

1-6. Hafta Ders Notları

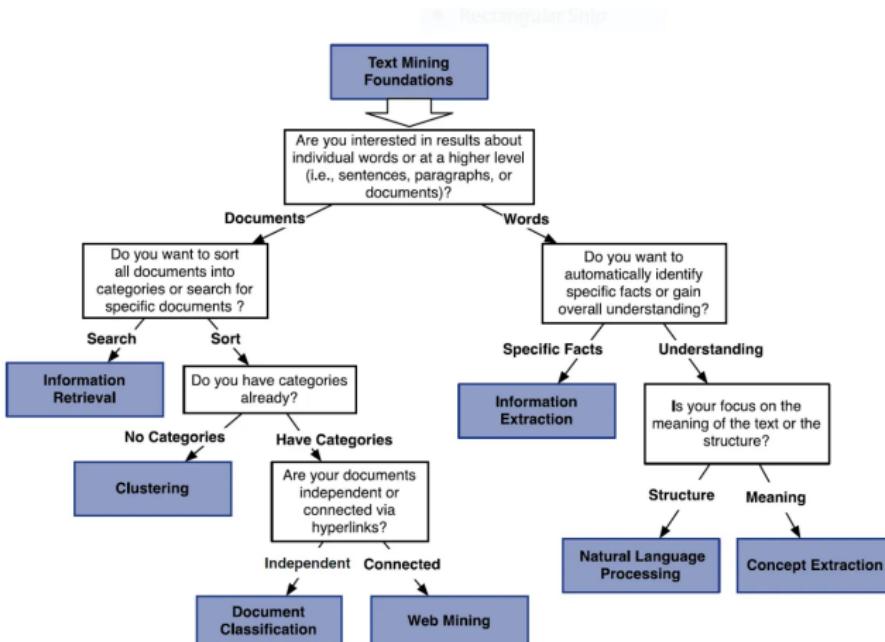
Dr.Furkan Göz

Kasım, 2023

- Seçenek 1: Özgün bir proje
- Seçenek 2: Var olan projeyi çalışma, anlama, anlatma (10-15 dk)

Tür	Açıklama
Dil	Python
Konu	NLP alanında özel bir konu
Kaynak	github
Grup sayısı	en fazla 2
Proje sayısı	her üye için 1 proje
Özel şartlar	en az 10 yıldız, 2023 yılında oluşturulmuş
Tarih	her hafta

Örnek konu arama: Topic modeling stars:>10 created:>2023-01-01



¹ Miner, G. (2012). Practical text mining and statistical analysis for non-structured text data applications

- Metin madenciliği, yapılandırılmış ve yapılandırılmamış metinlerden bilgiler elde etmeyi amaçlayan bir disiplindir.
- Hedef, metin verilerini sayısal formata dönüştürmek ve ardından bu verileri analiz etmektir.
- Metinlerin anlamını derinlemesine anlama amacı gütmemektedir.
- Metin sınıflandırma, metin kümeleme, bilgi çıkarma, içerik önerme

- Doğal Dil İşleme (NLP), insan dilinin anlaşılması, yorumlanması ve üretilmesini amaçlayan bir alan
- metin, konuşma, işaret, görüntü...
- Amaç bilgisayar sistemlerinin insan dilini veya metinleri anlayabilmesini sağlamak
- Sözdizim analizi ve anlamsal analizi
- Veri kaynağı tablolar, konuşmalar ve daha birçok farklı kaynaktan gelebilir. Dilin gramatik yapısını anlamak gibi problemler vardır.
- chatbot, makine çevirisi, metin özetleme

NLP-hard: Belirsizlikler

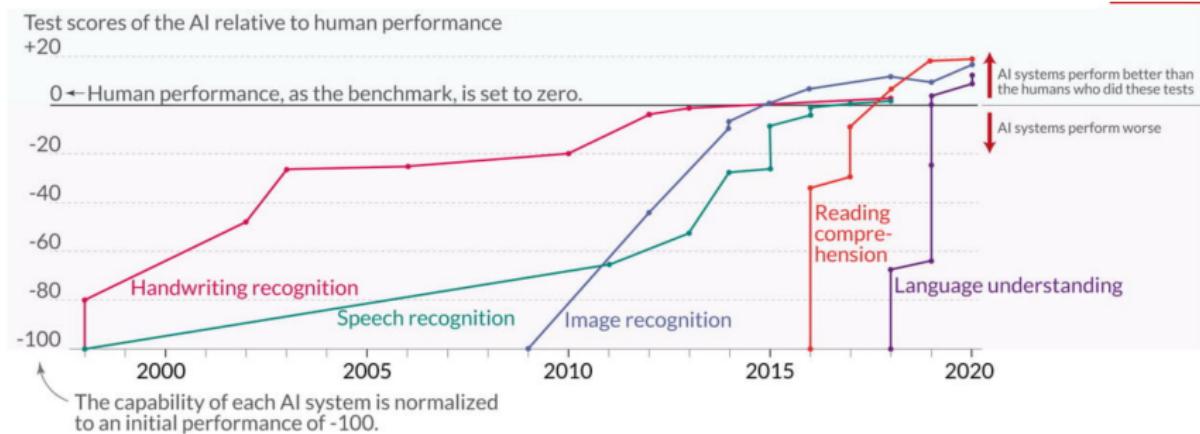
- Kelime anımları: fare
- Sözcük türleri: gül
- Nicelik: Her öğrenci bazı kitapları okur.
- Referans: Ali bir tane elma aldı. O tatlıydı.
- Söylem: Toplantı iptal edildi. Ali bugün ofise gelmeyecek.

NLP-hard: Zip's Law

- Önemsiz kelimelerin daha çok bulunması
- "the," "and," "in" gibi çok kullanılan kelimeler
- George Kingsley Zipf, James Joyce'un 2 Şubat 1922'de yayınlanan Ulysses romanını incelemiştir: en fazla bulunan sözcük 8000, en fazla bulunan 10. sözcük 800 defa kullanılmış

NLP-hard: Diğerleri

- Yazım yanlışları: internet dili (sözcük türleri?)
- Aynı anlama gelen cümleler: Ben geldim vs Geldim ben
- İfade: Pencere hala açık mı?



Data source: Kiela et al. (2021) – Dynabench: Rethinking Benchmarking in NLP
OurWorldInData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the author Max Roser

- Turing Testi, bir insanın bilgisayar ve insan arasında metin tabanlı iletişim yoluyla ayırt edip edemediğini ölçen bir testtir.
- Her iki tarafın kim olduğu değerlendirmeici tarafından bilinmez.
- Değerlendirici, bilgisayar programını insan ile ayırt edemiyorsa, program Turing Testini geçmiş sayılır ve insan benzeri zekaya sahip olduğu kabul edilir.
- Bu test, yapay zekanın insan benzeri düşünme ve davranış yeteneklerini ölçmek için kullanılır.

Regular Expressions - Regex

- Veri Ayıklama ve Düzenleme: Regex, metin verilerinden istenilen bilgileri çıkarmak veya metinleri düzenleme
- Desen Tanımlama: Regex metinlerdeki belirli desenleri tanımlama e-posta, telefon, URL
- Veri Doğrulama: Metin girişlerini doğrulama
- Metin Madenciliği: Regex, metin madenciliği projelerinde metinleri analiz etme
- Dil Modeli Eğitimi: Metinleri temizleme

Bu dört örnektten herhangi birini nasıl arayabiliriz?

- elephant
- elephants
- Elephant
- Elephants

[]

Pattern	Matches
[eE]lephant	Elephant, elephant
[1234567890]	Any digit

[A-Z]

[A-Z]	Bir büyük harf	Kocaeli University
[a-z]	Bir küçük harf	Kocaeli University
[0-9]	Bir rakam	Number 1

Değil

[^A-Z]	Büyük harf değil	Kocaeli University
[^Ss]	S de değil s de değil	Kocaeli University
[a^]	a ya da ^	Number a
a^b	Tam ifade	Number a^b
[^~]	^ olmasın	Kocaeli University

Regex



|

Pattern	Matches
begin start	start
a b c	a
[bB]egin [sS]tart	start

*?.+

colou?r	önceki karakter opsiyonel	color, colour
ab*c!	0 veya daha fazla	ac!, abc!, abbc!
ab+!	1 veya daha fazla	abc!, abbc!
a.c!	herhangi	abc!, adc!

$\wedge \ $$

Pattern	Matches
$\wedge [A-Z]$	Kocaeli
$\wedge [\wedge A-Za-z]$	1
$\backslash . \$$	Kocaeli.
$. \$$	Kocaeli!

Regex İfade	Eşdeğer	Açıklama
\d	[0-9]	bir rakam
\D	[^0-9]	rakam olmayan karakter
\w	[a-zA-Z0-9_]	alfanumerik karakter
\W	[^\w]	alfanumerik olmayan
\s	[\r\t\n\f]	boşluk karakteri (boşluk, sekme)
\S	[^\s]	boşluk olmayan

Desen	Açıklama
*	sıfır veya daha fazla kez
+	bir veya daha fazla kez
?	sıfır veya bir kez
{n}	n kez eşleşme
{n,m}	n ile m arasında eşleşme
{n,}	en az n kez eşleşme
{,m}	en fazla m kez eşleşme

Örnek: the

1. /the/ büyük harf yakalayamaz
2. /[tT]he/ kelimenin geçtiği durumları yakalayamaz
3. /\b[tT]he\b/ alt çizgi veya sayı gibi diğer karakterleri dikkate almaz
4. [^a-zA-Z] [tT]he[^a-zA-Z] / kelimenin bir satırın başında olduğu durumları yakalayamaz
5. /(^|[^a-zA-Z]) [tT]he([a-zA-Z]|\\$)/

```
1     import re
2     re.search(...)
3     re.match(...)
4     re.findIter(...)
5     re.findall(...)
6     re.sub(...)
7     re.split(...)
```

re.search(pattern, string)

```
import re

text = "Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümü B Blok 3. kattadır
pattern = r"Bilgi"

result = re.search(pattern, text)
if result:
    print("Eşleşme bulundu:", result.group())
else:
    print("Eşleşme bulunamadı.")
```

Eşleşme bulundu: Bilgi

```
import re

text = "Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümü B Blok 3. kattadır
pattern = r"\bBilgi\b"

result = re.search(pattern, text)
if result:
    print("Eşleşme bulundu:", result.group())
else:
    print("Eşleşme bulunamadı.")
```

Eşleşme bulunamadı.

re.match(pattern, string)

```
import re

text = "Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümü B Blok 3. kattadır."
pattern = r"Bilgisayar"

result = re.match(pattern, text)
if result:
    print("Eşleşme bulundu:", result.group())
else:
    print("Eşleşme bulunamadı.")
```

Eşleşme bulunamadı.

```
import re

text = "Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümü B Blok 3. kattadır."
pattern = r"\bKocaeli\b"

result = re.match(pattern, text)
if result:
    print("Eşleşme bulundu:", result.group())
else:
    print("Eşleşme bulunamadı.")
```

Eşleşme bulundu: Kocaeli

re.findall(pattern, string)

```
import re

text = "Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümü B Blok 3. kattadır."
pattern = r"B"

result = re.findall(pattern, text)
if result:
    print("Eşleşme bulundu:", result)
else:
    print("Eşleşme bulunamadı.")

Eşleşme bulundu: ['B', 'B', 'B']
```

```
import re

text = "Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümü B Blok 3. kattadır."
pattern = r"\bB\b"

result = re.findall(pattern, text)
if result:
    print("Eşleşme bulundu:", result)
else:
    print("Eşleşme bulunamadı.")

Eşleşme bulundu: ['B']
```

re.sub(pattern, repl, string)

```
import re

text = "Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümü B Blok 3. kattadır"
pattern = r"M"
replacement = "m"

result = re.sub(pattern, replacement, text)
print("Sonuç:", result)
```

Sonuç: Kocaeli Üniversitesi Bilgisayar mühendisliği bölümü B Blok 3. kattadır.

```
import re

text = "Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümü B Blok 3. kattadır"
pattern = r"\bMühendisliği\b"
replacement = "M."

result = re.sub(pattern, replacement, text)
print("Sonuç:", result)
```

Sonuç: Kocaeli Üniversitesi Bilgisayar M. bölümü B Blok 3. kattadır.

