Objetivo

Criar um modelo de classificação binária que detecte se uma imagem do conjunto de dados MNIST contém o número "5" ou não, esse projeto foi feito com base no livro Mãos À Obra: Aprendizado de Máquina com Scikit-Learn, Keras E TensorFlow.

1. Configurações basicas

Bibliotecas utilizadas

In [1]:
```
import sklearn
import numpy as np
import os
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Configuração das figuras

In [2]:
```
%matplotlib inline
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

Função para salvar figuras

In [3]:
```
PROJECT_ROOT_DIR = r"C:\Users\euric\Desktop\lml"
CHAPTER_ID = "classification"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

Semente para garantir os mesmos resultados

In [4]:
```
np.random.seed(42)
```

2. Obtendo o dataset MNIST

O dataset MNIST contém imagens de dígitos escritos à mão (0–9). Cada imagem tem 28x28 pixels (784 características).

In [5]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1, as_frame=False)
mnist.keys()
```

Out[5]:
```
dict_keys(['data', 'target', 'frame', 'categories', 'feature_names', 'target_na
mes', 'DESCR', 'details', 'url'])
```

Formato do dataset:

In [6]:
```
X, y = mnist["data"], mnist["target"]
X.shape
```

Out[6]:
```
(70000, 784)
```

In [7]:
```
y.shape
```

Out[7]:
```
(70000,)
```

Temos 70.000 imagens (linhas) e 784 pixels por imagem (colunas)

3. Transformação dos Dados

Os dados foram divididos em 60.000 imagens para treino, 10.000 para teste e o rótulo y foi convertido para tipo inteiro

In [8]:
```
y = y.astype(np.uint8)
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Criação dos rótulos:

y_train_5 e y_test_5 são True para imagens do dígito 5 e False para outros números.

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

4. Treinando o Classificador SGD

In [10]:
```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(max_iter=1000, tol=1e-3, random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

Out[10]:
```
SGDClassifier(random_state=42)
```

Predição com uma imagem:

In [11]:
```
some_digit = X[0]
sgd_clf.predict([some_digit])
```

Out[11]:
```
array([ True])
```

o modelo previu que some_digit é o número 5

5. Avaliando o modelo:

Validação Cruzada:

Na validação cruzada, os dados de treino são divididos em partes (chamadas de folds) de forma estratificada, garantindo que cada fold mantenha a proporção de exemplos positivos (número 5) e negativos (não 5). O processo é realizado em três etapas:

1. O classificador é clonado e retreinado com os dados de cada combinação de folds de treino.

2. O modelo treinado é avaliado no fold restante (dados de validação).

3. Esse processo se repete para garantir que cada fold seja usado como conjunto de validação exatamente uma vez.

Foi utilizada uma validação cruzada com 3 folds:

In [12]:
```
from sklearn.model_selection import StratifiedKFold
from sklearn.base import clone
```

```
skfolds = StratifiedKFold(n_splits=3, shuffle=True, 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))
```

```
0.9669
0.91625
0.96785
```

A menor acurácia dos folds foi de 91%, mas devido ao desequilíbrio de classes (apenas cerca de 10% das imagens representam o número 5), a alta acurácia pode ser enganosa. Ex: Um classificador que sempre prevê "não é 5" já teria uma acurácia de 90%.

Sendo necessário utilizar outras metricas para avaliar

6. Matriz de Confusão

A matriz de confusão é uma ferramenta que avalia o desempenho de um classificador, mostrando como as previsões se comparam aos rótulos reais. Ela é organizada em uma tabela que contém os seguintes valores:

True Negatives (TN): Quantidade de exemplos corretamente classificados como não sendo 5.

False Positives (FP): Quantidade de exemplos incorretamente classificados como sendo 5 (falsos alarmes).

False Negatives (FN): Quantidade de exemplos incorretamente classificados como não sendo 5 (erros de omissão).

True Positives (TP): Quantidade de exemplos corretamente classificados como sendo 5.

