Universidade Federal Fluminense

TÓPICOS ESP. SIST. INFORMAÇÃO

Prof.ªLeila Weitzel

3. Treinamento de Redes

Etapas do processo de classificação

- Etapas de um projeto de classificação com redes deep learning
- **Coleta de Dados:**
  - Comece coletando os dados relevantes para o seu problema. Isso pode envolver aquisição de dados de fontes externas, criação de conjuntos de treinamento e teste, e garantir que os dados estejam limpos e bem estruturados.
- **Codificação dos Rótulos:**
  - Os rótulos (classes) dos dados precisam ser codificados numericamente.
    - Exemplo: Gato = 0, Cachorro = 1.
- **Pré-processamento dos Dados:**
  - Nesta etapa, prepare os dados para alimentar a rede neural. Isso inclui normalização, tratamento de valores ausentes, codificação de rótulos, divisão em conjuntos de treinamento e teste, etc.

Etapas do processo de classificação

- **Divisão dos Dados:**
- Separamos os dados em conjuntos de treinamento, validação e teste.
- O conjunto de treinamento é usado para treinar o modelo, o conjunto de validação é usado para ajustar hiperparâmetros e o conjunto de teste é usado para avaliar o desempenho final.
  - Exemplo: em 80% treinamento, 10% validação e 10% teste.
- **Construção do Modelo:**
  - Escolha a arquitetura da rede neural. Pode ser uma rede totalmente conectada (feedforward), uma rede convolucional (CNN) ou uma rede recorrente (RNN), dependendo do tipo de dados e do problema.
- **Montagem das Camadas:**
  - Empilhe as camadas da rede neural. Isso envolve a definição das camadas de entrada, ocultas e de saída. Cada camada extrai representações dos dados inseridos na rede.

Etapas do processo de classificação

- Etapas de um projeto de classificação com redes deep learning
- **Compilação do Modelo:**
  - Configure o modelo com uma função de perda, otimizador e métricas de avaliação. A função de perda mede o quão bem o modelo está performando, o otimizador ajusta os pesos da rede e as métricas avaliam o desempenho.
- **Treinamento do Modelo:**
  - Alimente os dados de treinamento no modelo e ajuste os pesos iterativamente. Isso é feito através de backpropagation e otimização.

Etapas do processo de classificação

- Etapas de um projeto de classificação com redes deep learning
- **Avaliação do Modelo:**
  - Use os dados de teste para avaliar a precisão do modelo. Isso ajuda a entender como o modelo generaliza para dados não vistos.
- **Ajustes e Otimizações:**
  - Com base nos resultados da avaliação, faça ajustes no modelo, como alterar hiperparâmetros, adicionar regularização ou modificar a arquitetura.
- **Implantação:**
  - Finalmente, implante o modelo treinado em produção para uso prático

Coleta dos dados

- A coleta de dados é um processo fundamental em pesquisas e projetos científicos.
- Ela visa reunir informações relevantes para análise e tomada de decisões. O processo ocorre da seguinte forma:
    1. Definição do Objetivo
    2. Seleção das Fontes de Dados
    3. Planejamento e Ferramentas
    4. Coleta dos Dados
    5. Tratamento e Análise
    6. Continuidade e Periodicidade

Coleta dos dados

1. Definição do Objetivo:

–Antes de coletar dados, é crucial definir o objetivo da pesquisa. O que você deseja investigar ou entender?

- Exemplo: Se estamos estudando o impacto de um novo medicamento, nosso objetivo é coletar dados sobre sua eficácia e efeitos colaterais.

Coleta dos dados

2. Seleção das Fontes de Dados:

– Identificamos as fontes de onde obteremos os dados. Isso pode incluir:

- Fontes Documentais: Dados de documentos, relatórios, registros, etc.
- Observação Direta: Coleta de dados por meio de observação pessoal.
- Entrevistas: Conversas com pessoas relevantes para o estudo.
- Questionários: Perguntas estruturadas para os participantes.
- Exemplo: Se estamos estudando o comportamento de consumidores, podemos usar questionários online ou entrevistas presenciais.

Coleta dos dados

3. Planejamento e Ferramentas:

– Planejamos como coletar os dados. Isso envolve escolher as ferramentas adequadas, como questionários, softwares de pesquisa, etc.

- Exemplo: Criar um questionário online para coletar opiniões sobre um novo produto.

