Classificação

Slides extraídos do livro "*Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*" de Aurélien Géron

Sumário

Sumário

O Dataset MNIST

- Nesta aula, usaremos o dataset MNIST.
- É um conjunto de 70.000 pequenas imagens de dígitos manuscritos por estudantes do ensino médio e funcionários do US Census Bureau.
- Cada imagem é rotulada com o dígito que representa.
- É frequentemente chamado de "hello world" do Aprendizado de Máquina.

Figure: Dígitos do dataset MNIST.

Carregando os Dados

O Scikit-Learn fornece funções auxiliares para baixar datasets populares.

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', as_frame=False) # as_frame=False retorna arrays NumPy
```

Inspecionando os arrays de dados:

```
X, y = mnist.data, mnist.target
>>> X.shape
(70000, 784)
>>> y.shape
(70000,)
```

- 70.000 imagens, cada uma com 784 features.
- Cada imagem tem 28x28 pixels, e cada feature representa a intensidade de um pixel (0 a 255).

Visualizando um Dígito

Podemos pegar o vetor de features de uma instância, remodelá-lo para um array 28x28 e exibi-lo.

```
import matplotlib.pyplot as plt

def plot_digit(image_data):
    image = image_data.reshape(28, 28)
    plt.imshow(image, cmap="binary")
    plt.axis("off")

some_digit = X[0]
plot_digit(some_digit)
plt.show()
```

O rótulo confirma:

```
>>> y[0]
'5'
```

Figure: Exemplo de uma imagem MNIST.

Criando Conjuntos de Treino e Teste

- **Importante**: Sempre crie um conjunto de teste e deixe-o de lado antes de inspecionar os dados de perto.
- O dataset MNIST será dividido em um conjunto de treinamento (primeiras 60.000 imagens) e um conjunto de teste (últimas 10.000 imagens).
- O conjunto de treinamento já está embaralhado, o que é bom para a validação cruzada.

```
X_train, X_test, y_train, y_test = X[:60000], X[60000:], y[:60000], y[60000:]
```

Sumário

Treinando um Classificador Binário

Vamos simplificar o problema e tentar identificar apenas um dígito — por exemplo, o número 5.
Este "detector de 5" será um exemplo de **classificador binário**.
Primeiro, criamos os vetores alvo (rótulos) para esta tarefa de classificação:

```
y_train_5 = (y_train == '5')  # True para todos os 5s, False para outros
y_test_5 = (y_test == '5')
```

Agora, vamos escolher um classificador e treiná-lo.
Um bom ponto de partida é o Stochastic Gradient Descent (SGD).

```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

Vamos usá-lo para detectar a imagem do dígito 5 que vimos antes:

```
>>> sgd_clf.predict([some_digit])
array([ True])
```

Sumário

Medindo Acurácia com Validação Cruzada

Uma boa maneira de avaliar um modelo é usar validação cruzada. **Acurácia**: proporção de previsões corretas.

```
from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.95035, 0.96035, 0.9604 ])
```

Uau! Acima de 95% de acurácia em todos os folds da validação cruzada. Parece ótimo!
Mas... vamos ver um classificador "burro" que simplesmente classifica toda imagem como "não-5".

```
from sklearn.dummy import DummyClassifier

dummy_clf = DummyClassifier()
dummy_clf.fit(X_train, y_train_5)
>>> cross_val_score(dummy_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([0.90965, 0.90965, 0.90965])
```

Conclusão

A acurácia geralmente não é a medida de desempenho ideal para classificadores, especialmente quando se lida com **datasets desbalanceados**.

Matriz de Confusão

Uma maneira muito melhor de avaliar o desempenho de um classificador é olhar a **matriz de confusão**.
A ideia geral é contar o número de vezes que instâncias da classe A são classificadas como classe B.
Primeiro, obtemos as previsões usando `cross_val_predict`:

```
from sklearn.model_selection import cross_val_predict

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

# Agora, podemos obter a matriz de confusão:
from sklearn.metrics import confusion_matrix
>>> cm = confusion_matrix(y_train_5, y_train_pred)
>>> cm                          #        Predito
array([[53892,   687],      # Real [[TN, FP]
       [ 1891,  3530]])     #       [FN, TP]]
```

- **Verdadeiros Negativos (TN)**: 53.892 não-5's corretamente classificados.
- **Falsos Positivos (FP)**: 687 não-5's incorretamente classificados como 5's.
- **Falsos Negativos (FN)**: 1.891 5's incorretamente classificados como não-5's.
- **Verdadeiros Positivos (TP)**: 3.530 5's corretamente classificados.

