Procesamiento de información

Unidad 3

Marzo 2023

Autor
David Aarón Ramirez Olmeda

Introducción:

En la tarea a realizar se aplicarán técnicas de procesamiento de lenguaje natural para calcular modelos de probabilidad MLE y MLE con suavizado de Laplace para un modelo de bigramas utilizando el corpus "europarl.es". Además, se calcularán las probabilidades de ciertas oraciones utilizando ambos modelos y se realizarán predicciones de palabras dada una palabra inicial utilizando los modelos MLE y MLE con suavizado de Laplace.

Desarrollo

1

Calcular los modelos de probabilidad MLE y MLE con suavizado de Laplace para un modelo de bigramas. Con el corpus "europarl.es"

In [1]:
```
import string
from collections import Counter
import pickle
import pandas as pd
```

In [2]:
```python
# Lectura y preprocesamiento del corpus
with open("europarl.es", "r", encoding="utf8") as f:
    text = f.read()

# Convertir a minúsculas y borrar puntuación
text = text.lower()
text = text.translate(str.maketrans("", "", string.punctuation))

# Agregar marcadores de inicio y fin de oración
text = "<s> " + text.replace("\n", " </s>\n<s> ") + " </s>"

# Separar el texto en oraciones
sentences = text.split("\n")

# Calcular la frecuencia de cada palabra y de cada bigrama
unigram_counts = Counter()
bigram_counts = Counter()
for sentence in sentences:
    tokens = sentence.split()
    unigram_counts.update(tokens)
    bigram_counts.update(zip(tokens, tokens[1:]))

# Calcular la probabilidad MLE de cada bigrama
mle_probabilities = {}
for bigram, count in bigram_counts.items():
    previous_word = bigram[0]
    mle_probabilities[bigram] = count / unigram_counts[previous_word]

# Calcular la probabilidad MLE con suavizado de Laplace de cada bigrama
k = 1  # Constante de suavizado
laplace_probabilities = {}
for bigram, count in bigram_counts.items():
    previous_word = bigram[0]
    laplace_probabilities[bigram] = (count + k) / (unigram_counts[previous_

# Guardar las probabilidades en archivos
with open("mle_probabilities.pkl", "wb") as f:
    pickle.dump(mle_probabilities, f)

with open("laplace_probabilities.pkl", "wb") as f:
    pickle.dump(laplace_probabilities, f)
```

Este código procesa el corpus de texto, calcula las frecuencias de palabras y bigramas, y calcula las probabilidades MLE y las probabilidades MLE con suavizado de Laplace de cada bigrama. Estas probabilidades se guardan en archivos para su uso posterior

2

Calcular si la siguientes oraciones son posibles, es decir, calcular las probabilidades de las siguientes oraciones usando el modelo de MLE y MLE con suavizado de Laplace. Comparar las probabilidades.

In [3]:

```
# Cargar los diccionarios de probabilidades
with open("mle_probabilities.pkl", "rb") as f:
    mle_probabilities = pickle.load(f)

with open("laplace_probabilities.pkl", "rb") as f:
    laplace_probabilities = pickle.load(f)

# Función para calcular la probabilidad de una oración
def calculate_sentence_probability(sentence, probabilities):
    tokens = sentence.split()
    probability = 1.0
    for i in range(1, len(tokens)):
        bigram = (tokens[i-1], tokens[i])
        if bigram in probabilities:
            probability *= probabilities[bigram]
        else:
            probability = 0.0
            break
    return probability

# Calcular las probabilidades de las oraciones dadas
sentences = ["<s> el parlamento debe enviar un mensaje </s>",
             "<s> el parlamento debe enviar un consejo </s>",
             "<s> el abismo entre pobres y ricos </s>",
             "<s> el abismo entre ricos y pobres </s>",
             "<s> el abismo de la cantera entre pobres y ricos </s>",
             "<s> la comisión debe ser totalmente transparente </s>",
             "<s> la comisión debe ser transparente </s>"]



for sentence in sentences:
    mle_probability = calculate_sentence_probability(sentence, mle_probabil
    laplace_probability = calculate_sentence_probability(sentence, laplace_
    print("Oración:", sentence)
    print("Probabilidad MLE:", mle_probability)
    print("Probabilidad MLE con suavizado de Laplace:", laplace_probability
    print()
```

```
Oración: <s> el parlamento debe enviar un mensaje </s>
Probabilidad MLE: 4.4374173769257393e-13
Probabilidad MLE con suavizado de Laplace: 5.988734883346519e-21

Oración: <s> el parlamento debe enviar un consejo </s>
Probabilidad MLE: 3.357244293906632e-13
Probabilidad MLE con suavizado de Laplace: 9.358867331905685e-20

Oración: <s> el abismo entre pobres y ricos </s>
Probabilidad MLE: 3.807854577012913e-17
Probabilidad MLE con suavizado de Laplace: 1.673007647660524e-26

Oración: <s> el abismo entre ricos y pobres </s>
Probabilidad MLE: 8.648644418594847e-15
Probabilidad MLE con suavizado de Laplace: 1.1611469745072447e-24

Oración: <s> el abismo de la cantera entre pobres y ricos </s>
Probabilidad MLE: 0.0
Probabilidad MLE con suavizado de Laplace: 0.0

Oración: <s> la comisión debe ser totalmente transparente </s>
Probabilidad MLE: 3.597926526632743e-11
Probabilidad MLE con suavizado de Laplace: 7.496239620326898e-19

Oración: <s> la comisión debe ser transparente </s>
Probabilidad MLE: 2.5521292162248256e-09
Probabilidad MLE con suavizado de Laplace: 4.125572511955136e-15
```