Coleta dos dados

4. Coleta dos Dados:

- Realizamos a coleta conforme o planejado. Isso pode ser feito por meio de entrevistas, questionários, observação direta ou análise de documentos.
  - Exemplo: Entrevistar pacientes para entender sua experiência com o medicamento.

Coleta dos dados

5. Tratamento e Análise:

- Os dados coletados são organizados, limpos e preparados para análise.
- Podemos usar técnicas estatísticas para extrair insights relevantes.
  - Exemplo: Calcular a média de satisfação dos pacientes com base nas respostas do questionário.

Coleta dos dados

6. Continuidade e Periodicidade:

– A coleta de dados pode ser contínua (registrando eventos à medida que ocorrem), periódica (ciclos regulares, como um censo) ou ocasional (sem continuidade específica).

- Exemplo: Monitorar diariamente as vendas de um produto (coleta contínua).

Coleta dos dados

- Lembre-se de que a qualidade dos dados coletados afeta diretamente a confiabilidade das conclusões da pesquisa.
- Portanto, é essencial seguir boas práticas e métodos adequados para obter resultados significativos.

Etapas da classificação com Python

- **Passos:**

1. Importar bibliotecas
2. Carregar os dados
3. Separar em entrada e saída caso necessário
4. Particionar a base de dados dividir a base em treinamento (80%) e teste (20%).
5. Criar o modelo de rede
6. Treinar o modelo (função fit()) Avaliar o modelo (acurácia, matriz de confusão etc...)
7. Compilar o modelo : decidir quais hiperparametros usar
8. Fazer predições y_pred().

Etapas da classificação com Python

1. Importar as bibliotecas é a primeira seção do notebook:
   - Numpy
   - Keras
   - Pandas

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
```

```
1  from keras.preprocessing.text import Tokenizer
2  from keras.preprocessing.sequence import pad_sequences
3  from keras.models import Sequential
4  from keras.layers import Dense, Embedding, Flatten
5  from keras.utils import to_categorical
6  from sklearn.model_selection import train_test_split
7  import pandas as pd
8  import numpy as np
9  from sklearn.preprocessing import LabelEncoder
10 from keras.utils import to_categorical
```

Etapas da classificação com Python

2. Carregando ou lendo os dados; segunda seção do notebook

```
# Carregue a base de dados Pima Indians Diabetes
dataset = loadtxt('https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv',
```

```
df = pd.read_csv('data.csv')
```

Etapas da classificação com Python

3. Separando os dados em entrada (X) e saída (y):

- Por convenção o X é sempre em maiúsculo e y em minúsculo.
- Explicação no próximo slide

```
# Separe os dados entre entradas (X) e saídas (y)
X = dataset[:, 0:8]
y = dataset[:, 8]
```

```
# Separe os dados entre entradas (X) e saídas (y)
X = dataset[:, 0:8]
y = dataset[:, 8]
```

X = dataset[:, 0:8]:

- X é uma variável que estamos criando para armazenar parte dos dados do conjunto de dados. dataset é o conjunto de dados carregado anteriormente.
- Os dois pontos : indicam que queremos selecionar todas as linhas do conjunto de dados.
- O 0:8 especifica que queremos selecionar as colunas da 0ª à 7ª (pois a contagem começa em 0).

Portanto, X conterá todas as linhas e as primeiras 8 colunas do conjunto de dados.

y = dataset[:, 8]:

- y é outra variável que estamos criando para armazenar outra parte dos dados do conjunto de dados. Novamente, dataset é o conjunto de dados.
- O : indica que queremos selecionar todas as linhas.
- O 8 especifica que queremos selecionar apenas a 8ª coluna.

Portanto, y conterá todas as linhas da 8ª coluna do conjunto de dados.

Etapas da classificação com Python

4. Particionando a base em treinamento e teste

- ***train_test_Split:***
  - A função train_test_split do scikit-learn é amplamente utilizada para dividir um conjunto de dados em subconjuntos de treinamento e teste.
  - Ela permite controlar vários parâmetros para personalizar a divisão.

```
[ ]  1  # Dividindo os dados em treinamento e teste
     2  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Etapas da classificação com Python

4. Particionando a base em treinamento e teste

– Parâmetros Principais:

- **test_size:** Define o tamanho do conjunto de teste. Pode ser um valor entre 0.0 e 1.0 (proporção) ou um número inteiro (número absoluto de amostras).
- **train_size:** Define o tamanho do conjunto de treinamento. Se não especificado, é automaticamente definido como o complemento do tamanho do conjunto de teste.