Regex Örnek



```
import re

metin = "The ... The_ lazy cat. There are ... in the garden."

duzenli_ifade1 = r'\b[tT]he\b'
sonuc1 = re.findall(duzenli_ifade1, metin)
e1 = [son.group() for son in sonuc1]

duzenli_ifade2 = r'[^a-zA-Z](t|T)he[^a-zA-Z]'
sonuc2 = re.findall(duzenli_ifade2, metin)
e2 = [son.group() for son in sonuc2]

duzenli_ifade3 = r'(^|[^a-zA-Z])[tT]he([a-zA-Z]|$)'
sonuc3 = re.findall(duzenli_ifade3, metin)
e3 = [son.group() for son in sonuc3]

print("İlk Düzenli İfade Sonuçları:", e1)
print("İkinci Düzenli İfade Sonuçları:", e2)
print("Üçüncü Düzenli İfade Sonuçları:", e3)
```

İlk Düzenli İfade Sonuçları: ['The', 'the']

İkinci Düzenli İfade Sonuçları: [' The_', ' the ']

Üçüncü Düzenli İfade Sonuçları: ['The ', ' The_', ' the ']

Regex Örnek



```
email_regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

email_addresses = [
    'furkan.goz@kocaeli.edu.tr',
    'furkangoz@gmail',
    'furkangoz',
    'user@gmail.com',
    'furkan.goz@kocaeli.edu.tr1'#$
]

for email in email_addresses:
    if re.match(email_regex, email):
        print(f"'{email}' geçerli bir e-posta adresi.")
    else:
        print(f"'{email}' geçerli bir e-posta adresi değil.")
```

'furkan.goz@kocaeli.edu.tr' geçerli bir e-posta adresi.
'furkangoz@gmail' geçerli bir e-posta adresi değil.
'furkangoz' geçerli bir e-posta adresi değil.
'user@gmail.com' geçerli bir e-posta adresi.
'furkan.goz@kocaeli.edu.tr1' geçerli bir e-posta adresi değil.

Regex Örnek

```
import re

text = """Date: 09/10/2023, Phone Number: 5361234567,
Website: http://www.example.com or https://www.example.com/,
IP Address: 192.168.1.1"""

date_pattern = r'\d{2}[\.]\d{2}[\.]\d{2,4}'
phone_pattern = r'\d{10}'
website_pattern = r'https?://\S+'
ip_pattern = r'\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}'

dates = re.findall(date_pattern, text)
phone_numbers = re.findall(phone_pattern, text)
websites = re.findall(website_pattern, text)
ip_addresses = re.findall(ip_pattern, text)

for date in dates:
    print("Tarih:", date)

for phone_number in phone_numbers:
    print("Tel:", phone_number)

for website in websites:
    print("Website:", website)

for ip_address in ip_addresses:
    print("IP Address:", ip_address)
```

Tarih: 09/10/2023
Tel: 5361234567
Website: <http://www.example.com>
Website: <https://www.example.com/>
IP Address: 192.168.1.1

re.split(pattern,string)

```
import re

text = "KOU Bilgisayar      Müh. bölümü B Blok 3. kattadır."
pattern = r"\s+"

result = re.split(pattern, text)
print(result)
```

```
['KOU', 'Bilgisayar', 'Müh.', 'bölümü', 'B', 'Blok', '3.', 'kattadır.']}
```

```
import re

text = "KOU Bilgisayar      Müh. bölümü B Blok 3. kattadır."
pattern = r"\s"

result = re.split(pattern, text)
print(result)
```

```
['KOU', 'Bilgisayar', '', '', '', 'Müh.', 'bölümü', 'B', 'Blok', '3.', 'kattadır.']}
```

Bazları

- Metnin alt birimlere dönüştürülmesi
- Büyük küçük harf dönüşümü (Lowercase Conversion)
- Olumsuzlama işleme (Negation Handle)
- Alfabetik olmayan karakterlerin çıkarılması
- Etkisiz kelimelerin çıkarılması
- Kelime köklerinin bulunması
- Kısaltmaların normal hale getirilmesi
- Yazım hatalarının düzeltilmesi
-

Gürültü Azaltma

- Problem: özel karakterler, yazım hataları veya alakasız semboller
- Metni daha okunabilir ve analiz edilebilir hale getirir

Gürültü Azaltma

Örnek

Harikaydık, futbol budur!!!rosesmile #futbolgecesi

Hey, What's up!!!???? Nice 2 meet u :)

Dönüşüm

Harikaydık, futbol budur!

Hey, What's up? Nice to meet you.

- İşlemler: Noktalama, emoji, numara

Standartlaştırma

- Problem: tarih biçimleri, sayısal gösterimler ve birimlerde farklı kurallar olabilir
- tutarlılık

Standartlaştırma

Tarih

Ekim 16, 2023

16-10-2023

10/16/23

16.10.2023

Mesafe

10 mil

16 km

Sayısal Değerler

3,14

3.14

Standartlaştırma

Benzer Kelimeler

colour - color

car - automobile

Farklı Karakter

résumé - resume

Eş Anlamlı Kelimeler

kalp -yatak

Temel Ön İşlemler

- Tokenization
- Sentence Splitting
- Pos Tagging: Kelime türlerinin bulunması
- Stopwords
- Stemmazation
- Lemmatization

- Metinleri makinenin anlayabileceği birimlere dönüşümü
- Metnin en küçük birimlere çevrimi (kelime, karakter, kelime öbeği)
- zordur

Tire

knowledge-based, self-confidence, state-of-the-arts

Özel İsimler

O'Reilly, İstanbul'a
New York

Phrasal Verbs

give up, turn on

Diğer Dil Problemleri

Almanca, Arapça...

Boşluk kullanılmayan diller

Çince

Özel Karakter ve Semboller

C++, \$10

New York

Emojiler

lol, :)

Noktalama

gidiyorum.

dut, karpuz

```
1 import nltk #import re
2 import spacy
3 nltk.download('punkt')
4 from nltk.tokenize import word_tokenize
5 from textblob import TextBlob
6 from gensim.utils import simple_tokenize
7 from keras.preprocessing.text import text_to_word_sequence
8 text = "That U.S.A. don't poster-print costs $12.40..."
```

Örnek



```
1 tokens_nltk = word_tokenize(text)
2 print("(1)", tokens_nltk)
3
4 tokenizer = nltk.tokenize.TreebankWordTokenizer()
5 tokens = tokenizer.tokenize(text)
6 print("(2)",tokens)
7
8 tokens_split = text.split()
9 print("(3)",tokens_split)
10
11 blob = TextBlob(text)
12 tokens_textblob = blob.words
13 print("(4)",tokens_textblob)
```

```
1 tokens_gensim = list(simple_tokenize(text))
2 print("(5)", tokens_gensim)
3
4 tokens_keras = text_to_word_sequence(text)
5 print("(6)",tokens_keras)
6
7 nlp = spacy.load("en_core_web_sm")
8 doc = nlp(text)
9 tokens = [token.text for token in doc]
10 print("(7)",tokens)
11
12 pattern = r'[A-Z]\.\s+|\$\?\d+\.\d+%\?|\w+[-\']?\w+'
13
14 tokens = nltk.regexp_tokenize(text, pattern)      #tokens = re.
15    .findall(pattern, text)
15 print("(8)",tokens)
```

Örnek

- 1 (1) ['That', 'U.S.A.', 'do', "n't", 'poster-print', 'costs', '\$', '12.40', '...']
- 2 (2) ['That', 'U.S.A.', 'do', "n't", 'poster-print', 'costs', '\$', '12.40', '...']
- 3 (3) ['That', 'U.S.A.', "don't", 'poster-print', 'costs', '\$12.40...']
- 4 (4) ['That', 'U.S.A', 'do', "n't", 'poster-print', 'costs', '12.40']
- 5 (5) ['That', 'U', 'S', 'A', 'don', 't', 'poster', 'print', 'costs']
- 6 (6) ['that', 'u', 's', 'a', "don't", 'poster', 'print', 'costs', '12', '40']
- 7 (7) ['That', 'U.S.A.', 'do', "n't", 'poster', '-', 'print', 'costs', '\$', '12.40', '...']
- 8 (8) ['That', 'U.S.A.', "don't", 'poster-print', 'costs', '\$12.40']

```
1 (tokenization_nltk_word_tokenize) Execution Time: 1.030519
   seconds
2 (tokenization_nltk_treebank_word_tokenizer) Execution Time:
   0.439971 seconds
3 (tokenization_textblob) Execution Time: 1.834314 seconds
4 (tokenization_gensim_simple_tokenize) Execution Time: 0.061550
   seconds
5 (tokenization_keras_text_to_word_sequence) Execution Time:
   0.048674 seconds
6 (tokenization_nltk_regex_tokenize) Execution Time: 0.032881
   seconds
```

- analiz ve işleme için daha yönetilebilir hale getirmek
- bağlam ve anlam için tek tek cümleleri analiz etmek önemli olabilir (duygu analizi, makine çevirisi)
- bilgi edinme (kelimelerin indisini önemli olabilir)
- Dilbilgisi ve Sözdizimi Analizi
- Metin özetleme gibi amaçlar için kullanılabilir

```
1 import nltk
2 import spacy
3 from textblob import TextBlob
4
5 text = "This is the first sentence. This is the second sentence
       ! And this is the third sentence? Abbreviations like etc.
       or Dr. are common."
```

```
1 nltk_sentences = nltk.sent_tokenize(text)
2 print("NLTK Sentences:")
3 for sentence in nltk_sentences:
4     print(sentence)
5 nlp = spacy.load("en_core_web_sm")
6 doc = nlp(text)
7 print("\nspaCy Sentences:")
8 for sentence in doc.sents:
9     print(sentence.text)
10 blob = TextBlob(text)
11 print("\nTextBlob Sentences:")
12 for sentence in blob.sentences:
13     print(sentence)
14 sentence_pattern = r'(?!\\w\\.\\w.)(?=<\\.|\\?|\\!|\\s+(?=\\w))'
15 sentences = re.split(sentence_pattern, text)
16 print("\nRegex Sentences:")
17 for sentence in sentences:
18     print(sentence.strip())
```

1 NLTK Sentences:

2 This is the first sentence.

3 This is the second sentence!

4 And this is the third sentence?

5 Abbreviations like etc.

6 or Dr. are common.

