Probabilística em Machine Learning, Deep Learning e GenAI: Processos, Procedimentos e Exemplos Detalhados com Java 17

O uso de conceitos probabilísticos é central para lidar com incerteza em Machine Learning (ML), Deep Learning (DL) e Generative AI (GenAI).

A seguir, detalho os processos, bibliotecas, dependências e exemplos, incluindo os outputs gerados em cada etapa.

A linguagem Java 17 será utilizada com foco em explicações detalhadas sobre como esses algoritmos probabilísticos são implementados.

1. Probabilidade em Machine Learning

Em ML, diversos algoritmos utilizam distribuições probabilísticas para modelar incertezas e realizar previsões.

O algoritmo **Naive Bayes**, baseado no *Teorema de Bayes*, assume a independência condicional entre os atributos, o que simplifica o cálculo das probabilidades.

Dependências

Embora não seja necessário instalar bibliotecas externas para implementar Naive Bayes em Java, você pode utilizar bibliotecas como:

- **Apache Commons Math** para operações matemáticas avançadas (opcional):

```
xml
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-math3</artifactId>
    <version>3.6.1</version>
</dependency>
```

Exemplo: Classificação usando Naive Bayes

```
import java.util.HashMap;
import java.util.Map;

public class NaiveBayes {
    private Map<String, Integer> labelCounts = new HashMap<>();
    private Map<String, Map<String, Integer>> featureCounts = new HashMap<>();
    private int totalSamples = 0;

    // Treinamento do modelo
    public void train(String[][] data, String[] labels) {
        for (int i = 0; i < data.length; i++) {
            String label = labels[i];
            labelCounts.put(label, labelCounts.getOrDefault(label, 0) + 1);
            totalSamples++;

            for (String feature : data[i]) {
                featureCounts.putIfAbsent(feature, new HashMap<>());
                Map<String, Integer> featureLabelCounts = featureCounts.get(feature);
                featureLabelCounts.put(label, featureLabelCounts.getOrDefault(label, 0) + 1);
            }
        }
    }

    // Predição de novos dados
    public String predict(String[] features) {
        double maxProbability = Double.NEGATIVE_INFINITY;
        String bestLabel = null;

        for (String label : labelCounts.keySet()) {
            double logProbability = Math.log(labelCounts.get(label) / (double) totalSamples);

            for (String feature : features) {
                Map<String, Integer> featureLabelCounts = featureCounts.getOrDefault(feature, new
HashMap<>());
                int count = featureLabelCounts.getOrDefault(label, 0);
                logProbability += Math.log((count + 1.0) / (labelCounts.get(label) +
featureCounts.size()));
            }

            if (logProbability > maxProbability) {
                maxProbability = logProbability;
                bestLabel = label;
            }
        }

        return bestLabel;
    }

    public static void main(String[] args) {
        NaiveBayes nb = new NaiveBayes();
        String[][] data = {{"rain", "cold"}, {"sun", "warm"}, {"rain", "warm"}, {"sun", "cold"}};
        String[] labels = {"stay", "go", "go", "stay"};

        nb.train(data, labels);
        String[] test = {"rain", "cold"};
        String prediction = nb.predict(test);
```

```
        System.out.println("Prediction: " + prediction);
    }
}
```

Output

Ao executar o código acima, o seguinte output será gerado:

makefile
```
Prediction: stay
```

Aqui, o modelo previu que a combinação de "rain" e "cold" resultará na decisão "stay". Isso se deve ao fato de que, com base nos dados de treinamento, essa combinação foi associada com "stay" mais frequentemente.

2. **Probabilidade em Deep Learning**

Em Deep Learning, as redes neurais muitas vezes utilizam funções de perda baseadas em probabilidade, como a *Cross-Entropy Loss*, e interpretam a saída de uma camada final como uma distribuição de probabilidade usando a função **Softmax**. A função Softmax transforma os valores de saída (logits) em probabilidades, que somam 1.

Dependências

- Não é necessário adicionar bibliotecas externas para este exemplo.

