Sistemas Inteligentes – Aula 4

Hoje vamos implementar um modelo básico de aprendizado de máquina, utilizando Python, para classificação de emojis com base nos sentimentos associados.

Para facilitar o processo, iremos utilizar o Google Colab, um ambiente de desenvolvimento online que já apresenta as configurações necessárias.

Acesse https://colab.research.google.com/ e clique no botão "+ Novo notebook", no canto inferior esquerdo.

Para adicionar o código, clique no botão "+ Código" e coloque os códigos no seu ambiente de desenvolvimento (não coloque as explicações):

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
import random
```

- import pandas as pd:
  - Importa a biblioteca pandas, que é usada para trabalhar com dados estruturados (como tabelas).
- from sklearn.model_selection import train_test_split:
  - Importa a função que serve para dividir o conjunto de dados em dois grupos: um para treinar o modelo e outro para testar o desempenho do modelo.
- from sklearn.tree import DecisionTreeClassifier:
  - Importa o classificador de árvore de decisão. Esse é o algoritmo de aprendizado de máquina que usaremos para prever o sentimento baseado no emoji.
- from sklearn.preprocessing import LabelEncoder:
  - Importa o LabelEncoder, que é uma ferramenta para codificar dados categóricos (como emojis e sentimentos) em números, o que facilita para o modelo entender os dados.

```
dados = {
    'emoji': ['😄', '😄', '😖', '😖', '😵', '😵', '😎', '😎', '😤', '😤',
              '😵', '😵', '😕', '😕', '😂', '😂', '🥲', '🥲', '😭', '😭',
              '🤪', '🤪', '😳', '😳', '🤤', '🤤', '😊', '😊', '😱', '😱',
              '😵', '😵', '😇', '😇', '😌', '😌', '😴', '😴', '😭', '😭',
              '😄', '😄', '😖', '😖', '😵', '😵', '😎', '😎', '😤', '😤',
              '😵', '😵', '😕', '😕', '😂', '😂', '🥲', '🥲', '😭', '😭'],
    'sentimento': ['alegre', 'alegre', 'impaciente', 'impaciente', 'triste', 'triste',
                   'relaxado', 'relaxado', 'estressado', 'estressado', 'irritado', 'irritado',
                   'confuso', 'confuso', 'feliz', 'feliz', 'sonolento', 'sonolento',
                   'chorando', 'chorando', 'animado', 'animado', 'desanimado', 'desanimado',
                   'divertido', 'divertido', 'contente', 'contente', 'pensativo', 'pensativo',
                   'preocupado', 'preocupado', 'calmo', 'calmo', 'brincalhão', 'brincalhão',
                   'cansado', 'cansado', 'chorando', 'chorando', 'alegre', 'alegre',
                   'impaciente', 'impaciente', 'triste', 'triste', 'relaxado', 'relaxado',
```

```
                  'estressado', 'estressado', 'irritado', 'irritado', 'confuso', 'confuso',
                  'feliz', 'feliz', 'sonolento', 'sonolento', 'chorando', 'chorando']
}
df = pd.DataFrame(dados)
```

- `dados = {...}`:
  - Aqui estamos criando um dicionário onde cada chave contém uma lista de valores.
  - A chave `emoji` contém uma lista de emojis, e a chave `sentimento` contém uma lista de sentimentos correspondentes.
  - Repetimos os emojis e sentimentos para garantir que cada classe tenha pelo menos 2 exemplos.
- `df = pd.DataFrame(dados)`:
  - Convertemos o dicionário em um DataFrame, que é uma tabela onde podemos facilmente manipular e acessar nossos dados.