7

8 spaCy Sentences:

9 This is the first sentence.

10 This is the second sentence!

11 And this is the third sentence?

12 Abbreviations like etc. or Dr. are common.

```
1
2 TextBlob Sentences:
3 This is the first sentence.
4 This is the second sentence!
5 And this is the third sentence?
6 Abbreviations like etc.
7 or Dr. are common.
8
9 Regex Sentences:
10 This is the first sentence.
11 This is the second sentence!
12 And this is the third sentence?
13 Abbreviations like etc. or Dr. are common.
```

- Zip's Law.
- Hazır listeler.
 - a, an, the
 - bile, hiç, pek
- problemler?
 - To be or not to be
 - flight to Isparta

```
1 import nltk
2 from nltk.corpus import stopwords
3 from nltk.tokenize import word_tokenize
4
5 nltk.download('stopwords')
6 english_stopwords = set(stopwords.words('english'))
7
8 def remove_english_stopwords(text):
9     words = word_tokenize(text)
10    filtered_words = [word for word in words if word.lower()
11                      not in english_stopwords]
12    return ' '.join(filtered_words)
13
14 english_text = "The sentence is an example of some English stop
15 words."
16 english_text_without_stopwords = remove_english_stopwords(
17     english_text)
18 print(english_text_without_stopwords)
```

```
1 import spacy
2 nlp_en = spacy.load('en_core_web_sm')
3
4 def remove_stopwords(text, language):
5     if language == 'english':
6         doc = nlp_en(text)
7         filtered_words = [token.text for token in doc if not token.
8                         is_stop]
9         return ' '.join(filtered_words)
10
11 english_text = "The sentence is an example of some English stop
12 words."
13 english_text_without_stopwords = remove_stopwords(english_text,
14 'english')
15 print(english_text_without_stopwords)
```

Çıktı

sentence example English stop words .

```
1 import nltk
2 from stop_words import get_stop_words
3 from nltk.tokenize import word_tokenize
4 turkish_stopwords = get_stop_words('turkish')
5
6 def remove_turkish_stopwords(text):
7     words = word_tokenize(text)
8     filtered_words = [word for word in words if word.lower()
9                       not in turkish_stopwords]
10    return ' '.join(filtered_words)
11 turkish_text = "Bu, ıbaz etkisiz kelimeleri içeren bir cümledir
12 ."
13 turkish_text_without_stopwords = remove_turkish_stopwords(
14     turkish_text)
15 print(turkish_text_without_stopwords)
```

Çıktı

, bazı Türkçe etkisiz kelimeleri içeren cümledir .

- Bir sözcüğün kökünü veya temel biçimini bulma işlemidir.
- Kelimeleri basitleştirmek için son ekleri, ön ekleri veya kelime sonları kaldırılır.
- Belirli kurallara ve algoritmala dayanır.
- Birden fazla anlamı olan kelimeler nedeniyle yanlışlıklara yol açabilir.

- temel veya sözlük biçimlerine indirgemek için kullanılan bir doğal dil işleme (NLP) tekniği
- sözcük türlerini kullanır
 - "running" "run"
 - "better" "good"
 - "are" "be"

- Kural-tabanlı: dilbilgisi kurallarına göre çalışır. özel durumları bulması zordur.
- Sözlük-tabanlı: önceden belirlenmiş sözlükten eşleme yapar. sözcük dağarcığınar bağlı.
- Makine öğrenmesi-tabanlı: eğitim gerektirir. kaynak sayısı az diller için problem.
- Dile özel yaklaşımalar

- Stemming genellikle lemmatizasyondan daha hızlıdır, daha az maliyetlidir
- Stemming genellikle kelimenin cümle içindeki bağlamını dikkate almaz. Ön ekleri veya son ekleri keser, bu da anlam kaybına neden olabilir
- Lemmatizasyon kelimenin anlamını ve bağlamını dikkate alır. Daha etkilidir

- Porter Stemmer
- Snowball Stemmer
- Lancaster Stemmer

Stemming Örnek



```
1 import nltk
2 from nltk.stem import PorterStemmer, SnowballStemmer,
3 LancasterStemmer
4 nltk.download('punkt')
5 words = ["being", "was", "jumping", "jumps", "running", "books", "best",
6           "happily", "better"]
7 porter_stemmer = PorterStemmer()
8 snowball_stemmer = SnowballStemmer("english")
9 lancaster_stemmer = LancasterStemmer()
10
11
12 print("Word\t\tPorter\tSnowball\tLancaster")
13 print("-" * 50)
14
15 for word in words:
16     porter = porter_stemmer.stem(word)
17     snowball = snowball_stemmer.stem(word)
18     lancaster = lancaster_stemmer.stem(word)
19
20     print(f"{word}\t\t{porter}\t\t{snowball}\t\t{lancaster}")
```



Stemming Örnek



Word	Porter	Snowball	Lancaster

3 being	be	be	being
4 was	wa	was	was
5 jumping	jump	jump	jump
6 jumps	jump	jump	jump
7 running	run	run	run
8 books	book	book	book
9 best	best	best	best
10 happily	happili	happili	happy
11 better	better	better	bet

Lemmatization Örnek



```
1 import nltk
2 from nltk.stem import WordNetLemmatizer
3 from textblob import Word
4 import spacy
5 nltk.download('wordnet')
6 lemmatizer = WordNetLemmatizer()
7 words = ["being", "jumping", "jumps", "running", "books", "best",
       "happily", "better", "corpora"]
8 nltk_lemmas = [lemmatizer.lemmatize(word) for word in words]
9 print("NLTK Lemmatization:", nltk_lemmas)
10 nlp = spacy.load("en_core_web_sm")
11 spacy_lemmas = [token.lemma_ for token in nlp(" ".join(words))]
12 print("spaCy Lemmatization:", spacy_lemmas)
13 textblob_lemmas = [Word(word).lemmatize() for word in words]
14 print("TextBlob Lemmatization:", textblob_lemmas)
```

Lemmatization Örnek



```
1
2 NLTK Lemmatization: ['being', 'jumping', 'jump', 'running', ' '
  book', 'best', 'happily', 'better', 'corpus']
3 spaCy Lemmatization: ['be', 'jump', 'jump', 'run', 'book', ' '
  well', 'happily', 'well', 'corpora']
4 TextBlob Lemmatization: ['being', 'jumping', 'jump', 'running',
  'book', 'best', 'happily', 'better', 'corpus']
```

- Bilgisayarlar büyük ve küçük harfleri farklı şekilde saklarlar (örneğin, ASCII kodları)
- Küçük harfe dönüşüm: Tüm metni küçük harfe dönüştürmek, büyük ve küçük harfli karakterlerin uyumlu hale getirilmesini sağlar
- 'okul' kelimesini 'Okul' ile eşit olarak kabul edilmesini sağlar
- İstatistiksel gücünü artırır
- Derlemde büyük harfle yazıldığında veya yazılmadığında anlamsal olarak farklılık gösteren kelimeler
 - adalet - Adalet (özel isim)

```
1 import re
2 import spacy
3 import nltk
4 nltk.download('punkt')
5 from nltk.tokenize import word_tokenize
6 nlp = spacy.load("en_core_web_sm")
7 text = "Hello World"
8
9 lowercase_text1 = text.lower()
10 lowercase_text2 = text.casifold()
11 lowercase_text3 = ''.join([char.lower() for char in text])
12 lowercase_text5 = re.sub(r'[A-Z]', lambda x: x.group(0).lower()
13 ,text)
14 doc = nlp(text)
15 lowercase_text6 = " ".join(token.text.lower() for token in doc)
16 tokens = word_tokenize(text)
17 lowercase_text7 = " ".join(word.lower() for word in tokens)
```

- Metin analizi tekniğidir
- Metin verilerini, kelimelerin sırasını ve yapısını göz ardı ederek metni kelime sıklığına odaklayan bir temsil şeklidir
- Metin sınıflandırma, duygusal analizi

Adımlar

- Ön İşlemler (probleme göre)
- Tokenization
- Sözlük oluşturma
- Kelime sayılarının bulunması

Örnek Cümle

Hello, world! This is an example of BoW. Hello, Kocaeli.

Ön İşlemler: Lowercase conversion, removal punctuation

hello world this is an example of bow hello kocaeli

Tokenization

["hello", "world", "this", "is", "an", "example", "of", "bow", "hello", "kocaeli]

Sözlük oluşturma

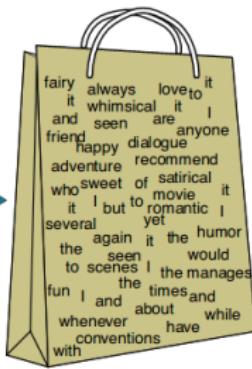
[“hello”, “world”, “this”, “is”, “an”, “example”, “of”, “bow”, “kocaeli”]

Kelime sayılarının bulunması

hello: 2 world: 1 this: 1 is: 1 an: 1 example: 1 of: 1 bow: 1 kocaeli: 1

Bag-of-Words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

Bag-of-Words



- Doc1: Text mining is to identify useful information.
- Doc2: Useful information is mined from text.
- Doc3: Apple is delicious.

	text	information	identify	mining	mined	is	useful	to	from	apple	delicious
Doc1	1	1	1	1	0	1	1	1	0	0	0
Doc2	1	1	0	0	1	1	1	0	1	0	0
Doc3	0	0	0	0	0	1	0	0	0	1	1

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 documents = [
3     "This is the first document.",
4     "This document is the second document.",
5     "And this is the third one.",
6     "Is this the first document?"
7 ]
8 vectorizer = CountVectorizer()
9 X = vectorizer.fit_transform(documents)
10 vocabulary = vectorizer.get_feature_names_out()
11 bow_matrix = X.toarray()
12 import pandas as pd
13 bow_df = pd.DataFrame(bow_matrix, columns=vocabulary)
14 print("Vocabulary:")
15 print(vocabulary)
16 print("\nBag of Words Matrix:")
17 print(bow_df)
```

Vocabulary:

```
['and' 'document' 'first' 'is' 'one' 'second' 'the' 'third' 'this']
```

Bag of Words Matrix:

	and	document	first	is	one	second	the	third	this	
0	0		1	1	0		0	1	0	1
1	0		2	0	1	0	1	1	0	1
2	1		0	0	1	1	0	1	1	1
3	0		1	1	1	0	0	1	0	1

- Terim Frekansı, bir terimin (kelimenin) bir dokümanda ne sıklıkta görüldüğünü ölçer. Bir terimin bir dokümanda çok görünmesi durumunda, bu terimlerin o doküman için daha önemli olduğunu varsayar.
- Ters Belge Frekansı, bir terimin tüm korpus içindeki önemini ölçer. Birçok dokümanda yaygın olarak bulunan terimlere daha düşük bir ağırlık vermektedir, çünkü bu terimler daha az ayırt edicidir.
- $\text{TF-IDF}(\text{terim}, \text{doküman}, \text{korpus}) = \text{TF}(\text{terim}, \text{doküman}) \times \text{IDF}(\text{terim}, \text{corpus})$

- $\text{TF}(\text{terim}, \text{doküman}) = \frac{\text{Terimin dokümda görüldüğü sayı}}{\text{Dokümandaki toplam terim sayısı}}$
- $\text{IDF}(\text{terim}, \text{corpus}) = \log \left(\frac{\text{corpus içindeki toplam doküman sayısı}}{\text{Terimi içeren doküman sayısı}} \right)$
- $\text{TF-IDF}(\text{terim}, \text{doküman}, \text{korpus}) = \text{TF}(\text{terim}, \text{doküman}) \times \text{IDF}(\text{terim}, \text{corpus})$

