Lista 1 - Mineração de Texto (Versão Didática e Completa)

1. TF-IDF (Term Frequency – Inverse Document Frequency)

O TF mede quantas vezes um termo aparece em um documento, enquanto o IDF avalia a importância do termo dentro de todo o conjunto. Multiplicando ambos, obtemos o TF-IDF, que destaca termos relevantes para cada texto.

Exemplo de cálculo:

```
TF(t,d) = (número de vezes que t aparece em d) / (total de termos em d)
IDF(t) = log(N / Df(t))

Exemplo: termo “telhado” em d1 = (1/4) * log(3/1) = 0.25 * 0.477 = 0.119
```

O cálculo completo mostra que palavras como 'telhado', 'quintal', 'brincam' e 'juntos' têm maior peso (importância).

Código Python para cálculo de TF-IDF:

```
import math

corpus = [
    "o gato está no telhado",
    "o cachorro está no quintal",
    "o gato e o cachorro brincam juntos"
]

docs = [d.split() for d in corpus]
vocab = sorted(set(word for doc in docs for word in doc))
N = len(docs)

tf = [{t: doc.count(t)/len(doc) for t in vocab} for doc in docs]
df = {t: sum(t in doc for doc in docs) for t in vocab}
idf = {t: math.log(N/df[t]) for t in vocab}

tfidf = [{t: tf[i][t]*idf[t] for t in vocab} for i in range(N)]

for i, d in enumerate(tfidf):
    print(f"d{i+1}:", d)
```

2. Modelo Bag of Words (BoW)

O BoW transforma o texto em vetores de frequência. Cada termo representa uma posição no vetor, e o valor é a quantidade de vezes que aparece no documento.

Termo	d1	d2	d3
o	1	1	1
gato	1	0	1
está	1	1	0
na	1	0	0
casa	1	0	0
cachorro	0	1	1
no	0	1	0

quintal	0	1	0
e	0	0	1
brincam	0	0	1

3. Cosseno da Similaridade

A similaridade de cosseno compara a direção dos vetores no espaço vetorial. Mesmo documentos com tamanhos diferentes podem ser semelhantes se tiverem termos parecidos.

```
cos(θ) = (A · B) / (||A|| * ||B||)

Resultados:
d1 = 0.25
d2 = 0.79
d3 = 0.63

Ranking: d2 > d3 > d1
```

4. Algoritmo Python – BoW e Similaridade

```
import numpy as np

def bow_vector(text, vocab):
    return np.array([text.count(t) for t in vocab])

def cosine_similarity(a, b):
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

corpus = [
    "o gato está na casa",
    "o cachorro está no quintal",
    "o gato e o cachorro brincam"
]

vocab = sorted(set(" ".join(corpus).split()))
vectors = [bow_vector(doc, vocab) for doc in corpus]
query = "o cachorro brinca no quintal"
v_query = bow_vector(query.split(), vocab)

for i, v in enumerate(vectors):
    sim = cosine_similarity(v_query, v)
    print(f"d{i+1}: {sim:.2f}")
```

5. Tokenização, Stopwords, Stemming e Lemmatização

Ao remover palavras comuns e reduzir as palavras às suas raízes (stem) ou formas base (lemas), a análise se torna mais eficiente e generalizada.

```
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

corpus = [
    "o gato está na casa",
    "o cachorro está no quintal",
    "o gato e o cachorro brincam"
]
stop = set(stopwords.words('portuguese'))
```

```
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

def preprocess(text):
    tokens = nltk.word_tokenize(text)
    tokens = [t.lower() for t in tokens if t.isalpha()]
    tokens = [t for t in tokens if t not in stop]
    stems = [stemmer.stem(t) for t in tokens]
    lemmas = [lemmatizer.lemmatize(t) for t in tokens]
    return stems, lemmas

for doc in corpus:
    s, l = preprocess(doc)
    print("Original:", doc)
    print("Stems:", s)
    print("Lemmas:", l)
```

6. Estrutura do Perceptron

O Perceptron é o modelo mais simples de rede neural. Cada entrada x_i é multiplicada por um peso w_i, somada com um bias b e passada por uma função de ativação.

```
z = Σ(x_i * w_i) + b
f(z) = 1, se z ≥ 0; 0, caso contrário
```

7. Perceptron de Camada Única

a) Função degrau – saída binária (0 ou 1). b) Função sigmóide – saída contínua entre 0 e 1.

```
import numpy as np

def step(x): return 1 if x >= 0 else 0

def perceptron_step(X, y, lr=0.1, epochs=10):
    w = np.zeros(X.shape[1])
    b = 0
    for _ in range(epochs):
        for i in range(len(X)):
            z = np.dot(X[i], w) + b
            y_pred = step(z)
            w += lr * (y[i] - y_pred) * X[i]
            b += lr * (y[i] - y_pred)
    return w, b
```

8. Rede Neural de Múltiplas Camadas (MLP)

A MLP conecta várias camadas de neurônios, cada uma aplicando pesos e funções de ativação. A saída de uma camada serve como entrada para a próxima.

```
a(l) = f( Σ(x_i^(l-1) * w_i^(l)) + b^(l) )
```

9. Perceptron com Diferentes Funções de Ativação

```
import numpy as np

def sigmoid(x): return 1/(1+np.exp(-x))
def tanh(x): return np.tanh(x)
def relu(x): return np.maximum(0, x)
def relu_exp(x, alpha=0.1): return np.where(x>=0, x, alpha*(np.exp(x)-1))
```

10. Classificação de Sentimentos com Rede Neural

Treina-se uma MLP para distinguir sentimentos positivos e negativos com base em textos processados.

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

texts = [
 "O filme foi incrível, adorei cada cena.",
 "O produto chegou quebrado e me decepcionou.",
 "Excelente atendimento, fiquei muito satisfeito.",
 "A comida estava fria e sem sabor.",
 "Gostei bastante da qualidade e do design.",
 "O serviço é péssimo, nunca mais volto.",
 "Uma experiência maravilhosa, recomendo a todos.",
 "O aplicativo trava o tempo todo, é horrível.",
 "Achei o hotel confortável e bem localizado.",
 "O atendimento foi lento e desorganizado."
]

labels = ["Positivo","Negativo","Positivo","Negativo","Positivo",
          "Negativo","Positivo","Negativo","Positivo","Negativo"]

X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.3)
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

clf = MLPClassifier(hidden_layer_sizes=(5,), max_iter=1000)
clf.fit(X_train_vec, y_train)
y_pred = clf.predict(X_test_vec)
print("Acurácia:", accuracy_score(y_test, y_pred))
```

Conclusão

Esta lista consolidou os principais fundamentos de mineração de texto: TF-IDF, BoW, similaridade de cosseno e redes neurais. Com cálculos, exemplos e códigos Python, o aprendizado teórico se conecta à prática da análise textual moderna.