류기를 만들어야합니다. 누군가가 '99% 정밀도를 달성하자'라고 말하면 반드시 '재현율 얼마에서?'라고 물어야합니다.

3.3.5 ROC 곡선

이번 섹션에서는 이진 분류 모델의 지표 중 하나인 ROC(receiver operating characteristic)를 배워보도록 하겠습니다. 거짓 양성 비율(FPR)에 대한 진짜 양성 비율(TPR, 재연율의 다른 이름)입니다.

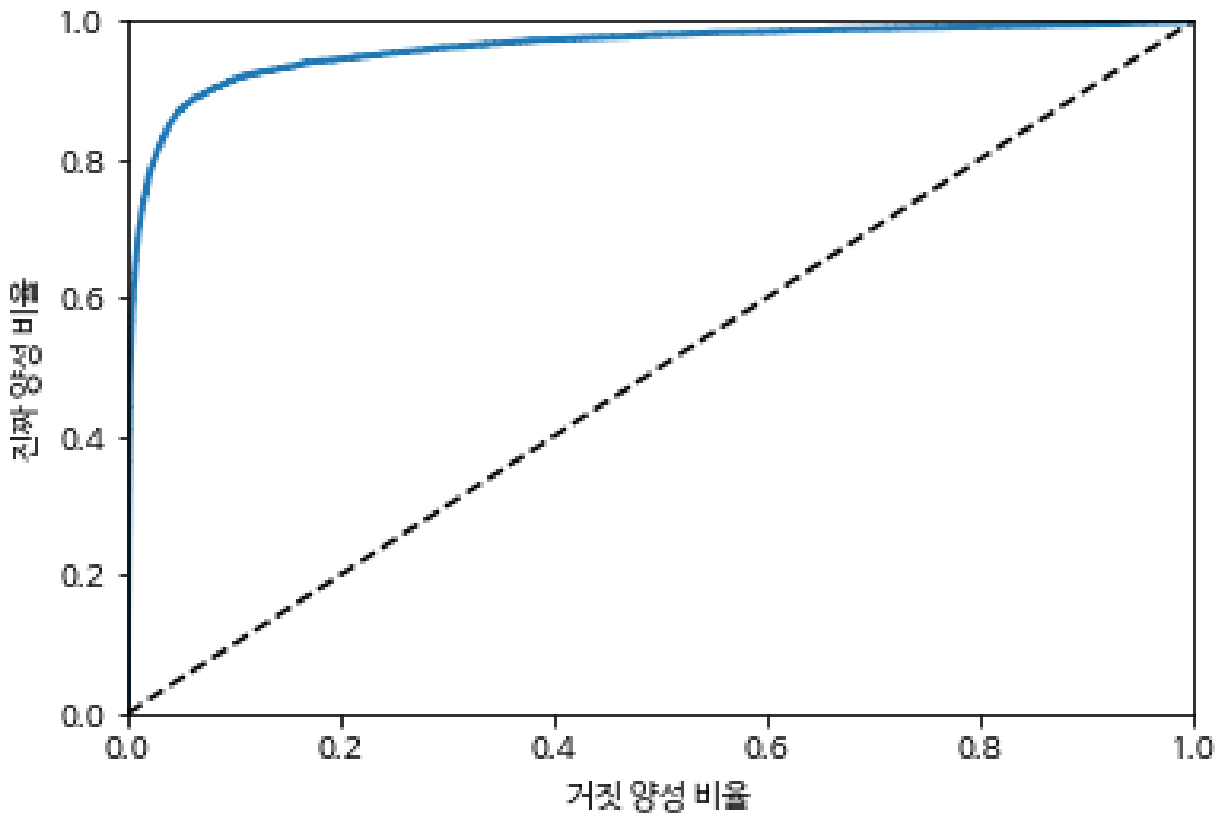
- 거짓 양성 비율(FPR) = $\frac{FP}{(FP+TN)}$ (낮을 수록 좋음)

```
from sklearn.metrics import roc_curve

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

In [50]:
```
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('거짓 양성 비율')
    plt.ylabel('진짜 양성 비율')

plot_roc_curve(fpr, tpr) ##양의 상관 관계
```



거짓 양성 비율도 진짜 양성 비율(재현율)과 트레이드오프 관계가 있습니다. 좋은 분류기는 y=x 그래프와 ROC 곡선이 최대한 멀리 떨어져 있어야 합니다. 곡선 아래의 면적을 새로운 지표로 생각하고 이를 통해 분류기들을 비교할 수 있습니다. 이를 **AUC(area under the curve) 측정** 이라고 합니다.

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_train_5, y_scores)
```

```
#결과
0.9614189997126434
```