```
1 import pandas as pd
2 from sklearn.model_selection import StratifiedKFold
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 from sklearn.naive_bayes import MultinomialNB
5 from sklearn.metrics import accuracy_score
```

```
1 data = {  
2     'text': ["This is a positive statement", "I am feeling  
great today", "This is a negative sentiment", "I'm upset  
and frustrated", "Feeling happy and content", "Not in a  
good mood", "A positive outlook on life", "Feeling down  
and sad", "Optimistic about the future", "I hate this  
weather", "A wonderful day ahead", "Dealing with  
disappointment", "Feeling positive and motivated", "I'm  
in a bad mood today", "Positive vibes for success"],  
3     'label': ["positive", "positive", "negative", "negative", "  
positive", "negative", "positive", "negative", "positive",  
"negative", "positive", "negative",  
4         "positive", "negative", "positive"]  
5 }
```

```
1 import pandas as pd
2 from sklearn.model_selection import StratifiedKFold
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 from sklearn.naive_bayes import MultinomialNB
5 from sklearn.metrics import accuracy_score
```

```
1 df = pd.DataFrame(data)
2 tfidf_vectorizer = TfidfVectorizer(stop_words='english')      #
    durum 1
3 #tfidf_vectorizer = TfidfVectorizer()      #durum2
4 kf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
5
6 classifier = MultinomialNB()
7 accuracy_scores = []
```

```
1 for train_index, test_index in kf.split(df['text'], df['label']):  
2     X_train, X_test = df['text'][train_index], df['text'][test_index]  
3     y_train, y_test = df['label'][train_index], df['label'][test_index]  
4  
5     X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)  
6     X_test_tfidf = tfidf_vectorizer.transform(X_test)  
7     classifier.fit(X_train_tfidf, y_train)  
8  
9     y_pred = classifier.predict(X_test_tfidf)  
10  
11    accuracy = accuracy_score(y_test, y_pred)  
12    accuracy_scores.append(accuracy)  
13  
14 for i, accuracy in enumerate(accuracy_scores):  
15     print(f"Fold {i + 1} Accuracy: {accuracy}")
```

Durum 1

Fold 1 Accuracy: 0.8

Fold 2 Accuracy: 0.6

Fold 3 Accuracy: 0.4

Top 5 words: ['bad', 'content', 'dealing', 'disappointment', 'frustrated']

Durum 2

Fold 1 Accuracy: 0.4

Fold 2 Accuracy: 0.4

Fold 3 Accuracy: 0.6

Top 5 words: ['about', 'bad', 'content', 'dealing', 'disappointment']

Olasılık Teorisi: Bir olayın ne kadar muhtemel olduğunu tahmin etmek.

- **Olasılıklar:** 0 ile 1 arasındaki sayılar.

- **Olasılık Fonksiyonu:**

- $P(A)$, A 'nın gerçekleşme olasılığını ifade eder.
- $P(A)$, 0 ile 1 arasında bir sayıdır.
- $P(A) = 1 \Rightarrow$ Kesin bir olay.
- $P(A) = 0 \Rightarrow$ İmkansız bir olay.

Örnek: Bir madeni parayı üç kez atalım. 2 kez tura gelme olasılığı nedir?

- 3/8

Olasılık

- $P(A)$
- Olay A 'nın olasılığı diğer olaylardan bağımsızdır.

Koşullu Olasılık

- $P(A|B)$
- Bu, B 'yi bildiğimiz durumda A 'nın olasılığı olarak okunur.

Örnek:

- $P(\text{put})$, bir metinde "put" kelimesini görmek olasılığıdır.
- $P(\text{on}|\text{put})$, "put" kelimesini gördükten sonra "on" kelimesini görme olasılığıdır.

Kelime dizilerine olasılık atayan modeller

- Cümlelere ve kelime dizilerine olasılık atayan en basit dil modeli n-gram'dır.
- Bir n-gram, N kelime dizisidir:
 - 1-gram (unigram) "lütfen" veya "dön" gibi tek kelime dizisidir.
 - 2-gram (bigram) iki kelime dizisidir, örneğin "lütfen dön", "gitme lütfen" veya "ev ödevi" gibi.
 - 3-gram (trigram) üç kelime dizisidir, örneğin "lütfen sağa dönünüz", "bugün okula gidelim" veya "ev ödevini yapmalısın" gibi.
- n-gram modellerini kullanarak bir n-gram'ın son kelimesinin önceki kelimeleme dayalı olasılığını tahmin edebiliriz ve aynı zamanda tüm kelime dizilerine olasılıklar atayabiliriz.

Olasılıksal dil modelleri, birçok NLP görevinde bir cümleye olasılık atamak için kullanılabilir.

- Makine Çevirisi:

- $P(\text{high winds tonight}) > P(\text{large winds tonight})$

- Yazım Düzeltme:

- "Thek office is about ten minutes from here"
 - $P(\text{The Office is}) > P(\text{Then office is})$

- Özetleme, soru cevaplama, ...

Amacımız bir cümlenin veya kelime dizisinin olasılığının hesaplanması:

- $P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots, w_n)$

Bir sonraki kelimenin olasılığı nedir?:

- $P(w_5 | w_1, w_2, w_3, w_4)$

Her ikisini hesaplayabilen bir model:

- $P(W)$ veya $P(w_n | w_1, w_2, \dots, w_{n-1})$ olarak adlandırılan bir dil modelidir.

Bir cümlenin w_1, w_2, w_n gibi kelime dizilerinin olasılıklarını nasıl hesaplayabiliriz?

- Kelime dizisi w_1, w_2, w_n 'nin olasılığı $P(w_1, w_2, w_n)$ 'dir.

Zincir Kuralı

$$P(w_1 \dots w_n) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1 w_2) \cdot \dots \cdot P(w_n|w_1 \dots w_{n-1}) \quad (1)$$

Örnek: $P(\text{the man from world}) = P(\text{the})P(\text{man}|\text{the})P(\text{from}|\text{the man})P(\text{world}|\text{the man from})$

Uzun bir kelime dizisinden bir kelimenin kesin olasılığını hesaplamak zordur (bazen imkansızdır).

- $P(w_n|w_1, \dots, w_{n-1})$ 'yi hesaplamaya çalışıyoruz, ki bu w_1 den w_{n-1} 'i gördükten sonra w_n 'yi görme olasılığıdır.
- $P(w_n|w_1, \dots, w_{n-1})$ 'yi aşağıdaki gibi kesin olarak hesaplamaya çalışabiliriz: $P(w_n|w_1, \dots, w_{n-1}) = \frac{\text{count}(w_1, \dots, w_{n-1}, w_n)}{\text{count}(w_1, \dots, w_{n-1})}$

Çok sayıda olası cümle ve bu olasılık değerlerini tahmin etmek için yeterli verimiz yoksa... Bu nedenle, $P(w_n|w_1, \dots, w_{n-1})$ 'yi yaklaşık olarak hesaplamamız gerekiyor.

$P(w_n | w_1, \dots, w_{n-1}) \approx P(w_n)$ (unigram)

$P(w_n | w_1, \dots, w_{n-1}) \approx P(w_n | w_{n-1})$ (bigram)

$P(w_n | w_1, \dots, w_{n-1}) \approx P(w_n | w_{n-1} w_{n-2})$ (trigram)

$P(w_n | w_1, \dots, w_{n-1}) \approx P(w_n | w_{n-1} w_{n-2} w_{n-3})$ (4-gram)

$P(w_n | w_1, \dots, w_{n-1}) \approx P(w_n | w_{n-1} w_{n-2} w_{n-3} w_{n-4})$ (5-gram)

Unigram

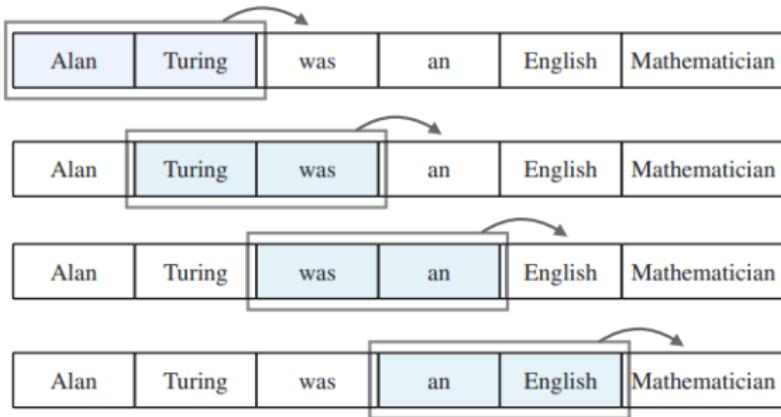
$$\begin{aligned} P(<\text{s}> \text{ the man from jupiter came } </\text{s}>) &\approx \\ P(\text{the}) \, P(\text{man}) \, P(\text{from}) \, P(\text{jupiter}) \, P(\text{came}) \end{aligned}$$

Bigram

$$\begin{aligned} P(<\text{s}> \text{ the man from jupiter came } </\text{s}>) &\approx \\ P(\text{the}|<\text{s}>) \, P(\text{man}|\text{the}) \, P(\text{from}|\text{man}) \, P(\text{jupiter}|\text{from}) \, P(\text{came}|\text{jupiter}) \, P(</\text{s}>|\text{came}) \end{aligned}$$

Trigram

$$\begin{aligned} P(<\text{s}> \text{ the man from jupiter came } </\text{s}>) &\approx \\ P(\text{the}|<\text{s}> \, <\text{s}>) \, P(\text{man}|<\text{s}> \, \text{the}) \, P(\text{from}|\text{the man}) \, P(\text{jupiter}|\text{man from}) \\ P(\text{came}|\text{from jupiter}) \, P(</\text{s}>|\text{jupiter came}) \, P(</\text{s}>|\text{came } </\text{s}>) \end{aligned}$$



<s> I am Sam </s>
<s> Sam I am </s>
<s> I fly </s>

- **Unique words:** I, am, Sam, fly
- **Bigrams:** <s> and </s> are also tokens. There are 6(4+2) tokens and $6 \times 6 = 36$ bigrams

$P(I <s>) = 2/3$	$P(Sam <s>) = 1/3$	$P(am <s>) = 0$	$P(fly <s>) = 0$	$P(<s> <s>) = 0$	$P(</s> <s>) = 0$
$P(I I) = 0$	$P(Sam I) = 0$	$P(am I) = 2/3$	$P(fly I) = 1/3$	$P(<s> I) = 0$	$P(</s> I) = 0$
$P(I am) = 0$	$P(Sam am) = 1/2$	$P(am am) = 0$	$P(fly am) = 0$	$P(<s> am) = 0$	$P(</s> am) = 1/2$
$P(I Sam) = 1/2$	$P(Sam Sam) = 0$	$P(am Sam) = 0$	$P(fly Sam) = 0$	$P(<s> Sam) = 0$	$P(</s> Sam) = 1/2$
$P(I fly) = 0$	$P(Sam fly) = 0$	$P(am fly) = 0$	$P(fly fly) = 0$	$P(<s> fly) = 0$	$P(</s> fly) = 1$
$P(I </s>) = 0$	$P(Sam </s>) = 1/3$	$P(am </s>) = 1/3$	$P(fly </s>) = 1/3$	$P(<s> </s>) = 0$	$P(</s> </s>) = 0$

Unigrams

$$P(I) = 3/8$$

$$P(am) = 2/8$$

$$P(Sam) = 2/8$$

$$P(fly) = 1/8$$

Trigrams

$$P(I|< s > < s >) = 2/3$$

$$P(Sam|< s > < s >) = 1/3$$

$$P(am|< s > I) = 1/2$$

$$P(fly|< s > I) = 1/2$$

$$P(I|< s > Sam) = 1$$

$$P(Sam|I am) = 1/2$$

$$P(< /s > |I am) = 1/2$$

$$P(< /s > |am Sam) = 1$$

$$P(< /s > |Sam < /s >) = 1$$

Corpus: Berkeley Restaurant Project Sentences

- There are 9222 sentences in the corpus.
- Raw bigram counts of 8 words (out of 1446 words)

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

- Unigram counts:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Normalize bigrams by unigram counts:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0.52	0.0063	0	
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0.0029	0	0	
spend	0.0036	0	0.0036	0	0	0	0	0

- Some other bigrams:

$$P(i|<s>) = 0.25$$

$$P(\text{english}|\text{want}) = 0.0011$$

$$P(\text{food}|\text{english}) = 0.5$$

$$P(</s>|\text{food}) = 0.68$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

- Compute the probability of sentence **I want English food**

$$P(< s > \text{ i want english food } < /s >)$$

$$= P(i|<s>) P(want|i) P(\text{english}|want) P(\text{food}|\text{english}) P(</s>|\text{food})$$

$$= 0.25 * 0.33 * 0.0011 * 0.5 * 0.68$$

$$= 0.000031$$

Perplexity, bir dil modelinin tahminlerinin gerçek verilere ne kadar uygun olduğunu ölçer.

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_{i-1})}}$$

....

Bir dil modelinin perplexity değeri mümkün olduğunca düşük olmalıdır.

Shakespeare 884647 token, 844 milyon bigram ihtimali içermektedir.
99.96%'sı bulunmuyor.

Smoothing: 1 eklenecek hesaplama yapılrsa 0'ların problemi çözülür.

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

$$\text{PP}(W) = \sqrt[N]{\prod_{i=1}^N \frac{1}{P(w_i|w_{i-1})}}$$

...

Bir dil modelinin perplexity değeri mümkün olduğunca düşük olmalıdır.