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)
```

Em vez de utilizar o conjunto de teste para avaliar o modelo, o código realiza uma validação cruzada utilizando a função cross_val_predict. Isso retorna as predições feitas pelo modelo durante a validação cruzada, garantindo que o modelo nunca veja os mesmos dados usados para validação.

In [14]:
```
from sklearn.metrics import confusion_matrix
confusion = confusion_matrix(y_train_5, y_train_pred)
fig, ax = plt.subplots(figsize=(5, 3))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=confusion,
                 rowLabels=["Não é 5", "É 5"],
                 colLabels=["Predito: Não é 5", "Predito: É 5"],
                 loc="center",
                 cellLoc="center")
table.auto_set_font_size(False)
table.set_fontsize(12)
table.auto_set_column_width(col=list(range(len(confusion[0]))))

plt.show()
```

	Predito: Não é 5	Predito: É 5
Não é 5	53892	687
É 5	1891	3530

Na tabela observamos que 2578 imagens foram classificadas incorretamente. Esse número representa a soma de 687 Falsos Positivos e 1891 Falsos Negativos, pode parecer pequeno em relação ao total, de 60000, mas a análise muda ao considerar o desbalanceamento das classes, em tono de 10% das imagens são 5, assim o modelo errou 1891 de 6000 imagens reais da classe "5".

Isso equivale a um erro de quase 1/3 para essa classe.

7. Precisão, Revocação e F1-Score

Precisão

A precisão mede a quantidade de predições corretas que o modelo fez sobre a classe positiva em relação ao total de predições feitas como positivas. Em outras palavras, ela avalia quantas das imagens que o modelo classificou como "5" realmente eram "5".

Precisão = Verdadeiros Positivos (TP)/(Verdadeiros Positivos (TP)+ Falsos Positivos (FP))

In [15]:
```
from sklearn.metrics import precision_score, recall_score
precision_score(y_train_5, y_train_pred)
```

Out[15]:
```
0.8370879772350012
```

Quando comparamos com a acurácia vemos uma dimuição de quase 10%, com 84% de precisão o modelo classifica corretamente em torno de 4 em 5 previsões.

Revocação

A revocação mede a capacidade do modelo de identificar todas as instâncias reais da classe positiva. Em outras palavras, ela avalia quantas das imagens que realmente são "5" o modelo foi capaz de identificar corretamente.

Revocação = Verdadeiros Positivos (TP)/(Verdadeiros Positivos (TP)+ Falsos Negativos (FN))

In [16]:
```
recall_score(y_train_5, y_train_pred)
```

Out[16]:
```
0.6511713705958311
```

Agora olhando a Revocação vemos uma piora ainda maior, com 65%, significa que 2 em 3 previsões feitas estão corretas.

F1-Score

O F1-Score é a média harmônica da precisão e revocação, sendo uma métrica que busca equilibrar ambas as medidas.

F1-Score = 2 * ((Precisão*Revocação)/(Precisão+Revocação))

In [17]:
```
from sklearn.metrics import f1_score
f1_score(y_train_5, y_train_pred)
```

Out[17]:
```
0.7325171197343847
```

Com o F1-Score de 73%, o modelo tem um desempenho razoável.

8. Trade-off entre Precisão e Revocação

Precisão e Revocação, são metricas que entram em conflito, porque, ao tentar melhorar uma, você prejudica a outra.

Se o modelo for ajustado para ser mais rigoroso ao classificar algo como positivo, Isso tende a reduzir os falsos positivos (FP), mas deixa de identificar alguns verdadeiros "5"s, aumentando os falsos negativos (FN) reduzindo a revocação, o inverso também ocorre, se ajustado para ser menos rigoroso e classificar mais exemplos como positivos, teremos uma menor precisão.

O SGD, para fazer sua previsão, atribui scores a classe e se esse score for maior que um determinado limiar é clasificado como positivo, não podemos definir diretamente o limiar, mas com decision_function retorna o score e faz previsões com qualquer limiar.

In [18]:
```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3, method="decision
```

In [19]:
```
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