Matriz de Confusão Ilustrada

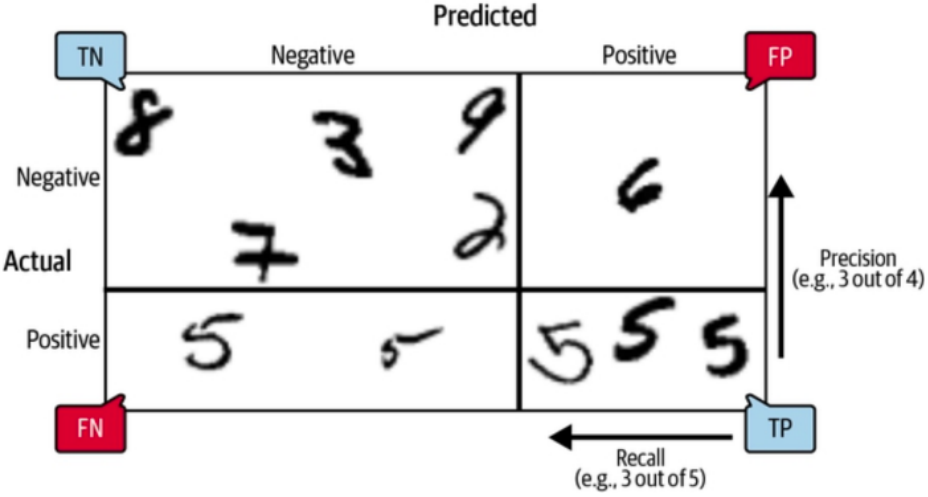


Figure: Uma matriz de confusão ilustrada mostrando exemplos de verdadeiros negativos (canto superior esquerdo), falsos positivos (canto superior direito), falsos negativos (canto inferior esquerdo) e verdadeiros positivos (canto inferior direito).

Precisão, Recall e F1 Score

Precisão

Acurácia das previsões positivas.
Responde: "Dentre todas as vezes que o modelo previu a classe positiva, qual a proporção de acertos?"

$$\text{precisão} = \frac{TP}{TP + FP}$$

Recall (Revocação ou Sensibilidade)

A proporção de instâncias positivas que são corretamente detectadas pelo classificador.
Responde: "Dentre as instâncias realmente positivas, qual a proporção de acertos?"

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

```
from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_train_pred)  # 0.837...
>>> recall_score(y_train_5, y_train_pred)     # 0.651...
```

Alto recall trivial: classificar tudo como positivo.
Alta precisão trivial: classificar como positivo apenas a instância de maior confiança.

F1 Score

A média harmônica da precisão e do recall.

$$F_1 = \frac{2 \times \text{precisão} \times \text{recall}}{\text{precisão} + \text{recall}}$$

Dá mais peso ao valor mais baixo, então o classificador só obterá um F1 Score alto se tanto o recall quanto a precisão forem altos.

```
from sklearn.metrics import f1_score
>>> f1_score(y_train_5, y_train_pred) # 0.732...
```

Trade-off Precisão/Recall

- Infelizmente, você não pode ter os dois: aumentar a precisão reduz o recall, e vice-versa. Isso é chamado de **trade-off precisão/recall**.
- A escolha depende do problema. Para detectar vídeos seguros para crianças, você prefere alta precisão (poucos vídeos ruins) e baixo recall (muitos vídeos bons rejeitados). Para detectar ladrões, você prefere alto recall (quase todos os ladrões pegos) e baixa precisão (alguns alarmes falsos).