```
bi_gram_model = {  
    ("Ben", "bir"): ["kitap", "öğrenciyim", "insanım"],  
    ("Türkiye'de", "hava"): ["bugün", "sıcak", "güzel"],  
    ("hava", "bugün"): ["güzel"],  
    ("Bu", "akşam"): ["sinemaya", "dışarı"],  
    ("bir", "kitap"): ["aldım", "okudum"],  
    ("akşam", "dışarı"): ["çıkacağım"],  
    ("akşam", "sinemaya"): ["gideceğim"]  
}
```

```
1 import random
2 #veri
3 current_gram = ("Türkiye'de", "hava")
4 result = list(current_gram)
5
6 for i in range(10):
7     next_words = bi_gram_model.get(current_gram)
8     if next_words:
9         next_word = random.choice(next_words)
10        result.append(next_word)
11        current_gram = (current_gram[1], next_word)
12    else:
13        break
14
15 generated_text = " ".join(result)
16 print(generated_text)
```

Başlangıç: "Türkiye'de", "hava"

Türkiye'de hava bugün güzel

Türkiye'de hava sıcak

Türkiye'de hava güzel

Top 5 words: ['bad', 'content', 'dealing', 'disappointment', 'frustrated']

Durum 2

Fold 1 Accuracy: 0.4

Fold 2 Accuracy: 0.4

Fold 3 Accuracy: 0.6

Top 5 words: ['about', 'bad', 'content', 'dealing', 'disappointment']

Bir metnin önceden belirlenen sınıflardan hangisine ya da hangilerine ait olduğunu belirlenmesi

- Duygu Analizi

- Yazarın bir nesne hakkında ifade ettiği duygunun (pozitif-negatif) tespit edilmesi.

- Spam Tespiti

- Bir e-postanın spam olup olmadığını belirlenmesi.

- Yazar Tanımlama

- Bir metin yazarının belirlenmesi.

- Yaş/Cinsiyet Tanımlama

- Metnin yazarı ile ilgili bilgilerin tespit edilmesi

- Dil Tanımlama

- Metnin dilinin belirlenmesi.

Metin sınıflandırma şu şekilde tanımlanabilir:

■ Girdi:

- Bir belge d
- Bir sınıf kümesi $C = \{c_1, c_2, \dots, c_n\}$

■ Çıktı:

- Tahmin edilen sınıf $c \in C$

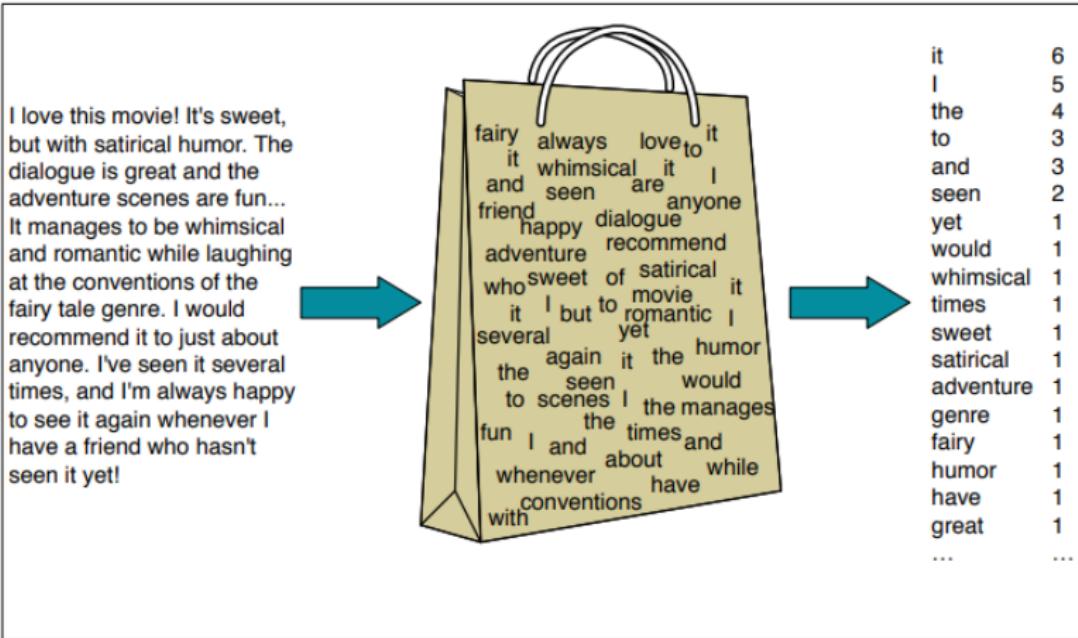
- Sınıflandırmanın amacı, tek bir gözlemi almak (metin), bazı faydalı özellikleri çıkarmak ve gözlemi bir dizi belirli sınıftan birine atamaktır.
- Metin sınıflandırma için kullanılan yöntemlerden birisi **handwritten rules** olabilir.
- Kelimelerin veya diğer özelliklerin kombinasyonlarına dayalı kurallar
- Kurallar iyi belirlenirse doğruluk yüksek olabilir.
- El yazısı kuralların oluşturulması ve sürdürülmesi maliyetli olabilir:
 - Kurallar kırılgan olabilir.
 - Alan bilgisi gerekebilir.

- Metin sınıflandırma genellikle denetimli makine öğrenmesi yöntemleri kullanılır.
- Denetimli makine öğrenmesi yöntemlerinin amacı yeni bir gözlemi doğru tahmin etmektir.
- Girdi:
 - Bir belge d
 - Sınıf kümesi $C = \{c_1, c_2, \dots, c_n\}$
 - m adet etiketlenmiş eğitim kümesi: $(d_1, c_1), \dots, (d_m, c_m)$
- Çıktı:
 - sınıflandırıcı model: $d \rightarrow c$

- Amacımız, yeni bir belgenin (d) doğru sınıf c sınıfına ($c \in C$) eşlemesini yapabilen bir sınıflandırıcıyı öğrenmektir.
- Olasılığa dayalı bir sınıflandırıcı, gözlemin sınıf olasılıklarını söyler
 - Bir gözlem verildiğinde, gözlemi üretme olasılığı en yüksek olan sınıfı belirler
- Bazı sınıflandırıcılar şunlardır:
 - Naive Bayes
 - Lojistik regresyon
 - Destek Vektör Makineleri
 - k-En Yakın Komşular

- Bir Naive Bayes Sınıflandırıcı, özelliklerin nasıl etkileşime girdiği hakkında bir varsayıım yapan Bayes kuralına dayalı bir sınıflandırıcıdır.
- Bir metin belgesi, kelime çantası olarak temsil edilir.
 - Bir metin belgesi, sıralamaları göz ardı edilerek yalnızca metindeki frekanslarına göre temsil edilir.

Naive Bayes Sınıflandırıcı



- Bir belge d için tüm sınıflar $c \in C$ olmak üzere, belge verildiğinde sınıfı \hat{c} olarak döndüren sınıflandırıcı, belgeye verilen sonraki olasılığı maksimize eden sınıfı seçer. Burada, " \hat{c} " işaretini, doğru sınıfın tahminini temsil eder.
- Bir belge d ve bir sınıf c için, en olası sınıf (MAP: maksimum a posteriori):

$$\hat{c} = c_{MAP} = \arg \max_{c \in C} P(c|\mathbf{d}) \quad (2)$$

- İlgili formül:

$$P(c|\mathbf{d}) = \frac{P(c) \cdot P(\mathbf{d}|c)}{P(\mathbf{d})}$$

- Bayes Kuralları ile:

$$\hat{c} = \arg \max_{c \in C} \frac{P(c) \cdot P(\mathbf{d}|c)}{P(\mathbf{d})} \quad (3)$$

- Paydanın (tüm sınıflar için aynı olduğu) atılması:

$$\hat{c} = \arg \max_{c \in C} P(c) \cdot P(\mathbf{d}|c) \quad (4)$$

- İlgili terimler:

- Belgenin olasılığı (likelihood of the document)
- Sınıfın öncelik olasılığı (prior probability of the class)

- Bir belge d 'yi f_1, f_2, \dots, f_n özelliklerinin bir kümesi olarak temsil edebiliriz:

$$\hat{c} = \arg \max_{c \in C} P(f_1, f_2, \dots, f_n | c) \cdot P(c) \quad (5)$$

- Hesaplama maliyetli. Her olası kombinasyonun olasılığını hesaplamak, büyük sayıda parametre ve büyük eğitim verileri gerektirir.
- Naive Bayes sınıflandırıcıları varsayılmış yapar (basitleştirmek için):
 - Kelime Çantası Varsayımları: Pozisyonun önemli olmadığı varsayımları.
 - Naive Bayes Varsayımları: Koşullu bağımsızlık varsayımları yapar; yani $P(f_i | c)$ olasılıkları, sınıf c verildiğinde bağımsızdır ve bu nedenle aşağıdaki gibi çarpılabilir:

$$P(f_1, \dots, f_n | c) = P(f_1 | c) \cdot \dots \cdot P(f_n | c) \quad (6)$$

$$\arg \max_{c \in C} P(c) \prod_{f \in F} P(f|c) \quad (7)$$

- Kelime ağırlıklarını dahil edecek olursak:

$$c_{\text{NB}} = \arg \max_{c \in C} P(c) \prod_{i \in \mathcal{P}} P(w_i|c) \quad (8)$$

$$c_{\text{NB}} = \arg \max_{c \in C} \left(\log P(c) + \sum_{i \in \mathcal{P}} \log P(w_i|c) \right) \quad (9)$$

- Olasılıklar $P(c)$ ve $P(w_i|c)$ nasıl öğrenilir?
- Eğitim veri kümelerinden öğreniriz: $(d_1, c_1), \dots, (d_m, c_m)$.
- $P(c)$, aşağıdaki gibi hesaplanabilir:

$$P(c) = \frac{N_c}{N_{\text{doc}}} \quad (10)$$

- Burada, N_c , sınıf c ile etiketlenmiş belgelerin sayısıdır ve N_{doc} , toplam belge sayısıdır.

- $P(w_i|c)$ hesaplamak için:

- c kategorisi ile etiketlenmiş tüm belgeleri bir büyük "kategori c " metni olarak birleştirin.
- Daha sonra birleştirilmiş belgedeki w_i kelimesinin frekansını kullanarak olasılığın maksimum olasılık tahminini hesaplayın.

$$P(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_{w \in \mathcal{V}} \text{count}(w, c)} \quad (11)$$

- \mathcal{V} tüm sınıflardaki kelimelerin birleşimidir.

- $P(\text{güzel kelimesi}|\text{pozitif kategori verildiğinde})$:

$$P(\text{güzel}|\text{pozitif}) = \frac{\text{count}(\text{güzel, pozitif})}{\sum_{w \in \mathcal{V}_{\text{pozitif}}} \text{count}(w, \text{pozitif})} \quad (12)$$

- $P(w_i|c)$ için sıfır olasılık değerlerinden kurtulmak için bir düzenleme yöntemi kullanılmalı.
- En basit çözüm, bir ekleme (Laplace) düzenlemesidir:

$$P(w_i|c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in \mathcal{V}} (\text{count}(w, c) + 1) + |\mathcal{V}|} \quad (13)$$

- Eğitim belgelerinde hiç görünmeyen ve kelime dağarcığında bulunmayan test verilerindeki kelimeler hakkında ne yapmalıyız?
- Bilinmeyen kelimeler için çözüm, onları yok saymaktır — test belgesinden kaldırırmak ve onlar için hiçbir olasılığı dahil etmemek.
- Bazı sistemler başka bir kelime sınıfını tamamen görmezden gelmeyi tercih eder (stopwords)