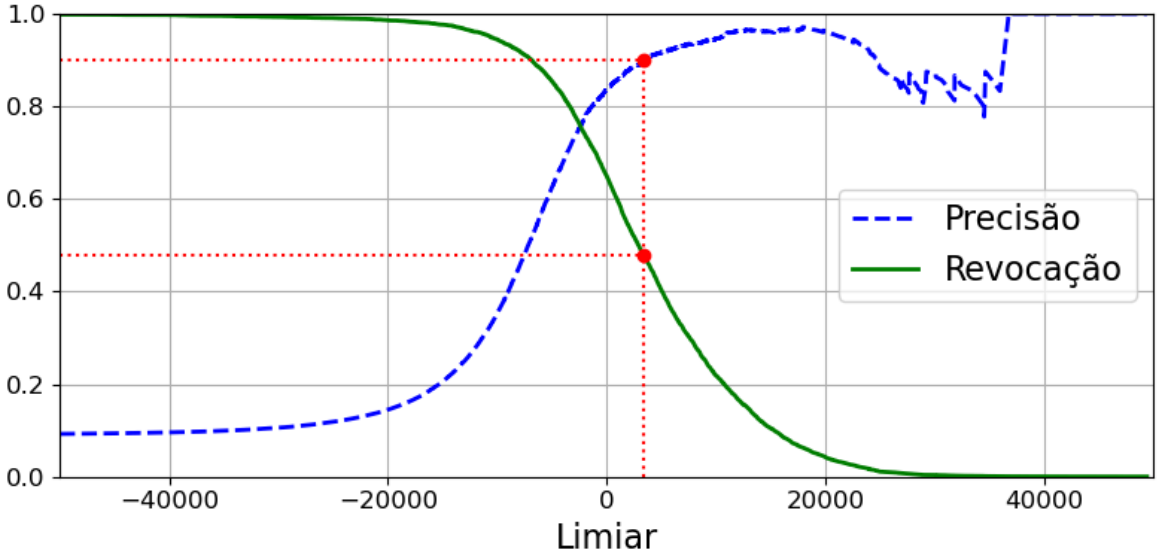
In [20]:
```
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precisão", linewidth=2)
    plt.plot(thresholds, recalls[:-1], "g-", label="Revocação", linewidth=2)
    plt.legend(loc="center right", fontsize=16)
    plt.xlabel("Limiar", fontsize=16)
    plt.grid(True)
    plt.axis([-50000, 50000, 0, 1])



recall_90_precision = recalls[np.argmax(precisions >= 0.90)]
threshold_90_precision = thresholds[np.argmax(precisions >= 0.90)]
```

```
plt.figure(figsize=(8, 4))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.plot([threshold_90_precision, threshold_90_precision], [0., 0.9], "r:")
plt.plot([-50000, threshold_90_precision], [0.9, 0.9], "r:")
plt.plot([-50000, threshold_90_precision], [recall_90_precision, recall_90_preci
plt.plot([threshold_90_precision], [0.9], "ro")
plt.plot([threshold_90_precision], [recall_90_precision], "ro")
save_fig("precision_recall_vs_threshold_plot")
plt.show()
```

```
Saving figure precision_recall_vs_threshold_plot
```



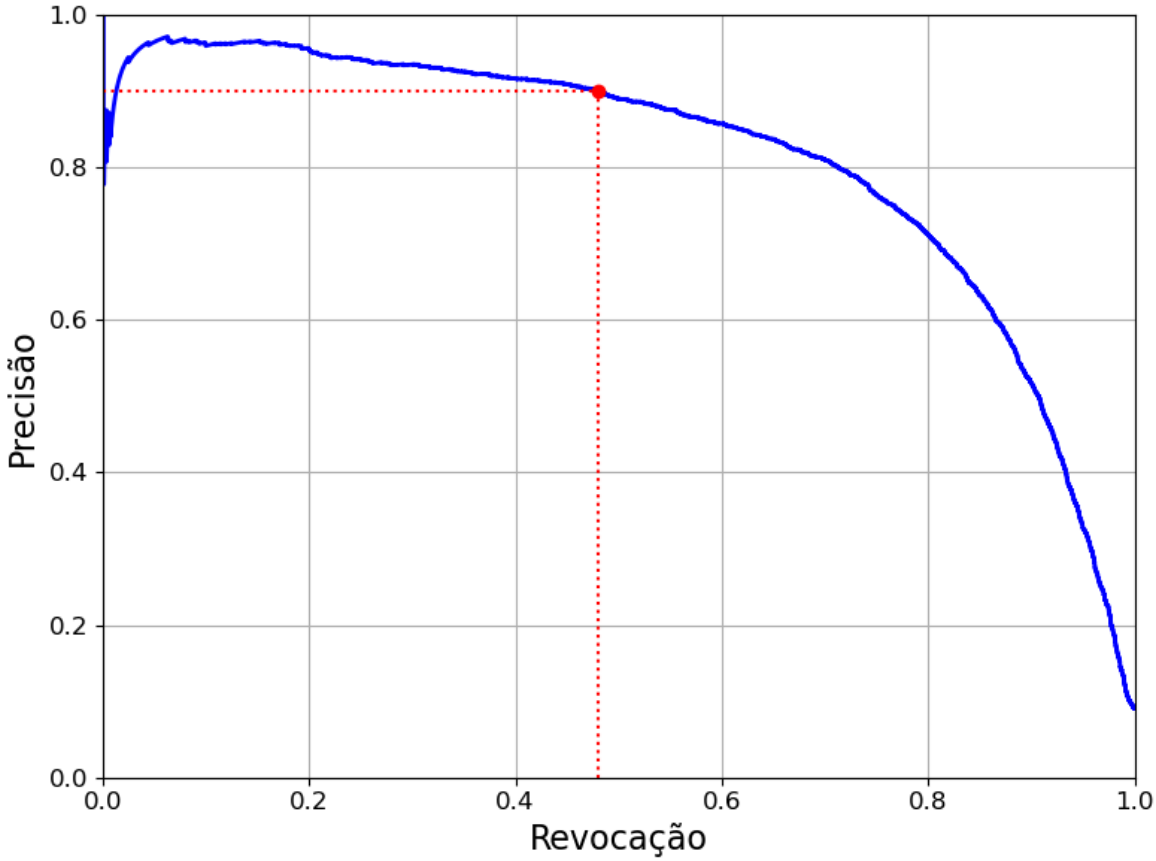
Na figura podemos ver como a precisão e revocação mudam em relação ao Limiar, os pontos vermelhos indicam os valores para uma precisão de 90%, mas temos uma Revocação menor que a anterior(65%), ou seja para ter 90% de certeza de que um dígito é 5, o modelo acaba perdendo mais da metade dos 5 verdadeiros.

