UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Gerador de Horóscopo com N-Gram

INTELIGÊNCIA ARTIFICIAL

Gabriella Nascimento dos Santos da Silva
João Vinícius de Almeida Argolo
José Victor Ribeiro de Jesus
Mariana Silva Costa
Tiago Rodrigues dos Santos

SÃO CRISTÓVÃO – SERGIPE
FEVEREIRO DE 2026

Propósito	URL
Repositório do Projeto	https://github.com/Kwttni/Gerador_de_Horoscopo_com_N-Gram
Link do app no Hugging Face Spaces	https://huggingface.co/spaces/tiagorodriguesdev/Horoscope-Generator-with-N-Grams

Problema Escolhido

O problema escolhido foi a **geração automática de previsões de horóscopo** a partir de um conjunto de textos existentes.

Horóscopos seguem padrões linguísticos específicos, com frases que frequentemente tratam de emoções, decisões pessoais, relacionamentos e situações do cotidiano. Dessa forma, o objetivo do projeto é desenvolver um sistema capaz de **aprender esses padrões a partir de um dataset de horóscopos reais** e gerar novos textos automaticamente, mantendo uma estrutura semelhante à dos exemplos utilizados no treinamento.

Como o Algoritmo de IA se Encaixa na Solução

Para resolver esse problema foi utilizado um **modelo de linguagem baseado em N-grams**.

Esse tipo de algoritmo aprende padrões estatísticos de sequência entre palavras presentes no dataset. Durante o treinamento, o modelo analisa os textos e registra com que frequência determinadas palavras aparecem após determinadas sequências.

No projeto foi utilizado um modelo **4-gram**, no qual a previsão da próxima palavra depende das três palavras anteriores.

Assim, quando o sistema gera um novo texto, ele escolhe a próxima palavra com base nas probabilidades aprendidas durante o treinamento.

Isso permite que o modelo produza frases com estrutura semelhante às descrições de horóscopo presentes no dataset, tornando a geração de texto mais coerente.

Como o Projeto Funciona

1. O dataset contendo horóscopos é carregado e tratado.
2. Um modelo N-Gram é treinado com as descrições.
3. O modelo aprende probabilidades de sequência entre palavras.

4. Ao selecionar um signo, o sistema gera um novo texto baseado nessas probabilidades.
5. O texto é traduzido para português.
6. Também são sorteados:
    - Humor do dia
    - Cor da sorte
    - Número da sorte

Correspondência entre Pseudocódigo e Implementação

Abaixo são apresentados trechos do pseudocódigo utilizado como referência e suas respectivas implementações no projeto.

Pré-processamento e Tokenização

```
function TRAIN-NGRAM-MODEL(corpus, n) returns a model
  model new NGRAM-MODEL structure
  tokens PREPROCESS-AND-TOKENIZE(corpus)
  // Adiciona tokens de padding no início se necessário para contexto inici
  tokens ADD-PADDING(tokens, n-1)

for i = 1 to LENGTH(tokens) - n + 1 do
    window SUBSEQUENCE(tokens, i, i + n - 1)
    history window[1 ... n-1]
    word window[n]
    INCREMENT-COUNT(model.counts, history, word)
    INCREMENT-COUNT(model.context_totals, history)

return model
```

```python
def train_model(self, corpus_list):
    """
    Treina o modelo utilizando uma lista de textos.

    Para cada texto:
    1. Tokeniza
    2. Adiciona padding
    3. Percorre janelas de tamanho n
    """
    for corpus in corpus_list:
        tokens = preprocess_and_tokenize(corpus)
        tokens = add_padding(tokens, self.n)

        for i in range(len(tokens) - self.n + 1):
            window = tokens[i : i + self.n]
            history = tuple(window[:-1])
            word = window[-1]

            self.counts[history][word] += 1
            self.context_totals[history] += 1
```

Criação dos N-grams

```
function GET-PROBABILITY(model, word, history) returns a probability
  // Assume que history tem tamanho n-1
  numerator GET-COUNT(model.counts, history, word)
  denominator GET-COUNT(model.context_totals, history)

if denominator = 0 then return 0 // Ou aplicar Suavização (Smoothing) a
  return numerator / denominator
```

```python
def get_probality(self, word, history):
    """
    Calcula a probabilidade condicional
    """
    history = tuple(history)

    numerator = self.counts[history][word]
    denominator = self.context_totals[history]

    if denominator == 0:
        return 0

    return numerator / denominator
```

Atualização das Contagens

```
function GENERATE-TEXT(model, seed_history, length) returns a string
  output seed_history
  current_history seed_history
  for i = 1 to length do
    candidates model.VOCABULARY()
    // Escolhe a próxima palavra baseada nas probabilidades calculadas
    next_word SAMPLE-FROM-DISTRIBUTION(model, candidates, current_history)
    APPEND(output, next_word)
    UPDATE(current_history, next_word) // Desliza a janela
  return output
```

```python
def generate_text(self, seed_history, length):
    """
    Gera um novo texto baseado no modelo treinado.

    seed_history:
    histórico inicial usado para começar a geração.
    Exemplo:
    ["<s>", "<s>", "<s>"]

    length:
    número máximo de tokens a gerar.
    """
    result = list(seed_history)
    current_history = tuple(seed_history)

    for _ in range(length):
        candidates = self.counts[current_history]

        if not candidates:
            break

        words = list(candidates.keys())
        weights = list(candidates.values())

        next_word = random.choices(words, weights=weights)[0]

        result.append(next_word)
        current_history = tuple(result[-(self.n - 1):])

        if next_word == "</s>":
            break

    return " ".join(result)
```