```
function TRAIN NAIVE BAYES(D,C) returns log P(c) and log P(w|c)
    for each class c ∈ C           # Calculate P(c) terms
        Ndoc = number of documents in D
        Nc = number of documents from D in class c
        logprior[c] ← log  $\frac{N_c}{N_{doc}}$ 
        V ← vocabulary of D
        bigdoc[c] ← append(d) for d ∈ D with class c
    for each word w in V           # Calculate P(w|c) terms
        count(w,c) ← # of occurrences of w in bigdoc[c]
        loglikelihood[w,c] ← log  $\frac{count(w,c) + 1}{\sum_{w' \text{ in } V} (count(w',c) + 1)}$ 
    return logprior, loglikelihood, V

function TEST NAIVE BAYES(testdoc, logprior, loglikelihood, C, V) returns best c
    for each class c ∈ C
        sum[c] ← logprior[c]
    for each position i in testdoc
        word ← testdoc[i]
        if word ∈ V
            sum[c] ← sum[c] + loglikelihood[word,c]
    return argmaxc sum[c]
```

Gerçek film incelemelerinden uyarlanmış aşağıdaki eğitim ve test belgelerini içeren duyu analizi problemi:

	Cat	Documents
Training	-	<ul style="list-style-type: none">just plain boring- entirely predictable and lacks energy- no surprises and very few laughs+ very powerful+ the most fun film of the summer
Test	?	predictable with no fun

Sınıfların olasılıkları ($P(c)$):

$$P(-) = \frac{3}{5} \quad P(+) = \frac{2}{5}$$

- 'with' eğitim setinde bulunmuyor, bu yüzden onu tamamen çıkarıyoruz.
- Eğitim setinde kalan 'predictable', 'no' ve 'fun' kelimeleri için olasılıklar şunlardır:"

$$P(\text{"predictable"}|-) = \frac{1+1}{14+20} \quad P(\text{"predictable"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"no"}|-) = \frac{1+1}{14+20} \quad P(\text{"no"}|+) = \frac{0+1}{9+20}$$

$$P(\text{"fun"}|-) = \frac{0+1}{14+20} \quad P(\text{"fun"}|+) = \frac{1+1}{9+20}$$

- Örnek cümle için seçilen sınıf negatiftir.

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+|S)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

- Duygu analizi gibi problemlerde, kelime görünürlüğü kelime sıklığından daha önemli olabilir.
- "good" kelimesinin görünmesi bize çok şey anlatır. Ancak kelimenin 5 kez görünmesi bize çok fazla bilgi vermeyebilir.
- "Binary Multinomial Naive Bayes" kelime sayılarını 1 olarak alır.

Four original documents:

- it was pathetic the worst part was the boxing scenes
- no plot twists or great scenes
- + and satire and great plot twists
- + great scenes great film

After per-document binarization:

- it was pathetic the worst part boxing scenes
- no plot twists or great scenes
- + and satire great plot twists
- + great scenes film

	NB Counts		Binary Counts	
	+	-	+	-
and	2	0	1	0
boxing	0	1	0	1
film	1	0	1	0
great	3	1	2	1
it	0	1	0	1
no	0	1	0	1
or	0	1	0	1
part	0	1	0	1
pathetic	0	1	0	1
plot	1	1	1	1
satire	1	0	1	0
scenes	1	2	1	2
the	0	2	0	1
twists	1	1	1	1
was	0	2	0	1
worst	0	1	0	1

Örnek

I really like this movie

I really don't like this movie

Olumlu Cümle

Don't dismiss this film

Doesn't let us get bored

Çözüm

didn't like this movie, but I

didn't NOT_like NOT_this NOT_movie but I

Dil Modeli: Naive Bayes



Model pos

<u>P(w pos)</u>	w
0.1	I
0.1	love
0.01	this
0.05	fun
0.1	film

Model neg

<u>P(w neg)</u>	w
0.2	I
0.001	love
0.01	this
0.005	fun
0.1	film

I	love	this	fun	film
0.1	0.1	0.01	0.05	0.1
0.2	0.001	0.01	0.005	0.1

$$P(s|pos) = 0.0000005$$

$$P(s|neg) = 0.000000001$$

$$P(s|pos) > P(s|neg)$$

- Sınıflandırıcının ne kadar iyi olduğunu değerlendirmek için kullanılabilecek ölçütler
- Değerlendirme sırasında Sınıflandırıcının test seti sonuçları gold-standard etiketlerle karşılaştırılır
- Bu karşılaştırmanın sonucunda, ölçütlerimizi hesaplamadan önce bir contingency table oluşturulur

		<i>gold standard labels</i>		
		gold positive	gold negative	
<i>system output labels</i>	system positive	true positive	false positive	precision = $\frac{tp}{tp+fp}$
	system negative	false negative	true negative	
		recall = $\frac{tp}{tp+fn}$		accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$

- Kesinlik (Precision) ve Duyarlılık (Recall), Doğru Pozitifleri (True Positives) ölçer. Sadece birini incelemek yanıltıcı olabilir.
- $tp=1, fp=0, fn=99$ durumunda Kesinlik = %100 (Ancak Duyarlılık=%1)
- $tp=1, fp=99, fn=0$ durumunda Duyarlılık = %100 (Ancak Kesinlik=%1)
- F-ölçüm, kesinlik ve duyarlılığı birlestiren bir ölçüt olarak kullanılır.

$$F_{\beta} = \frac{(\beta^2 + 1) \cdot P \cdot R}{\beta^2 \cdot P + R} \quad (14)$$

- En sık kullanılan ölçüt, $F_1 = 1$ olarak adlandırılır:

$$F1 = \frac{2 \cdot P \cdot R}{P + R} \quad (15)$$

Karmaşıklık Matrisi

		gold labels			
		urgent	normal	spam	
system output	urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
	normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
	spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
		$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recall}_s = \frac{200}{1+50+200}$	

- 20 Newsgroups, internet üzerindeki haber gruplarından alınmış metin belgelerini içerir.
- **Kategoriler:** Toplamda 20 farklı kategoride metin belgelerini içerir.
- alt.atheism, comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.windows.x, misc.forsale, rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey, sci.crypt, sci.electronics, sci.med, sci.space, soc.religion.christian, talk.politics.guns, talk.politics.mideast, talk.politics.misc, talk.religion.misc
- Toplam belge sayısı: 18846
Toplam kelime sayısı: 3423145

```
1 import nltk
2 nltk.download('stopwords')
3 nltk.download('punkt')
4 import re
5 import numpy as np
6 from sklearn.datasets import fetch_20newsgroups
7 from sklearn.feature_extraction.text import TfidfVectorizer
8 from sklearn.naive_bayes import MultinomialNB
9 from sklearn.model_selection import train_test_split
10 from sklearn.metrics import accuracy_score,
11     classification_report
12 from nltk.corpus import stopwords
13 from nltk.stem import PorterStemmer
14 from nltk.tokenize import word_tokenize
15 from sklearn.metrics import precision_score, recall_score,
16     f1_score
```

```
1 def preprocess_text(text):
2     text = text.lower()
3     text = re.sub(r'[^a-zA-Z\s]', ' ', text)
4     words = word_tokenize(text)
5     stemmer = PorterStemmer()
6     words = [stemmer.stem(word) for word in words]
7     return ' '.join(words)
```

```
1 newsgroups_data = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))  
2  
3 cleaned_data = [preprocess_text(doc) for doc in newsgroups_data  
 .data]  
4  
5 tfidf_vectorizer = TfidfVectorizer(max_features=1000)  
6 X_tfidf = tfidf_vectorizer.fit_transform(cleaned_data)
```

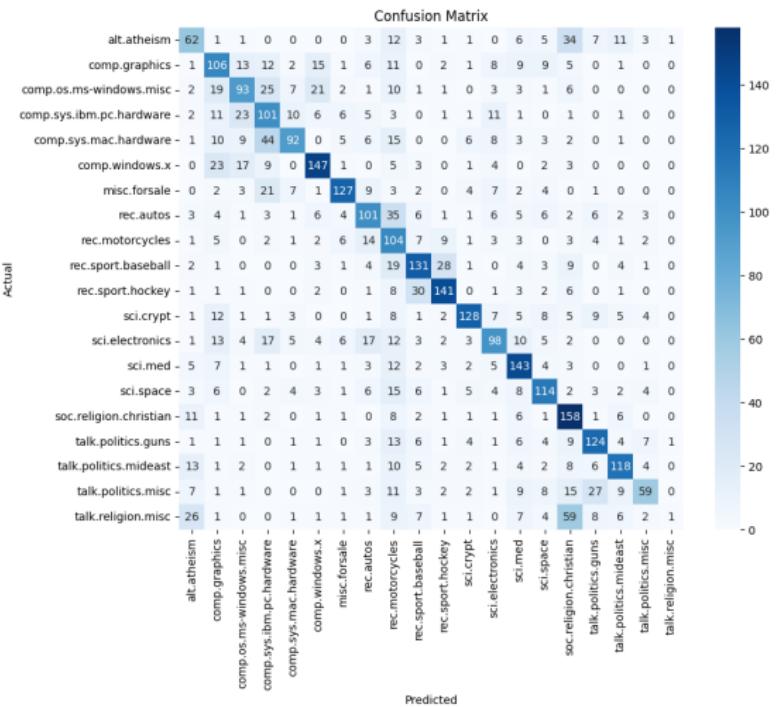
```
1 X_train, X_test, y_train, y_test = train_test_split(X_tfidf,
2                                                 newsgroups_data.target, test_size=0.2, random_state=42)
3
4
5 naive_bayes_classifier = MultinomialNB()
6 naive_bayes_classifier.fit(X_train, y_train)
7
8 y_pred = naive_bayes_classifier.predict(X_test)
```