Outra forma de vizualizar esse trade-off, é plotar a Precisão/Revocação.

In [21]:
```
def plot_precision_vs_recall(precisions, recalls):
    plt.plot(recalls, precisions, "b-", linewidth=2)
    plt.xlabel("Revocação", fontsize=16)
    plt.ylabel("Precisão", fontsize=16)
    plt.axis([0, 1, 0, 1])
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_precision_vs_recall(precisions, recalls)
plt.plot([recall_90_precision, recall_90_precision], [0., 0.9], "r:")
plt.plot([0.0, recall_90_precision], [0.9, 0.9], "r:")
plt.plot([recall_90_precision], [0.9], "ro")
save_fig("precision_vs_recall_plot")
plt.show()
```

```
Saving figure precision_vs_recall_plot
```



Na figura podemos ver como a Revocação muda em relação a Precisão e escolher um ponto de equilibrio, vemos que para 90% de precisão temos em torno de 45% revocação, menos que anteriormente, o que sugere que 73% e 65% já eram um bom ponto de equilibrio.

9. Curva ROC

A curva ROC (Receiver Operating Characteristic) mede a capacidade do modelo de distinguir entre classes positivas e negativas em diferentes limiares de decisão. Ela representa a relação entre a taxa de verdadeiros positivos (TPR), ou revocação, e a taxa de falsos positivos (FPR). O FPR é definido como a proporção de instâncias negativas classificadas incorretamente como positivas e pode ser calculado como FPR=1−TNR, onde TNR (Taxa de Verdadeiros Negativos) é a proporção de negativos classificados corretamente, também conhecida como especificidade. Dessa forma, a curva ROC mostra a revocação em relação à especificidade,