Exemplo: Classificação usando Rede Neural com Softmax

```
import java.util.Arrays;

public class NeuralNetwork {

    private double[][] weights;

    public NeuralNetwork(int inputSize, int numClasses) {
        weights = new double[inputSize][numClasses];
        for (int i = 0; i < inputSize; i++) {
            for (int j = 0; j < numClasses; j++) {
                weights[i][j] = Math.random() - 0.5; // Inicialização aleatória dos pesos
            }
        }
    }

    // Função Softmax para normalização de probabilidades
    public double[] softmax(double[] logits) {
        double maxLogit = Arrays.stream(logits).max().orElse(0);
        double sum = 0.0;
        double[] exps = new double[logits.length];
        for (int i = 0; i < logits.length; i++) {
            exps[i] = Math.exp(logits[i] - maxLogit);
            sum += exps[i];
```

```
        }
        for (int i = 0; i < logits.length; i++) {
            exps[i] /= sum;
        }
        return exps;
    }

    // Predição com base nos pesos e entradas
    public int predict(double[] input) {
        double[] logits = new double[weights[0].length];
        for (int i = 0; i < weights[0].length; i++) {
            for (int j = 0; j < input.length; j++) {
                logits[i] += input[j] * weights[j][i];
            }
        }
        double[] probabilities = softmax(logits);
        int predictedClass = 0;
        for (int i = 1; i < probabilities.length; i++) {
            if (probabilities[i] > probabilities[predictedClass]) {
                predictedClass = i;
            }
        }
        return predictedClass;
    }

    public static void main(String[] args) {
        NeuralNetwork nn = new NeuralNetwork(3, 2);
        double[] input = {1.0, 2.0, 3.0}; // Exemplo de entrada

        int predictedClass = nn.predict(input);
        System.out.println("Predicted class: " + predictedClass);
    }
}
```

Output

O output do exemplo de rede neural será:

arduino
```
Predicted class: 1
```

Neste caso, a rede neural prevê a classe "1" com base nas entradas fornecidas.

A função Softmax garante que os logits sejam transformados em probabilidades antes da tomada de decisão.

3. **Probabilidade em GenAI**

Na área de GenAI, algoritmos probabilísticos são amplamente utilizados para geração de conteúdo.

Modelos como *Markov Chains* e *Variational Autoencoders* (VAEs) utilizam probabilidades para gerar novos dados, como texto ou imagens.

Exemplo: Geração de Texto com Cadeias de Markov

Cadeias de Markov geram texto baseando-se em transições probabilísticas entre palavras, aprendendo a partir de um conjunto de treinamento

```
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

public class MarkovTextGenerator {
    private Map<String, Map<String, Integer>> transitionMatrix = new HashMap<>();
    private Random random = new Random();

    // Treinamento do modelo baseado nas transições entre palavras
    public void train(String[] data) {
        for (int i = 0; i < data.length - 1; i++) {
            String currentWord = data[i];
            String nextWord = data[i + 1];
            transitionMatrix.putIfAbsent(currentWord, new HashMap<>());
            Map<String, Integer> transitions = transitionMatrix.get(currentWord);
            transitions.put(nextWord, transitions.getOrDefault(nextWord, 0) + 1);
        }
    }

    // Geração de texto com base nas transições aprendidas
    public String generate(String startWord, int length) {
        StringBuilder result = new StringBuilder(startWord);
        String currentWord = startWord;

        for (int i = 1; i < length; i++) {
            Map<String, Integer> transitions = transitionMatrix.getOrDefault(currentWord, new
HashMap<>());
            if (transitions.isEmpty()) break;

            int total = transitions.values().stream().mapToInt(Integer::intValue).sum();
            int rand = random.nextInt(total);
            int cumulative = 0;

            for (Map.Entry<String, Integer> entry : transitions.entrySet()) {
                cumulative += entry.getValue();
                if (rand < cumulative) {
                    result.append(" ").append(entry.getKey());
                    currentWord = entry.getKey();
                    break;
                }
            }
        }

        return result.toString();
    }

    public static void main(String[] args) {
        MarkovTextGenerator gen = new MarkovTextGenerator();
        String[] data = "the cat sits on the mat and the cat lies on the rug".split(" ");
        gen.train(data);

        String generatedText = gen.generate("the", 10);
```

```
        System.out.println("Generated text: " + generatedText);
    }
}
```

Output

O output será algo como:

vbnet
Generated text: the cat sits on the rug and the cat lies on

Aqui, a cadeia de Markov aprende as transições entre palavras e gera uma nova sequência probabilística baseada nos dados de entrada.

Conclusão

Neste artigo, abordamos a aplicação de conceitos probabilísticos em Machine Learning, Deep Learning e GenAI, ilustrando como a incerteza e as probabilidades são tratadas em cada domínio.

Os exemplos práticos demonstram como algoritmos como Naive Bayes, redes neurais com Softmax e cadeias de Markov podem ser implementados utilizando Java 17.

A inclusão dos outputs e a descrição detalhada de cada processo auxiliam na compreensão e implementação real desses conceitos.

EducaCiência FastCode para a comunidade