```
1 accuracy = accuracy_score(y_test, y_pred)
2 report = classification_report(y_test, y_pred, target_names=
3     newsgroups_data.target_names)
4
5 print(f"đDoruluk: {accuracy}")
6 print(report)
7 precision = precision_score(y_test, y_pred, average=None)
8 recall = recall_score(y_test, y_pred, average=None)
9 f1 = f1_score(y_test, y_pred, average=None)
10 print(precision, recall, f1)
11 for i, target_name in enumerate(newsgroups_data.target_names):
12     print(f"Class: {target_name}")
13     print(f"Precision: {precision[i]}")
14     print(f"Recall: {recall[i]}")
15     print(f"F1-Score: {f1[i]}")
```

```
1 from sklearn.metrics import confusion_matrix
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4
5 conf_matrix = confusion_matrix(y_test, y_pred)
6
7 plt.figure(figsize=(10, 8))
8 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
9             xticklabels=newsgroups_data.target_names, yticklabels=
10            newsgroups_data.target_names)
11 plt.xlabel('Predicted')
12 plt.ylabel('Actual')
13 plt.title('Confusion Matrix')
14 plt.show()
```

Doğal Dil İşleme ve Metin Madenciliğine Giriş

	precision	recall	f1-score
alt.atheism	0.43	0.41	0.42
comp.graphics	0.47	0.52	0.50
comp.os.ms-windows.misc	0.54	0.48	0.51
comp.sys.ibm.pc.hardware	0.42	0.55	0.48
comp.sys.mac.hardware	0.68	0.45	0.54
comp.windows.x	0.68	0.68	0.68
misc.forsale	0.77	0.66	0.71
rec.autos	0.55	0.52	0.53
rec.motorcycles	0.32	0.62	0.42
rec.sport.baseball	0.60	0.62	0.61
rec.sport.hockey	0.71	0.71	0.71
sci.crypt	0.78	0.64	0.70
sci.electronics	0.58	0.49	0.53
sci.med	0.60	0.74	0.66
sci.space	0.62	0.60	0.61
soc.religion.christian	0.48	0.78	0.59
talk.politics.guns	0.63	0.66	0.65
talk.politics.mideast	0.69	0.65	0.67
talk.politics.misc	0.66	0.37	0.47
talk.religion.misc	0.33	0.01	0.01
accuracy			0.57



```
data = [
    ("Bu bir spam metnidir", "spam"),
    ("Bu ise bir spam değil", "spam değil"),
    ("Spam mesajı", "spam"),
    ("Normal bir metin", "spam değil"),
    ("Spam içerir mi?", "spam"),
    ("Bu kesinlikle spam değil", "spam değil")]
```

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 from sklearn.naive_bayes import MultinomialNB
3 from sklearn.model_selection import train_test_split
4 from sklearn.metrics import accuracy_score,
    classification_report
5
6 data = [
7     ....
8 ]
9
10
11 documents = [item[0] for item in data]
12 labels = [item[1] for item in data]
```

```
1 vectorizer = CountVectorizer(ngram_range=(2, 2))
2 X = vectorizer.fit_transform(documents)
3 print(X)
4
5 X_train, X_test, y_train, y_test = train_test_split(X, labels,
6   test_size=0.2, random_state=42)
7
8 naive_bayes_classifier = MultinomialNB()
9 naive_bayes_classifier.fit(X_train, y_train)
```

```
1 y_pred = naive_bayes_classifier.predict(X_test)
2
3
4 accuracy = accuracy_score(y_test, y_pred)
5 report = classification_report(y_test, y_pred)
6 print(f"Doruluk: {accuracy}")
7 print(report)
```

(0, 2)	1
(0, 1)	1
(0, 12)	1
(1, 1)	1
(1, 3)	1
(1, 5)	1
(1, 9)	1
(2, 11)	1
(3, 8)	1
(3, 0)	1
(4, 10)	1
(4, 6)	1
(5, 9)	1
(5, 4)	1
(5, 7)	1
	precision
spam	1.00
spam değil	1.00
	recall
spam	1.00
spam değil	1.00
	f1-score
spam	1.00
spam değil	1.00
accuracy	1.00

- Regresyon, istatistik ve makine öğrenmesinde, bir veya daha fazla bağımsız değişken ile bir bağımlı değişken arasındaki ilişkiyi modellemek için kullanılan bir yöntemler grubudur.
- Regresyon analizi, bağımsız değişkenlerin bağımlı değişken üzerindeki etkisini anlamak, tahminler yapmak veya gelecekteki eğilimleri öngörmek için kullanılır.

Reklam Harcamalarının Satışlara Etkisi: Bir Lineer Regresyon Örneği

- Bağımsız Değişken (X): Reklam Harcamaları (TL cinsinden)
- Bağımlı Değişken (Y): Satış Miktarı (adet cinsinden)
- Model aşağıdaki lineer denklem ile ifade edilebilir:

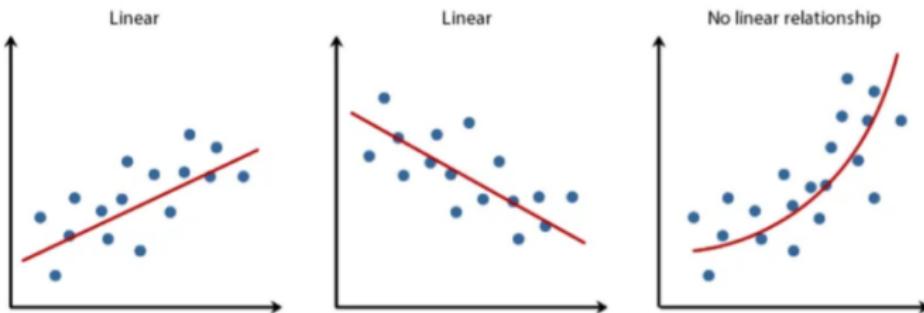
$$Y = \beta_0 + \beta_1 X + \epsilon$$

burada,

- β_0 y-intercept (X 'in sıfır olduğu yerde Y 'nin aldığı değer),
- β_1 eğim (slope) (X 'in bir birim değişiminde Y 'nin ne kadar değişeceğini belirtir),
- ϵ hata terimidir.
- Eğitimden sonra elde edilen modelimiz şu şekildedir:

$$\text{Satışlar} = 50 + 10 \times \text{Reklam Harcamaları}$$

Lineer Regresyon



- Lojistik regresyon, bağımlı değişkenin kategorik olduğu durumlar için kullanılan istatistiksel bir modelleme yöntemidir.
- Genellikle ikili sınıflandırma problemlerinde kullanılır, bağımlı değişken iki değerden birini alabilir.
- Bu yöntem, bağımsız değişkenlerin (veya özelliklerin) bir seti verildiğinde, bağımlı değişkenin olası sonuçlarının olasılıklarını tahmin etmek için kullanılır.

Lojistik Regresyon:

- Bağımlı değişken kategoriktir (örneğin, evet/hayır, 0/1).
- Olasılık tahmin eder (0 ile 1 arasında bir değer).
- Çıktıyı olasılık olarak yorumlamak için bir logit fonksiyonu kullanır.
- İkili veya çoklu sınıflandırma problemleri için uygundur.

Lineer Regresyon:

- Bağımlı değişken sürekliidir (örneğin, bir evin fiyatı, bir arabanın hızı).
- Doğrudan bir çıktı değeri tahmin eder (herhangi bir reel sayı olabilir).
- Bağımsız değişkenlerle bağımlı değişken arasındaki doğrusal ilişkiye modellemek için kullanılır.
- Tahmin edilen değerlerin reel sayılar üzerinde herhangi bir değeri olabilir, olasılık değildir.

Lojistik regresyon ve Naive Bayes modelleri arasındaki temel farklar:

■ Model Tipi:

- **Lojistik Regresyon:** Parametrik bir modeldir. Belirli sayıda parametre ile veriye en iyi uyan parametreler öğrenilir.
- **Naive Bayes:** Parametrik olmayan, olasılıksal bir modeldir. Her sınıfın ve her özelliğin olasılığını bağımsız olarak hesaplar ve Bayes teoremini kullanarak sınıflandırma yapar.

■ Özellik Bağımsızlığı:

- **Lojistik Regresyon:** Özellikler arasındaki ilişkileri ve etkileşimleri modele dahil edebilir.
- **Naive Bayes:** Özelliklerin birbirinden bağımsız olduğu varsayımini yapar. Bu varsayıım her zaman doğru olmayıabilir.

■ Tahmin Süreci:

- **Lojistik Regresyon:** Bir örneğin her bir sınıf'a ait olma olasılığını verir ve genellikle daha iyi olasılık tahminleri sunar.
- **Naive Bayes:** Doğrudan sınıflandırma yapar ve her sınıf için son olasılığı hesaplar.

■ Eğitim Süreci:

- **Lojistik Regresyon:** Modeli eğitmek için gradient descent gibi iteratif yöntemler gerektirir. Bu, büyük veri kümelerinde zaman alıcı olabilir.
- **Naive Bayes:** Hızlı eğitim sürecine sahiptir, çünkü her özelliğin sınıflara göre olasılıklarını bağımsız olarak hesaplar.

- Veri seti: Veri setinde, her bir giriş (veya gözlem) için bağımsız değişkenler ve karşılık gelen bir kategorik bağımlı değişken bulunur.
- Modelimizi oluştururken, her bir bağımsız değişkenin modeldeki önemini belirleyen katsayıları (veya ağırlıkları) hesaplamamız gereklidir.
- Her bir katsayı başlangıçta rastgele bir değerle başlatılır. Sonrasında, modelin eğitimi sırasında bu katsayılar, veri setimizle en iyi uyumu sağlayacak şekilde ayarlanır.
- Modelin eğitiminde, 'Maksimum Olabilirlik Tahmini' adı verilen bir yöntem kullanılır. Bu yöntem, gözlemlenen veriye göre model parametrelerinin 'olasılığını' maksimize etmeye çalışır.

- Maksimum olabilirlik, modelimizin tahminlerinin gerçek verilere ne kadar 'benzediğini' ölçer. Yani model, gerçekten olmuş olanları tahmin etmeye ne kadar yakınsa, o kadar 'iyi' bir modeldir.
- Eğitim sürecinde, modelin hata oranı hesaplanır ve katsayılar bu hata oranını azaltacak şekilde güncellenir.
- Modelin doğruluğu artana ve hata oranı belirli bir eşik değerin altına düşene kadar bu işlem tekrar edilir.
- Son olarak, eğitim tamamlandığında, model yeni verilerle test edilir ve performansı ölçülür.