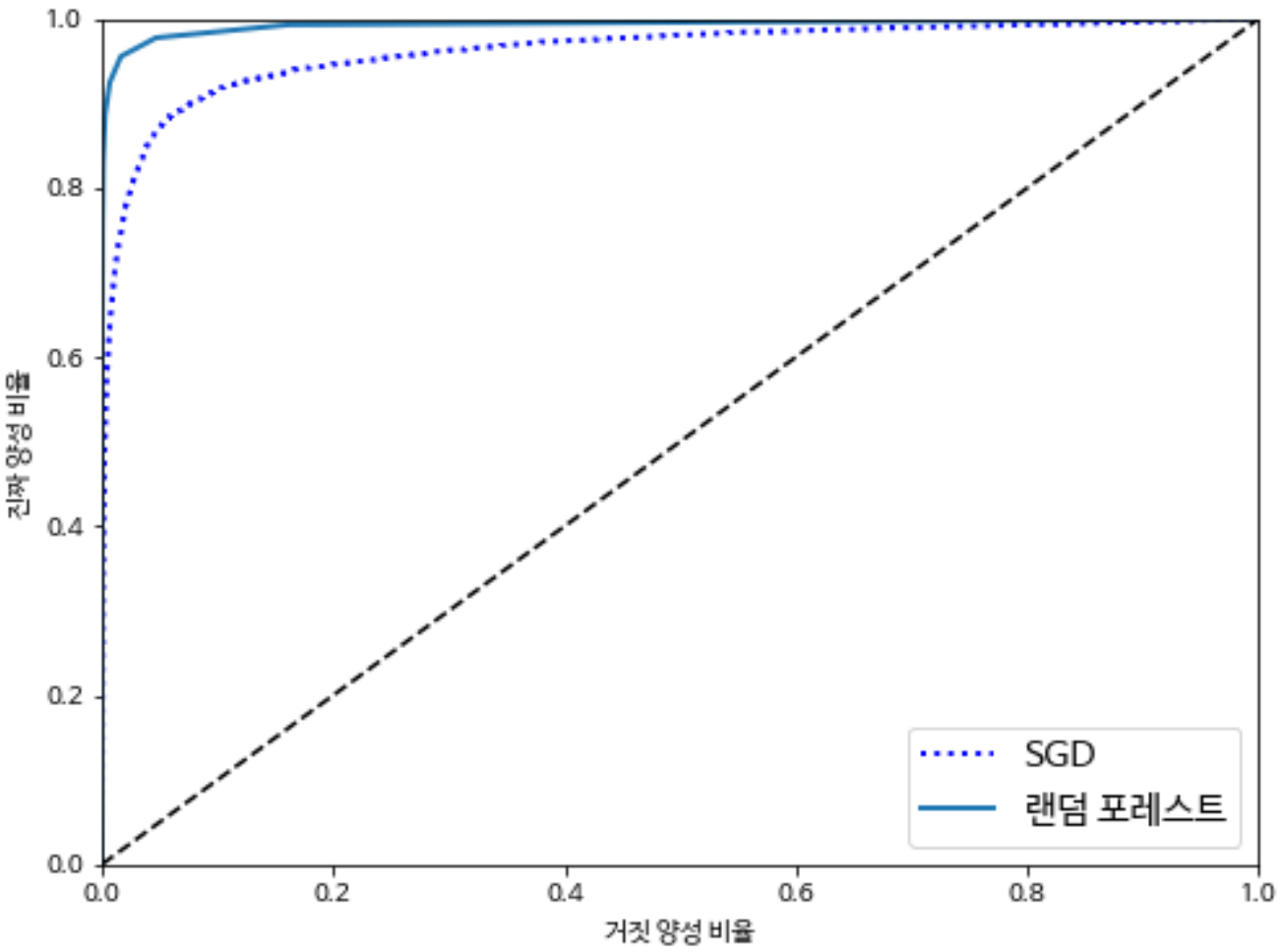
RandomForestClassifier와 SGDClassifier를 비교해보겠습니다

```
from sklearn.ensemble import RandomForestClassifier

forest_clf = RandomForestClassifier(n_estimators=10, random_state=42)
y_probas_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3,
                                    method="predict_proba")
```

```
y_scores_forest = y_probas_forest[:, 1] ## 5 클래스에 들어갈 확률을 점수로 사용
fpr_forest, tpr_forest, thresholds_forest = roc_curve(y_train_5,y_scores_forest)
## y_train_5는 bool 값을 가지고 있는 배열
```

In [60]:
```
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, "b:", linewidth=2, label="SGD")
plot_roc_curve(fpr_forest, tpr_forest, label="랜덤 포레스트")
plt.legend(loc="lower right", fontsize=14)
plt.show()
```




```
roc_auc_score(y_train_5, y_scores_forest)
```

```
0.9928250745111685
```

이것으로 RandomForestClassifier가 SGDClassifier보다 좋은 것을 알 수 있습니다. 추가로 정밀도와 재현율을 구해보겠습니다.

```
y_train_pred_forest = cross_val_predict(forest_clf, X_train, y_train_5, cv=3)
precision_score(y_train_5, y_train_pred_forest)
```

```
#결과
0.9870386643233744
```

```
recall_score(y_train_5, y_train_pred_forest)
```

```
#결과
0.8288138719793396
```

3.4 다중 분류

이진 분류가 두 개의 클래스를 구별한다면 다중 분류기는 둘 이상의 클래스를 구별할 수 있습니다. 하지만 일부 알고리즘은 여러 개의 클래스를 직접 처리할 수 있지만, 몇 몇 알고리즘은 이진 분류만 가능합니다. 그럼에도 불구하고 이진 분류기를 여러 개 이용해 다중 클래스를 분류하는 기법도 있습니다.

1. 일대다(OvA)전략: 특정 숫자 하나만 구분하는 숫자별 이진 분류기 10개(0에서 부터 9까지)를 훈련시켜 클래스가 10개인 숫자 이미지 분류 시스템을 만들 수 있습니다. 이미지를 분류할 때 각 분류기의 결정 점수 중에서 가장 높은 것을 클래스로 선택하면 됩니다.