```
encoder_emoji = LabelEncoder()
encoder_sentimento = LabelEncoder()

df['emoji_codificado'] = encoder_emoji.fit_transform(df['emoji'])
df['sentimento_codificado'] = encoder_sentimento.fit_transform(df['sentimento'])
```

- `encoder_emoji = LabelEncoder()` e `encoder_sentimento = LabelEncoder()`:
  - O `LabelEncoder` transforma os dados em números para que o modelo de aprendizado de máquina possa trabalhar com eles.
  -

```
X = df[['emoji_codificado']]
y = df['sentimento_codificado']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=19, random_state=42,
stratify=y)
```

- `X = df[['emoji_codificado']]`:
  - Pega os números que representam os emojis (entrada para o modelo).
- `y = df['sentimento_codificado']`:
  - Pega os números que representam os sentimentos (o que o modelo deve prever).
- `X_train, X_test, y_train, y_test = train_test_split(...)`:
  - Divide os dados:
    i. **Treino (`X_train`, `y_train`)**: usado para ensinar o modelo.
    ii. **Teste (`X_test`, `y_test`)**: usado para verificar se o modelo aprendeu bem.
  - `test_size=19`: mantém 19 exemplos no conjunto de teste.
  - `random_state=42`: garante que a divisão seja sempre igual.
  - `stratify=y`: garante que as classes (sentimentos) fiquem balanceadas no treino e no teste.

```
for i in range(5):
    idx = random.randint(0, len(y_train) - 1)
    y_train.iloc[idx] = random.choice(y_train.unique())
```

- Selecionamos 5 índices aleatórios no conjunto de treinamento e mudamos os rótulos para valores aleatórios, criando inconsistências e simulando um ambiente realista. Como o

conjunto de treinamento não é mais perfeito, o modelo enfrentará maior dificuldade para identificar corretamente os padrões, reduzindo a acurácia no conjunto de teste.

```
modelo = DecisionTreeClassifier()
modelo.fit(X_train, y_train)
```

- `modelo = DecisionTreeClassifier()`:
  - Aqui estamos criando o modelo de árvore de decisão. Esse modelo é muito bom para classificação e decide qual é o melhor "ramos" (decisões) a serem seguidos para prever uma classe.
- `modelo.fit(X_train, y_train)`:
  - O `fit` treina o modelo com os dados de treinamento (`X_train` e `y_train`). O modelo aprende a relação entre os emojis e seus sentimentos correspondentes.

```
previsoes = modelo.predict(X_test)
previsoes_texto = encoder_sentimento.inverse_transform(previsoes)
realidade_texto = encoder_sentimento.inverse_transform(y_test)
```

- `previsoes = modelo.predict(X_test)`:
  - O `predict` faz com que o modelo use o que aprendeu no treinamento para prever os sentimentos dos emojis no conjunto de teste (`X_test`).
- `previsoes_texto = encoder_sentimento.inverse_transform(previsoes)`:
  - `inverse_transform` converte os números de volta para os sentimentos originais. Ou seja, se a previsão for `0`, ela será convertida para "alegre", e assim por diante.
- `realidade_texto = encoder_sentimento.inverse_transform(y_test)`:
  - Da mesma forma, a realidade (o valor verdadeiro) dos sentimentos no conjunto de teste é convertida de volta para os nomes dos sentimentos.

```
resultado = pd.DataFrame({
    'emoji_testado': encoder_emoji.inverse_transform(X_test['emoji_codificado']),
    'sentimento_real': realidade_texto,
    'sentimento_previsto': previsoes_texto
})

print("Resultados do teste:")
print(resultado)
acuracia = (previsoes == y_test).mean() * 100
print(f"\nAcurácia do modelo: {acuracia:.2f}%")
```

- `print("Resultados do teste:")` e `print(resultado)`:
  - Exibimos os resultados para que possamos ver o desempenho do modelo.
- `acuracia = (previsoes == y_test).mean() * 100`:
  - Calculamos a acurácia do modelo. Ela representa a porcentagem de previsões corretas. Comparamos as previsões com as reais e calculamos a média de acertos.
- `print(f"\nAcurácia do modelo: {acuracia:.2f}%")`:
  - Exibimos a acurácia do modelo formatada com duas casas decimais.

Ao executar seu código, você pode ver quais sentimentos são relacionados com cada emoji, e se a previsão foi correta, com uma porcentagem de acertos sendo mostrada abaixo dos dados, variando a cada vez que você executa.

Agora é com você:

1. Altere esse código para que a acurácia do sistema caia para aproximadamente 50%.
2. Altere esse código para que a acurácia do sistema suba para 100%.
3. Pesquise como inserir seeds no código e faça a alteração, para que os resultados sejam sempre fixos.
4. Altere os dados para que, ao invés de emotes, os dados sejam compostos por outro tipo de informação à sua escolha.