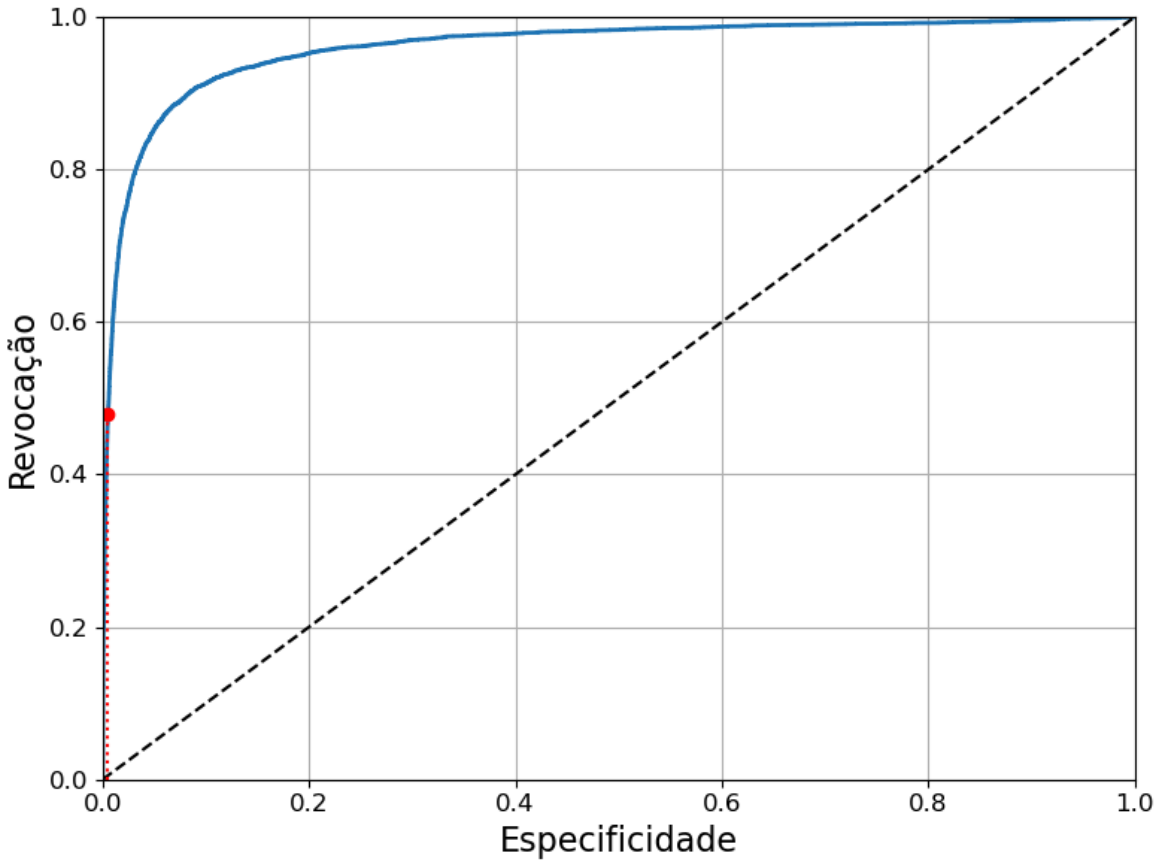
permitindo avaliar o desempenho do modelo em diversos limiares.

In [22]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_train_5, y_scores)
```

In [24]:
```
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
    plt.axis([0, 1, 0, 1])
    plt.xlabel('Especificidade', fontsize=16)
    plt.ylabel('Revocação', fontsize=16)
    plt.grid(True)

plt.figure(figsize=(8, 6))
plot_roc_curve(fpr, tpr)
fpr_90 = fpr[np.argmax(tpr >= recall_90_precision)]
plt.plot([fpr_90, fpr_90], [0., recall_90_precision], "r:")
plt.plot([0.0, fpr_90], [recall_90_precision, recall_90_precision], "r:")
plt.plot([fpr_90], [recall_90_precision], "ro")
save_fig("roc_curve_plot")
plt.show()
```

```
Saving figure roc_curve_plot
```



Na figura podemos ver a linha diagonal preta que representa o desempenho de um modelo aleatório, quanto mais distante a curva ROC estiver dessa linha, melhor é o desempenho do modelo, nesse caso, a curva ROC está bem

próxima do canto superior esquerdo, indicando um bom desempenho geral, isso significa que ele consegue manter uma alta Revocação enquanto minimiza a taxa de falsos positivos, o ponto vermelho representa a revocação escolhida 43%.

Outra forma de avaliar é calcular a área sobre a curva, fornecendo um único número que resume o desempenho geral do modelo, idealmente queremos um valor igual a 1, que indica que o modelo consegue distinguir perfeitamente entre as classes positivas e negativas.

In [25]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train_5, y_scores)
```

Out[25]:
```
0.9604938554008616
```

Com um valor de 96%, o ROC AUC demonstra que o modelo possui uma boa capacidade de distinguir entre as classes positivas e negativas. No entanto, é importante lembrar que o conjunto de dados é desbalanceado, com apenas cerca de 10% das imagens representando o número "5". Nesse contexto, o modelo terá alta acurácia ao prever que "não é 5", mas isso não necessariamente reflete sua real capacidade de identificar a classe positiva. Por isso, a curva Precisão/Revocação oferece uma visão mais detalhada do desempenho do modelo, destacando que, ainda há espaço para melhorias, especialmente na identificação correta dos "5".