Java e Generative AI

Generative AI com Java refere-se ao desenvolvimento de soluções de inteligência artificial (IA) que sejam amplas e aplicáveis em diferentes contextos, sem depender de um problema ou conjunto de dados específico.

A análise de sentimento com Naive Bayes e WEKA em Java envolve usar o algoritmo de Naive Bayes para classificar textos como positivos, negativos ou neutros.
Vamos explorar o Aprendizado supervisionado com Naive Bayes, que é mais uma abordagem de machine learning tradicional.

Resumindo de maneira didática:

✓ Primeiro, você precisa de um conjunto de textos já classificados como "positivo", "negativo" ou "neutro" (por exemplo, avaliações de produtos).
Esses textos são organizados em um formato específico que o WEKA entende, chamado ARFF.
✓ Carregar os Dados no WEKA, no Java, usamos a biblioteca WEKA para carregar os textos e seus sentimentos associados.
O WEKA permite que esses textos sejam transformados em algo que o computador possa analisar, usando uma tabela onde cada linha é um texto e a última coluna é o sentimento (positivo ou negativo).
✓ Com os dados prontos, agora é treinar o modelo , você usa o algoritmo Naive Bayes para aprender a diferença entre textos positivos e negativos.
O Naive Bayes é um método simples que calcula a probabilidade de um texto ser positivo ou negativo, com base nas palavras que aparecem.
✓ Para saber se o modelo aprendeu bem, você faz uma "validação cruzada", iso significa dividir o conjunto de dados em partes, treinando o modelo em uma parte e testando na outra, verificando o quão preciso ele é ao prever o sentimento de novos textos.
✓ Depois que o modelo está treinado e avaliado, ele pode ser usado para classificar novos textos.
Você fornece um texto, e o modelo prevê se o sentimento desse texto é positivo, negativo ou neutro.

A ideia é criar modelos ou algoritmos capazes de aprender e se adaptar a diferentes cenários, utilizando técnicas de aprendizado de máquina (Machine Learning), redes neurais, ou outros métodos de IA, e implementá-los utilizando a linguagem Java.

Exemplo simples e didático:
Imagine que você deseja criar um modelo de IA para classificar mensagens de texto como "positivas" ou "negativas". Esse tipo de problema pode ser generalizado para diferentes aplicações de classificação de texto, como filtrar e-mails, analisar opiniões de clientes, etc.
Vamos usar a biblioteca Weka, que é uma API popular de aprendizado de máquina em Java, para este exemplo.

Passos para o exemplo:
Usaremos o algoritmo Naive Bayes, um classificador comum em aprendizado de máquina, para treinar um modelo que possa classificar sentimentos em textos.

Treinaremos o modelo com uma pequena quantidade de dados de texto já categorizados como "positivo" ou "negativo". EducaCiência FastCode

Código simplificado: java

```
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;

public class SentimentAnalysis {
   public static void main(String[] args) throws Exception {
      // Carregar o conjunto de dados (arquivo .arff com textos rotulados)
      DataSource source = new DataSource("sentiment_data.arff");
      Instances dataset = source.getDataSet();

      // Definir qual atributo será classificado (último no conjunto de dados)
      dataset.setClassIndex(dataset.numAttributes() - 1);

      // Criar e treinar o classificador NaiveBayes
      NaiveBayes classifier = new NaiveBayes();
      classifier.buildClassifier(dataset);

      // Exemplo de nova instância para classificar
      Instance newText = dataset.get(0); // Simulando um novo texto para classificação
      double label = classifier.classifyInstance(newText);

      // Exibir resultado da classificação
      System.out.println("Classificação: " + dataset.classAttribute().value((int) label));
   }
}
```

Explicação:
• **DataSource**: carrega um conjunto de dados contendo mensagens de texto com rótulos "positivo" ou "negativo".
• **NaiveBayes**: é o algoritmo de aprendizado de máquina usado para criar o modelo de classificação.
• **classifyInstance**: classifica uma nova instância de texto e retorna o rótulo predito (positivo ou negativo).

Este exemplo simples mostra como você pode generalizar o uso de IA em Java para um problema comum, como a análise de sentimentos, mas o mesmo conceito pode ser adaptado para diferentes tipos de classificação, como diagnóstico médico, filtragem de spam, etc.

Para tornar o código mais didático e mostrar como testá-lo, vou adicionar algumas mensagens com System.out.println para explicar cada etapa do processo e mostrar o que está acontecendo no código em tempo real.

Agorq , explicarei o que você precisa fazer para realizar o teste.

Versão com explicações no código: java

```
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instance;
import weka.core.Instances;
```

```
import weka.core.converters.ConverterUtils.DataSource;

public class SentimentAnalysis {
    public static void main(String[] args) throws Exception {
        // Carregar o conjunto de dados (arquivo .arff com textos rotulados)
        System.out.println("Carregando o conjunto de dados...");
        DataSource source = new DataSource("sentiment_data.arff");
        Instances dataset = source.getDataSet();
        System.out.println("Conjunto de dados carregado com sucesso!");

        // Definir qual atributo será classificado (último no conjunto de dados)
        System.out.println("Definindo o atributo que será classificado...");
        dataset.setClassIndex(dataset.numAttributes() - 1);
        System.out.println("Atributo de classe definido: " + dataset.classAttribute().name());

        // Criar e treinar o classificador NaiveBayes
        System.out.println("Treinando o modelo NaiveBayes...");
        NaiveBayes classifier = new NaiveBayes();
        classifier.buildClassifier(dataset);
        System.out.println("Modelo treinado com sucesso!");

        // Exemplo de nova instância para classificar
        System.out.println("Classificando um novo exemplo de texto...");
        Instance newText = dataset.get(0); // Simulando um novo texto para classificação
        double label = classifier.classifyInstance(newText);

        // Exibir resultado da classificação
        System.out.println("Texto: " + newText);
        System.out.println("Classificação prevista: " + dataset.classAttribute().value((int) label));
    }
}
```

Como testar:
Obtenha um arquivo.arff: O Weka trabalha com arquivos no formato .arff para conjuntos de dados.
Você pode criar um arquivo simples contendo exemplos de mensagens de texto rotuladas como "positivo" ou "negativo".

Por exemplo: Arff

```
@relation sentiment_analysis

@attribute text string
@attribute class {positivo, negativo}

@data
"Hoje o dia está maravilhoso", positivo
"Estou muito triste com o que aconteceu", negativo
"Que ótimo trabalho você fez", positivo
"Não estou satisfeito com o serviço", negativo
```

Incluir o Weka no seu projeto: Você precisará adicionar o Weka ao seu projeto Java.

Se estiver utilizando uma IDE como NetBeans ou IntelliJ, você pode baixar a biblioteca Weka e adicioná-la ao seu build path:

Baixe o Weka: Link para download do Weka - https://www.cs.waikato.ac.nz/ml/weka
✓ https://docs.weka.io/planning-and-installation/bare-metal/obtaining-the-weka-install-file

Adicione o arquivo .jar à sua biblioteca de projetos.

Executar o código: Compile e execute o código. Ele irá carregar o conjunto de dados, treinar o classificador NaiveBayes, e então classificará uma nova instância (no exemplo, ele classifica o primeiro texto no conjunto de dados).

Observação de resultados: Após a execução, você verá as mensagens System.out.println no console, explicando cada etapa e exibindo o texto e a classificação feita pelo modelo.

Saída esperada: No console, você verá algo assim: yaml
Carregando o conjunto de dados...
Conjunto de dados carregado com sucesso!
Definindo o atributo que será classificado...
Atributo de classe definido: class
Treinando o modelo NaiveBayes...
Modelo treinado com sucesso!
Classificando um novo exemplo de texto...
Texto: Hoje o dia está maravilhoso
Classificação prevista: positivo

Teste no Netbeans

Vou realizar o teste no Netbeans para facilitar seu entendimento , porém , farei alguns ajustes técnicos para facilitar nosso output

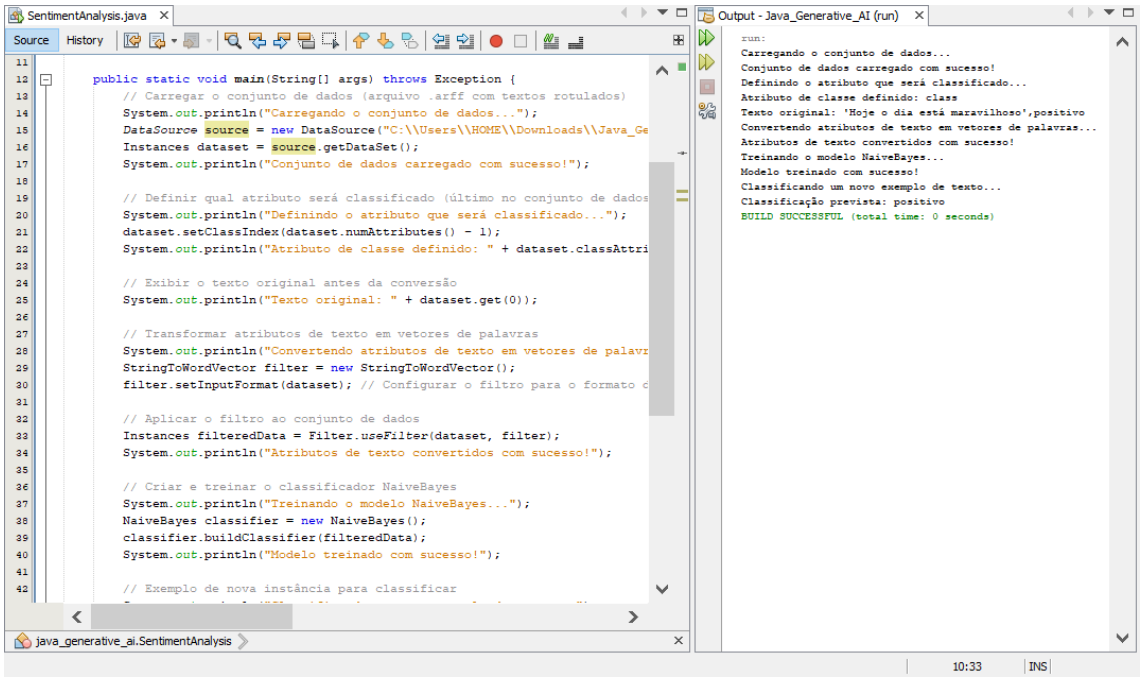
- *NaiveBayes não pode lidar diretamente com atributos do tipo String.*
  Para resolver esse problema, precisamos converter os atributos String em um formato numérico (ou transformá-los em vetores de palavras, como vetores de frequência de termos).

Vamos adicionar a transformação com *StringToWordVector* ao código para preparar os dados de texto para o classificador *NaiveBayes*.

Explicação das mudanças:

- **StringToWordVector**: O filtro StringToWordVector converte os atributos de texto em vetores de frequência de palavras, adequando o conjunto de dados para ser usado pelo NaiveBayes.
- **Filtragem dos dados**: A função Filter.useFilter() aplica o filtro ao conjunto de dados, retornando uma nova versão (filteredData) que será usada para treinar o classificador.
- **Carregamento dos dados:** O código carrega o arquivo .arff usando o DataSource.

- Definição do atributo de classe: O código define o atributo de classe (que é o último atributo do conjunto de dados).
- **Conversão de texto para vetores de palavras:** Utiliza o filtro StringToWordVector para transformar os textos em vetores de palavras, convertendo os atributos de texto (String) em uma forma numérica que o classificador pode processar.
- **Treinamento do NaiveBayes**: Após a filtragem, o modelo NaiveBayes é treinado com os dados convertidos.
- **Classificação de novo texto**: Um exemplo de texto é classificado, e o resultado da classificação é exibido.



Codigo: Java

```
package java_generative_ai;

import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.StringToWordVector;

public class SentimentAnalysis {

   public static void main(String[] args) throws Exception {
      // Carregar o conjunto de dados (arquivo .arff com textos rotulados)
      System.out.println("Carregando o conjunto de dados...");
      DataSource source = new
DataSource("C:\\Users\\HOME\\Downloads\\Java_GenerativeAI\\Java_Generative_AI\\src\\java_generative_ai\\se
ntiment_data.arff");
      Instances dataset = source.getDataSet();
      System.out.println("Conjunto de dados carregado com sucesso!");

      // Definir qual atributo será classificado (último no conjunto de dados)
      System.out.println("Definindo o atributo que será classificado...");
      dataset.setClassIndex(dataset.numAttributes() - 1);
      System.out.println("Atributo de classe definido: " + dataset.classAttribute().name());
```

```
        // Exibir o texto original antes da conversão
        System.out.println("Texto original: " + dataset.get(0));

        // Transformar atributos de texto em vetores de palavras
        System.out.println("Convertendo atributos de texto em vetores de palavras...");
        StringToWordVector filter = new StringToWordVector();
        filter.setInputFormat(dataset); // Configurar o filtro para o formato dos dados

        // Aplicar o filtro ao conjunto de dados
        Instances filteredData = Filter.useFilter(dataset, filter);
        System.out.println("Atributos de texto convertidos com sucesso!");

        // Criar e treinar o classificador NaiveBayes
        System.out.println("Treinando o modelo NaiveBayes...");
        NaiveBayes classifier = new NaiveBayes();
        classifier.buildClassifier(filteredData);
        System.out.println("Modelo treinado com sucesso!");

        // Exemplo de nova instância para classificar
        System.out.println("Classificando um novo exemplo de texto...");
        Instance newText = filteredData.get(0); // Simulando um novo texto para classificação
        double label = classifier.classifyInstance(newText);

        // Exibir resultado da classificação
        System.out.println("Classificação prevista: " + filteredData.classAttribute().value((int) label));
    }
}
```

Output: yaml

```
run:
Carregando o conjunto de dados...
Conjunto de dados carregado com sucesso!
Definindo o atributo que será classificado...
Atributo de classe definido: class
Texto original: 'Hoje o dia está maravilhoso',positivo
Convertendo atributos de texto em vetores de palavras...
Atributos de texto convertidos com sucesso!
Treinando o modelo NaiveBayes...
Modelo treinado com sucesso!
Classificando um novo exemplo de texto...
Classificação prevista: positivo
BUILD SUCCESSFUL (total time: 0 seconds)
```

Output - Java_Generative_AI (run)

```
run:
Carregando o conjunto de dados...
Conjunto de dados carregado com sucesso!
Definindo o atributo que será classificado...
Atributo de classe definido: class
Texto original: 'Hoje o dia está maravilhoso',positivo
Convertendo atributos de texto em vetores de palavras...
Atributos de texto convertidos com sucesso!
Treinando o modelo NaiveBayes...
Modelo treinado com sucesso!
Classificando um novo exemplo de texto...
Classificação prevista: positivo
BUILD SUCCESSFUL (total time: 0 seconds)
```