Etapas da classificação com Python

4. Particionando a base em treinamento e teste
   - **Parâmetros Principais:**
     - **random_state:**
       - Tem um papel importante na reprodutibilidade dos resultados. É um valor numérico que controla a aleatoriedade em algoritmos que envolvem geração de números aleatórios. Usado para inicializar o gerador de números aleatórios antes de executar o algoritmo. A aleatoriedade é comum em algoritmos de aprendizado de máquina, como divisão de dados em treinamento e teste, inicialização de pesos em redes neurais, embaralhamento de dados, etc. Definir um valor específico para o random_state permite que os resultados sejam reproduzíveis.
     - **shuffle:**
       - Determina se os dados devem ser embaralhados antes da divisão. Por padrão, é verdadeiro.
     - **stratify:**
       - Usado para estratificação. Se fornecido, garante que as proporções das classes sejam mantidas nos conjuntos de treinamento e teste.

Etapas da classificação com Python

5. Criar o modelo
   - Estamos criando um modelo sequencial
   - Explicação no próximo slide
   - Não vamos entrar nos detalhes de cada camada agora.

```python
# Inicialize o modelo sequencial
model = Sequential()

# Adicione uma camada com 12 neurônios e função de ativação ReLU
model.add(Dense(12, input_dim=8, activation='relu'))

# Adicione outra camada com 8 neurônios e função de ativação ReLU
model.add(Dense(8, activation='relu'))

# Camada de saída com função de ativação sigmoid (para classificação binária)
model.add(Dense(1, activation='sigmoid'))
```

Etapas da classificação com Python

- **Modelo Sequencial:**
  - O modelo sequencial é o mais simples e comumente usado no Keras.
  - Ele permite criar redes neurais empilhando camadas uma após a outra.
  - Cada camada é conectada à camada anterior, formando uma sequência linear.
  - É adequado para arquiteturas de rede simples, como classificação e regressão.
- **Modelo Funcional:**
  - O modelo funcional é mais flexível e poderoso.
  - Permite criar redes com múltiplas entradas, saídas e caminhos de conexão complexos.
  - É útil para arquiteturas não lineares, como redes com bifurcações, compartilhamento de camadas e modelos com várias saídas.

Etapas da classificação com Python

- **Modelo Sequencial:**
  - Usando o Keras

```python
# Inicialize o modelo sequencial
model = Sequential()

# camada de entrada
model.add(Input(shape=(8,)))

# Adicione uma camada com 12 neurônios e função de ativação ReLU
model.add(Dense(12, activation='relu'))

# Adicione outra camada com 8 neurônios e função de ativação ReLU
model.add(Dense(8, activation='relu'))

# Camada de saída com função de ativação sigmoid (para classificação binária)
model.add(Dense(1, activation='sigmoid'))
```

- **Modelo funcional com**
  - uma camada de incorporação (embeddings),
  - duas camadas LSTM bidirecionais,
  - uma camada densa e
  - uma camada de saída.
- O modelo é ideal classificação binária (análise de sentimento).
- Ideal quando você precisa de mais flexibilidade em sua arquitetura de rede neural

```python
import tensorflow as tf
from tensorflow.keras.layers import Input, Embedding, Bidirectional, LSTM, Dense, Dropout

# Defina a entrada
input_text = Input(shape=(100,))  # Tamanho do vetor de entrada

# Camada de incorporação
embedding_layer = Embedding(input_dim=512, output_dim=64, input_length=100)(input_text)

# Camadas LSTM bidirecionais
lstm1 = Bidirectional(LSTM(64, return_sequences=True))(embedding_layer)
lstm2 = Bidirectional(LSTM(64))(lstm1)

# Camada densa
dense_layer = Dense(32, activation='relu')(lstm2)

# Camada de saída
output = Dense(1, activation='sigmoid')(dense_layer)

# Crie o modelo funcional
model = tf.keras.Model(inputs=input_text, outputs=output)

# Compile o modelo
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Exiba o resumo do modelo
model.summary()
```

Etapas da classificação com Python

- **Observações:**
- O modelo sequencial é fácil de usar e adequado para muitos cenários.
- O modelo funcional é mais versátil e permite criar arquiteturas personalizadas.
- A escolha entre eles depende da complexidade do problema e dos requisitos do projeto.

Etapas da classificação com Python

6. Treinar o modelo

```
model.fit(X_train, y_train, epochs=100, batch_size=10):
```

- **X_train:**
  - É o conjunto de dados de treinamento (recursos).
  - Contém as entradas para o modelo.
- **y_train:**
  - É o conjunto de rótulos correspondentes aos dados de treinamento.
  - Contém as saídas esperadas para o modelo.
- **epochs:**
  - Número de vezes que o modelo percorre todo o conjunto de treinamento.
  - Cada época atualiza os pesos do modelo com base no erro calculado.

Etapas da classificação com Python

6. Treinar o modelo

```
model.fit(X_train, y_train, epochs=100, batch_size=10)
```

- **batch_size:**
  - define o número de amostras que serão propagadas pela rede em cada iteração (ou época) durante o treinamento. Em outras palavras, é o tamanho do lote de dados usado para atualizar os pesos da rede.
  - O modelo calcula o gradiente com base em um lote de dados e ajusta os pesos.
  - **Como Funciona**:
    - Durante o treinamento, os dados são divididos em lotes (ou mini-lotes).
    - Cada lote é usado para calcular o gradiente e ajustar os pesos da rede.
    - O processo é repetido para todos os lotes até que todas as amostras tenham sido processadas.
  - **Exemplo:**
    - Suponha que você tenha 1050 amostras de treinamento e defina batch_size=100.
    - O algoritmo usará os primeiros 100 exemplos para treinar a rede, depois os próximos 100 e assim por diante.
    - Se o número total de amostras não for divisível pelo tamanho do lote, o último lote pode ser menor.

Etapas da classificação com Python

6. Treinar o modelo

```
model.fit(X_train, y_train, epochs=100, batch_size=10)
```

- **batch_size:**
  - **Vantagens**:
    - **Economia de Memória**: Usar um tamanho de lote menor requer menos memória, o que é importante quando o conjunto de dados não cabe na memória do computador.
    - **Treinamento Mais Rápido**: Atualizar os pesos após cada lote (em vez de após cada amostra) acelera o treinamento.
  - **Desvantagens**:
    - **Estimativa Menos Precisa do Gradiente**: Quanto menor o tamanho do lote, menos precisa é a estimativa do gradiente.
    - **Flutuações no Treinamento**: Lotes menores podem levar a flutuações mais intensas nos gradientes.

Etapas da classificação com Python

7. Compilar o modelo (mais para frente detalharemos)
   - **loss="binary_crossentropy"**: Esta é a função de perda (loss function) utilizada para calcular a diferença entre as previsões do modelo e os rótulos (labels) verdadeiros durante o treinamento. Neste caso, a função de perda é a entropia cruzada binária, que é comumente usada em problemas de classificação binária, onde cada exemplo de entrada pertence a uma das duas classes.
   - **optimizer="adam"**: O otimizador é o algoritmo utilizado para ajustar os pesos da rede neural durante o treinamento, com o objetivo de minimizar a função de perda. "adam" se refere ao otimizador Adam, que é um método de otimização popular e eficiente, especialmente para problemas de deep learning.
   - **metrics=["accuracy"]:** As métricas são utilizadas para avaliar o desempenho do modelo durante o treinamento, mas não são usadas para otimização. Neste caso, a métrica utilizada é a acurácia, que mede a proporção de exemplos classificados corretamente pelo modelo.

```
# Compilando o modelo
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Etapas da classificação com Python

8. Fazer predições

- Neste exemplo, X_new é um array numpy contendo os dados de entrada para os quais você deseja fazer a previsão.
- No exemplo, X_new contém apenas um conjunto de dados, mas você pode passar vários conjuntos de dados de uma vez se desejar fazer previsões para múltiplos exemplos.

```python
import numpy as np

# Dados de entrada para fazer a previsão
X_new = np.array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])

# Fazer a previsão
predictions = model.predict(X_new)

# Arredondar a previsão para obter a classe (0 ou 1)
predicted_class = np.round(predictions[0])

# Imprimir a previsão
print("Previsão:", predicted_class)
```

- O método predict retorna um array numpy com as previsões do modelo para cada exemplo de entrada. Como estamos lidando com um problema de classificação binária, usamos np.round(predictions[0]) para arredondar a previsão para a classe mais provável (0 ou 1). Note que predictions é uma matriz 2D, então predictions[0] seleciona a previsão para o primeiro (e único) exemplo em X_new.
- Finalmente, o código imprime a previsão, que será 0 ou 1, dependendo da entrada fornecida e do modelo treinado.

See You
Next time