- Olasılık Tahmini: Lojistik regresyon, bir olayın gerçekleşme olasılığını (p) ve gerçekleşmemeye olasılığını ($1-p$) modellemek için kullanılır.
- Logit Dönüşümü: Olasılığı lineer bir denkleme bağlamak için lojistik fonksiyonun tersi olan logit fonksiyonu kullanılır. Bu dönüşüm, olasılığı sınırlı aralıktan $(0, 1)$ çıkartıp tüm reel sayılar üzerinde tanımlı bir aralığa taşır.
- Bağımlı Değişken: Bağımlı değişken kategoriktir ve genellikle ikili ya da çoklu sınıflardan birine aittir.
- Maksimum Olabilirlik (Likelihood) Tahmini (MLE): Lojistik regresyon modelleri genellikle MLE kullanılarak eğitilir. MLE, gözlemlenen verinin olasılığını maksimize eden parametre değerlerini bulmaya çalışır.

Lojistik regresyon modelinin temel matematiksel formülasyonu:

- Model, verilen bir bağımsız değişken X için olasılık tahmini yapar. Bu olasılık p , belirli bir olayın meydana gelme şansını ifade eder.
- Olasılığı lineer bir modelle ilişkilendirmek için logit dönüşümü kullanılır:

$$\log \left(\frac{p}{1-p} \right) = \beta_0 + \beta_1 X$$

- Bu denklem, olasılığı sınırlı $(0, 1)$ aralıktan çıkarır ve onu tüm reel sayılar üzerinde tanımlanabilir hale getirir.

- Elde edilen lineer form, lojistik fonksiyon aracılığıyla bir olasılık değerine dönüştürülür:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

- Burada, p belirli bir sınıfın olasılığını, β_0 modelin kesim noktasını (intercept) ve β_1 modelin eğimini (slope) temsil eder.

- 1 Her bir özelliğin (feature) ağırlığı (β) ve model sabiti (β_0):

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}}$$

- 2 Rastgele başlangıç değerlerine sahip ağırlıkları (β) başlatılır.
- 3 Maksimum Olabilirlik Tahmini (MLE) kullanarak, veri setinin verdiği olasılıkları maksimize eden ağırlıklar bulunur:

$$L(\beta) = \prod_{i=1}^n p(x_i)^{y_i} (1 - p(x_i))^{(1-y_i)}$$

- 1** Gradient Decent: Maksimum olabilirlik fonksiyonunu maksimize edecek şekilde ağırlıklar güncellenir. Bu Gradyan fonksiyonu kullanılarak yapılır:

$$\nabla L(\beta) = X^T(y - p(X))$$

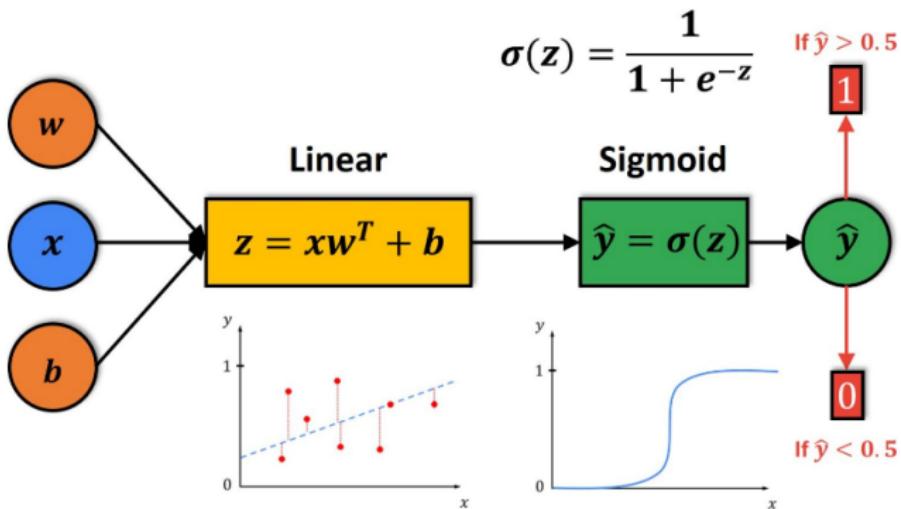
- 2** Ağırlıklar güncellenir:

$$\beta := \beta + \alpha \nabla L(\beta)$$

Burada α öğrenme oranıdır ve ne kadar hızlı "öğrenileceğini" belirler.

- 3** Yukarıdaki adımlar belirlenen bir hata oranı altına düşene kadar veya belirli bir iterasyon sayısına ulaşana kadar tekrar edilir.
- 4** Model eğitildikten sonra, yeni veri örnekleri üzerinde performans ölçülür.

Logistic Regression



Adayın Sınav Puanı (SAT) ve Lise Not Ortalaması (GPA) ile kabul edilme durumunu ($\text{Kabul}=1$, $\text{Red}=0$) ilişkilendiren lojistik bir regresyon modelidir. Modelin matematiksel formülasyonu aşağıdaki gibidir:

$$\log \left(\frac{p}{1-p} \right) = \beta_0 + \beta_1(\text{SAT}) + \beta_2(\text{GPA}) \quad (16)$$

Bu formülasyonda p , öğrencinin kabul edilme olasılığını ifade etmektedir. Eğitilen model sonucunda elde edilen parametreler aşağıdaki gibidir:

- $\beta_0 = -6$
- $\beta_1 = 0.002$
- $\beta_2 = 1$

Bir öğrencinin SAT puanı 1800 ve GPA'sı 4.0 ise, kabul olasılığı aşağıdaki gibi hesaplanır:

$$\begin{aligned}\log\left(\frac{p}{1-p}\right) &= -6 + 0.002 \times 1800 + 1 \times 4.0 \\ &= -6 + 3.6 + 4 \\ &= 1.6\end{aligned}$$

Olasılığı hesaplamak için, logit fonksiyonun tersini alırız:

$$p = \frac{1}{1 + e^{-1.6}} \approx 0.832 \tag{17}$$

Bu sonuç, öğrencinin yaklaşık olarak %83.2 ihtimalle kabul edileceğini göstermektedir.

Example: sentiment classification

- Suppose we are doing binary sentiment classification on movie review text, and we want to assign the sentiment class + or - to a review document *doc*.
- Each input observation (document) will be represented by 6 features of the input:

Var	Definition
x_1	count(positive lexicon \in doc)
x_2	count(negative lexicon \in doc)
x_3	$\begin{cases} 1 & \text{if "no" } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns \in doc)
x_5	$\begin{cases} 1 & \text{if "!" } \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Logistic Regression

- Let's assume for the moment that we've already learned a real-valued weight for each of 6 features.
 - The 6 weights corresponding to the 6 features are [2.5, -5.0, -1.2, 0.5, 2.0, 0.7], and the bias term $b = 0.1$.
 - For example, the weight w_1 indicates how the number of positive lexicon words (great, nice, enjoyable, etc.) is important to a positive sentiment decision.
- A sample mini test document showing the extracted features in the vector x .

It's hokey. There are virtually no surprises, and the writing is second-rate. So why was it so enjoyable? For one thing, the cast is great. Another nice touch is the music. I was overcome with the urge to get off the couch and start dancing. It sucked me in, and it'll do the same to you.

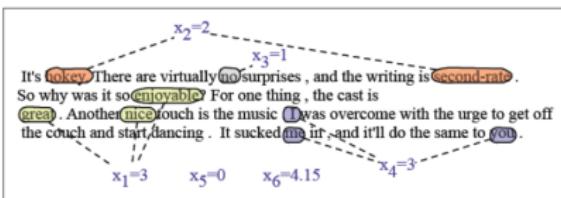
$x_1=3$ $x_2=2$ $x_3=1$ $x_4=3$ $x_5=0$ $x_6=4.15$

Logistic Regression

- Given these 6 features and the input review x , $P(+|x)$ and $P(-|x)$ can be computed:

$$\begin{aligned} p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\ &= \sigma(1.805) \\ &= 0.86 \end{aligned}$$

$$\begin{aligned} p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\ &= 0.14 \end{aligned}$$



İkili ve multinomial lojistik regresyon arasındaki temel farklar şunlardır:

■ İkili Lojistik Regresyon:

- Bağımlı değişken iki olası kategoriye (örn. evet/hayır) ayrılır.
- Olasılık tahmini için *sigmoid* fonksiyonu kullanılır.
- Model, verilen bağımsız değişkenlere göre tek bir kategorinin olasılığını hesaplar.

■ Multinomial Lojistik Regresyon:

- Bağımlı değişken üç veya fazla kategoriye (ve bu kategoriler arasında sıralama yoktur) ayrılır.
- Her kategori için olasılık tahmini yapmak için *softmax* fonksiyonu kullanılır.
- Model, bağımsız değişkenlerin her biri için tüm kategorilerin olasılıklarını hesaplar ve bu olasılıkların toplamı 1 olur.

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.model_selection import train_test_split
4 from sklearn.feature_extraction.text import TfidfVectorizer
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.metrics import accuracy_score
7
8 df = pd.DataFrame({
9     'text': ['I love this product', 'Worst service ever', 'Happy with my purchase', 'I am so angry about this', 'What a great day', 'Not a good quality'],
10    'sentiment': [1, 0, 1, 0, 1, 0] # 1: pozitif, 0: negatif
11 })
```

```
1 X_train, X_test, y_train, y_test = train_test_split(df['text'],  
         df['sentiment'], test_size=0.2, random_state=42)  
2  
3 vectorizer = TfidfVectorizer(max_features=1000)  
4 X_train_vectors = vectorizer.fit_transform(X_train)  
5 X_test_vectors = vectorizer.transform(X_test)  
6  
7  
8 model = LogisticRegression()  
9 model.fit(X_train_vectors, y_train)
```

```
1 y_pred = model.predict(X_test_vectors)
2 accuracy = accuracy_score(y_test, y_pred)
3 print(f"Model Accuracy: {accuracy*100:.2f}%")
4
5 new_reviews = ['This is an amazing product', 'I hate this
   product']
6 new_reviews_vectors = vectorizer.transform(new_reviews)
7 new_predictions = model.predict(new_reviews_vectors)
```

```
1 for review, prediction in zip(new_reviews, new_predictions):
2     print(f"Review: '{review}' Prediction: {'Positive' if
3         prediction == 1 else 'Negative'}")
4
5
6 from sklearn.metrics import classification_report
7
8 y_pred = model.predict(X_test_vectors)
9 report = classification_report(y_test, y_pred, target_names=['
    Negative', 'Positive'])
10 print(report)
```

```
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.metrics import classification_report
5 from sklearn.preprocessing import StandardScaler
6
7 iris = load_iris()
8 X, y = iris.data, iris.target
9
10 X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

```
1 scaler = StandardScaler()
2 X_train = scaler.fit_transform(X_train)
3 X_test = scaler.transform(X_test)
4
5
6 model = LogisticRegression(multi_class='multinomial', solver='
    lbfgs', max_iter=1000)
7 model.fit(X_train, y_train)
8
9 y_pred = model.predict(X_test)
10 report = classification_report(y_test, y_pred)
11 print(report)
```