Las probabilidades tan bajas que se muestran en la salida del código son comunes en el procesamiento del lenguaje natural y en particular en el modelado de lenguaje. Esto se debe a que la probabilidad de una oración se calcula multiplicando las probabilidades de sus bigramas, que son secuencias de dos palabras consecutivas. Como hay muchas combinaciones posibles de bigramas en una oración, la probabilidad total puede ser extremadamente baja.

En el caso específico de este código, las probabilidades bajas también se deben a que se están usando modelos de lenguaje simples, basados únicamente en bigramas y sin tener en cuenta otras características lingüísticas más complejas.

3

Predicción de palabras, dada una palabra inicial mostrar las siguientes cinco palabras más probables de acuerdo con los modelos MLE y MLE con suavizado de Laplace.

In [4]:

```
words = ['los', 'tribunales', 'nacionales']
result = []
for word in words:
    res1 = {k: v for k, v in mle_probabilities.items() if k[0] == word and
    res1 = sorted(res1.items(), key=lambda x: (-x[1], x[0]))[0:5]
    res2 = {k: v for k, v in laplace_probabilities.items() if k[0] == word
    res2 = sorted(res2.items(), key=lambda x: (-x[1], x[0]))[0:5]
    res1 = [('MLE', bigram[0], bigram[1]) for bigram in res1]
    res2 = [('Laplace', bigram[0], bigram[1]) for bigram in res2]
    result.extend(res1 + res2)

df = pd.DataFrame(result, columns=['Modelo', 'Palabra', 'Prob'])
df['Prob'] = df['Prob'].apply(lambda x: f"{x*100:.2f}%")
df
```

Out[4]:

	Modelo	Palabra	Prob
0	MLE	(los, estados)	6.97%
1	MLE	(los, países)	4.59%
2	MLE	(los, derechos)	3.64%
3	MLE	(los, que)	3.09%
4	MLE	(los, ciudadanos)	2.59%
5	Laplace	(los, estados)	3.61%
6	Laplace	(los, países)	2.38%
7	Laplace	(los, derechos)	1.89%
8	Laplace	(los, que)	1.60%
9	Laplace	(los, ciudadanos)	1.34%
10	MLE	(tribunales, nacionales)	12.12%
11	MLE	(tribunales, de)	11.36%
12	MLE	(tribunales, y)	5.30%
13	MLE	(tribunales, en)	4.55%
14	MLE	(tribunales, del)	3.79%
15	Laplace	(tribunales, nacionales)	0.04%
16	Laplace	(tribunales, de)	0.03%
17	Laplace	(tribunales, y)	0.02%
18	Laplace	(tribunales, en)	0.01%
19	Laplace	(tribunales, del)	0.01%
20	MLE	(nacionales, de)	13.61%
21	MLE	(nacionales, y)	13.52%
22	MLE	(nacionales, en)	5.01%
23	MLE	(nacionales, que)	4.21%
24	MLE	(nacionales, a)	1.88%
25	Laplace	(nacionales, de)	0.31%
26	Laplace	(nacionales, y)	0.31%
27	Laplace	(nacionales, en)	0.12%
28	Laplace	(nacionales, que)	0.10%
29	Laplace	(nacionales, a)	0.04%