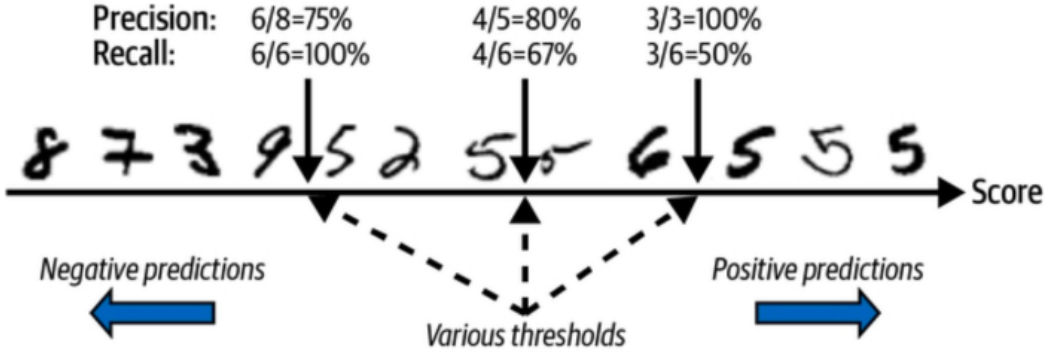


Figure: As imagens são classificadas por sua pontuação do classificador. As que estão acima do limiar de decisão escolhido são consideradas positivas; quanto maior o limiar, menor o recall, mas (em geral) maior a precisão.

Curvas de Precisão e Recall

O Scikit-Learn permite obter as pontuações usadas para fazer as previsões e, com elas, calcular a precisão e o recall para todos os limiares possíveis.

```
y_scores = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3,
                             method="decision_function")
from sklearn.metrics import precision_recall_curve
precisions, recalls, thresholds = precision_recall_curve(y_train_5, y_scores)
```

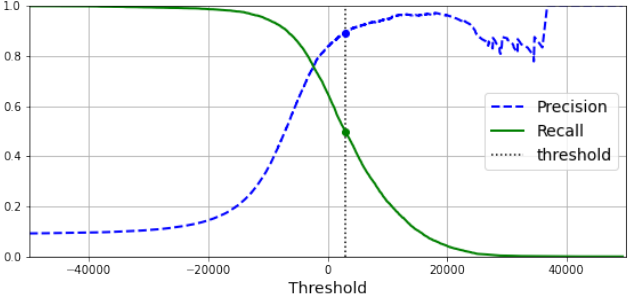


Figure: Precisão e recall versus o limiar de decisão.

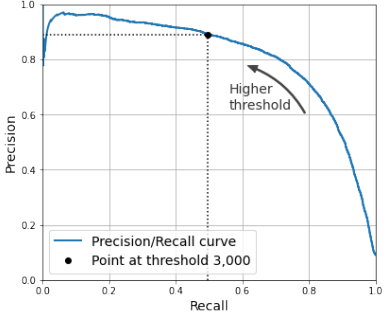


Figure: Precisão versus recall.

A Curva ROC

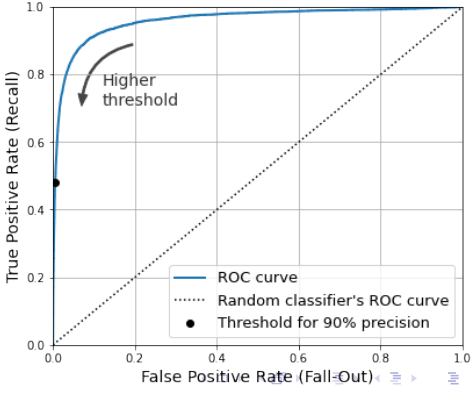
A curva **ROC (Receiver Operating Characteristic)** é outra ferramenta comum.
Ela plota a **taxa de verdadeiros positivos (TPR)** contra a **taxa de falsos positivos (FPR)**.

- TPR proporção de instâncias positivas classificadas como positivas (recall).
- FPR proporção de instâncias negativas classificadas como positivas.
- Um classificador perfeito terá uma **área sob a curva (AUC)** ROC igual a 1, enquanto um classificador puramente aleatório terá uma AUC ROC igual a 0.5.

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds =
    roc_curve(y_train_5, y_scores)

# ... código para plotar ...
```

```
from sklearn.metrics import roc_auc_score
>>> roc_auc_score(y_train_5, y_scores)
0.96049...
```



Comparando Modelos: SGD vs. Random Forest

Vamos treinar um `RandomForestClassifier` e comparar sua curva PR com a do `SGDClassifier`.

- `RandomForestClassifier` não tem um método `decision_function()`, mas tem `predict_proba()`. Podemos usar a probabilidade da classe positiva como pontuação.

```
from sklearn.ensemble import RandomForestClassifier
forest_clf = RandomForestClassifier(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]
precisions_forest, recalls_forest, _ =
    precision_recall_curve(y_train_5, y_scores_forest)
```

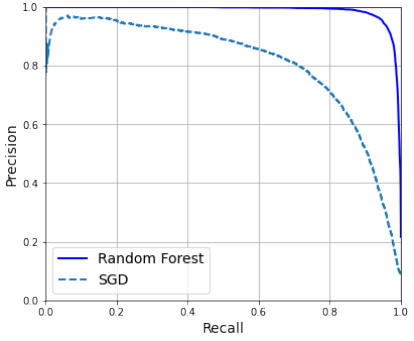


Figure: O RandomForestClassifier é superior.

O Random Forest tem um F1 Score de 0.924 e uma ROC AUC de 0.998. Muito melhor!

Sumário

Classificação Multiclasse

Classificadores multiclasse (ou multinomiais) podem distinguir entre mais de duas classes.

- Alguns algoritmos (como LogisticRegression, RandomForestClassifier) podem lidar com múltiplas classes nativamente.
- Outros (como SGDClassifier, SVC) são estritamente classificadores binários.

Estratégias para usar classificadores binários em tarefas multiclasse

- **One-versus-the-Rest (OvR) ou One-versus-All (OvA)**: Treinar 10 classificadores binários, um para cada dígito (um detector de 0, um detector de 1, etc.). Para classificar uma imagem, obtenha a pontuação de decisão de cada classificador e selecione a classe cujo classificador apresentar a maior pontuação.
- **One-versus-One (OvO)**: Treinar um classificador binário para cada par de dígitos (0 vs 1, 0 vs 2, 1 vs 2, etc.). Para o MNIST, isso significa treinar $\binom{10}{2} = 45$ classificadores. Para classificar uma imagem, execute-a através de todos os 45 classificadores e veja qual classe vence mais duelos.

O Scikit-Learn detecta quando você tenta usar um algoritmo de classificação binária para uma tarefa de classificação multiclasse e executa automaticamente OvR ou OvO, dependendo do algoritmo.

Sumário

Análise de Erros

Depois de encontrar um modelo promissor, uma maneira de melhorá-lo é analisar os tipos de erros que ele comete.

- Primeiro, olhe a matriz de confusão. Para tarefas multiclasse, é útil visualizá-la como uma imagem.

```
from sklearn.metrics import ConfusionMatrixDisplay

# ... obter y_train_pred do SGDClassifier treinado
# em todas as classes

# Plotar a matriz de confusão
ConfusionMatrixDisplay.from_predictions(y_train,
    y_train_pred)
plt.show()

# Plotar a matriz normalizada
ConfusionMatrixDisplay.from_predictions(y_train,
    y_train_pred, normalize="true", values_format=".0%")
plt.show()
```

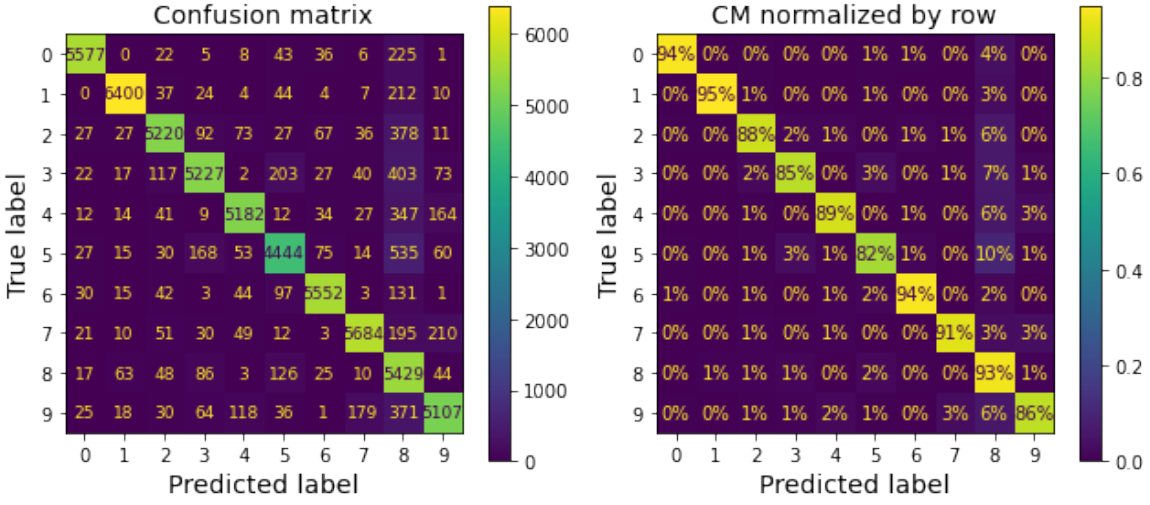


Figure: Matriz de confusão (esquerda) e normalizada (direita).

Focando nos Erros

Analisar a matriz de confusão pode fornecer insights. Por exemplo, normalizando pelos erros, podemos ver quais dígitos são mais confundidos com outros.

- Olhando para os gráficos, parece que nossos esforços deveriam ser gastos na redução dos falsos 8s.
- Poderíamos tentar coletar mais dados de treinamento para dígitos que se parecem com 8s (mas não são).
- Ou poderíamos criar novas features que ajudassem o classificador (ex: um algoritmo para contar o número de laços fechados).

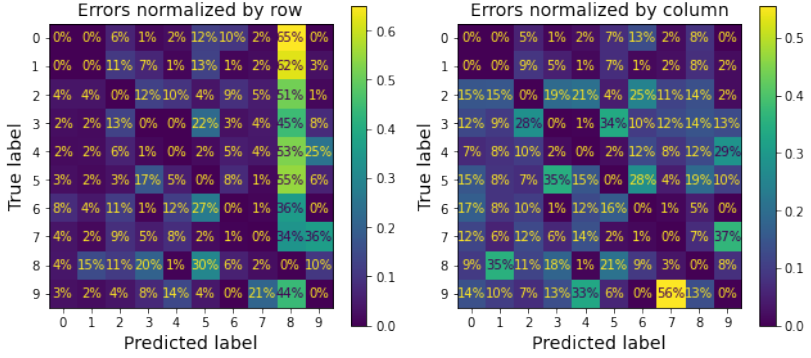


Figure: Matriz de confusão com apenas erros, normalizada por linha (esquerda) e coluna (direita).

Analisando Erros Individuais

- Analisar erros individuais também pode fornecer insights.
- O `SGDClassifier` é um modelo linear, então ele é sensível a deslocamento e rotação de imagens.
- Uma forma de reduzir a confusão 3/5 seria pré-processar as imagens para garantir que estejam bem centralizadas e não muito rotacionadas.
- Uma abordagem mais simples é aumentar o conjunto de treinamento com variantes ligeiramente deslocadas e rotacionadas das imagens. Isso é chamado de **aumento de dados (data augmentation)**.

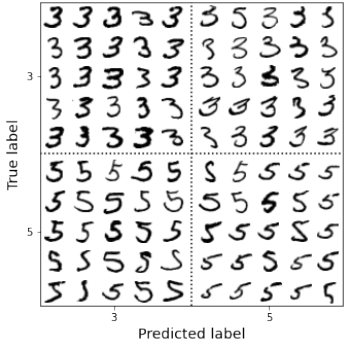


Figure: Exemplos de 3s e 5s que o classificador confunde.

Sumário

Classificação Multilabel e Multioutput

Classificação Multilabel

Um sistema de classificação que produz múltiplas tags binárias. Por exemplo, um classificador de reconhecimento facial que identifica várias pessoas na mesma foto deve anexar uma tag por pessoa.
Ex.: A saída para uma foto de Alice e Charlie seria [True, False, True] para [Alice, Bia, Charlie].

Classificação Multioutput

Uma generalização da classificação multilabel onde cada rótulo pode ser multiclasse (pode ter mais de dois valores possíveis).

Exemplo: Remoção de Ruído de Imagens

- Entrada: uma imagem de dígito com ruído.
- Saída: uma imagem de dígito limpa.
- A saída é multioutput: uma tag por pixel, e 256 classes por tag (intensidade do pixel de 0 a 255).

Figure: Uma imagem ruidosa e o alvo limpo.

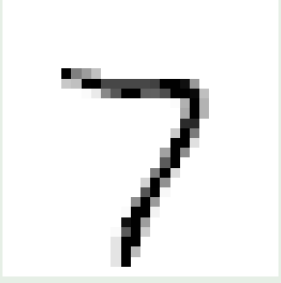


Figure: A imagem limpa pelo modelo.

Fim