Este código es una forma de encontrar las cinco bigramas más probables que comienzan con cada palabra de la lista dada utilizando los dos modelos

3.1

Probar con el inicio de la palabra "la", probar con el inicio de la palabra "parlamento"

In [5]:
```
words = ['la', 'parlamento']
result = []
for word in words:
    res1 = {k: v for k, v in mle_probabilities.items() if k[0] == word and
    res1 = sorted(res1.items(), key=lambda x: (-x[1], x[0]))[0:5]
    res2 = {k: v for k, v in laplace_probabilities.items() if k[0] == word
    res2 = sorted(res2.items(), key=lambda x: (-x[1], x[0]))[0:5]
    res1 = [('MLE', bigram[0], bigram[1]) for bigram in res1]
    res2 = [('Laplace', bigram[0], bigram[1]) for bigram in res2]
    result.extend(res1 + res2)

df1 = pd.DataFrame(result, columns=['Modelo', 'Palabra', 'Prob'])
df1['Prob'] = df1['Prob'].apply(lambda x: f"{x*100:.2f}%")
df1
```

Out[5]:

	Modelo	Palabra	Prob
0	MLE	(la, comisión)	9.70%
1	MLE	(la, unión)	5.29%
2	MLE	(la, política)	1.63%
3	MLE	(la, sra)	1.40%
4	MLE	(la, ue)	1.35%
5	Laplace	(la, comisión)	6.89%
6	Laplace	(la, unión)	3.76%
7	Laplace	(la, política)	1.16%
8	Laplace	(la, sra)	1.00%
9	Laplace	(la, ue)	0.96%
10	MLE	(parlamento, europeo)	29.52%
11	MLE	(parlamento, y)	6.50%
12	MLE	(parlamento, en)	3.92%
13	MLE	(parlamento, aprueba)	2.62%
14	MLE	(parlamento, que)	2.62%
15	Laplace	(parlamento, europeo)	3.48%
16	Laplace	(parlamento, y)	0.77%
17	Laplace	(parlamento, en)	0.46%
18	Laplace	(parlamento, aprueba)	0.31%
19	Laplace	(parlamento, que)	0.31%

Incluir 3 ejemplos para demostrar sus modelos de predicción "interactiva".

1. Modelo de predicción MLE:

In [6]:
```
res1 = {k: v for k, v in mle_probabilities.items() if k[0] == 'español' and
sorted(res1.items(), key=lambda x: (-x[1], x[0]))[0:5]
```

Out[6]:
```
[(('español', 'que'), 0.12871287128712872),
 (('español', 'y'), 0.0891089108910891),
 (('español', 'en'), 0.0594059405940594),
 (('español', 'sr'), 0.039603960396039604),
 (('español', 'ha'), 0.0297029702970297)]
```

2. Modelo de predicción con suavizado de Laplace:

In [7]:
```
res1 = {k: v for k, v in mle_probabilities.items() if k[0] == 'ojo' and '</
sorted(res1.items(), key=lambda x: (-x[1], x[0]))[0:5]
```

Out[7]:
```
[(('ojo', 'de'), 0.3333333333333333),
 (('ojo', 'la'), 0.16666666666666666),
 (('ojo', 'no'), 0.16666666666666666),
 (('ojo', 'para'), 0.16666666666666666),
 (('ojo', 'y'), 0.16666666666666666)]
```

3. Modelo de predicción de la próxima palabra en una oración:

In [8]:
```
text = 'los tribunales nacionales'
words = text.split()
previous_word = words[-1]

next_words = []
for bigram, probability in mle_probabilities.items():
    if bigram[0] == previous_word and bigram[1] != "</s>":
        next_words.append((bigram[1], probability))

next_words = sorted(next_words, key=lambda x: x[1], reverse=True)

# Mostrar las tres siguientes palabras más probables
print("Las tres siguientes palabras más probables son:")
for i in range(3):
    if i < len(next_words):
        print(f"{i+1}. {next_words[i][0]} {next_words[i][1]}")
    else:
        break
```

```
Las tres siguientes palabras más probables son:
1. de 0.1360787824529991
2. y 0.135183527305282
3. en 0.050134288272157566
```

Conclusiones

Se calculó la probabilidad de ciertas oraciones utilizando ambos modelos y se compararon las probabilidades. En general, esta tarea permite comprender cómo funcionan los modelos de lenguaje y cómo se pueden utilizar para realizar predicciones de palabras y calcular la probabilidad de ciertas oraciones.

In [ ]: