AFYON KOCATEPE ÜNİVERSİTESİ

BİLGİSAYAR ANABİLİM DALI

TEZLİ YÜKSEK LİSANS PROGRAMI

VERİ MADENCİLİĞİ
BAYES ALGORİTMASI UYGULAMASI

FİNAL RAPORU

Baran Emre BULAK

230706001

Danışman

Doç. Dr. Gür Emre GÜRAKSIN

Afyonkarahisar, 2023

BAYES SINIFLANDIRMA ALGORITMASI

Bayes sınıflandırma algoritması, veri sınıflandırmada kullanılan istatistiksel bir yöntemdir. Bayes teoremi temel alınarak oluşturulan bu algoritma, veri örneklerini sınıflara ayırır.

Temel İlke :

Bayes teoremi, koşullu olasılıkları hesaplarken kullanılır ve bir olayın gerçekleşmesi durumunda başka bir olayın ne kadar olası olduğunu belirler.

Anlatımı :

$P(A|B) = (P(B|A) * P(A)) / P(B)$

Anlamı :

P(A|B): B olayının gerçekleştiği durumda A olayının gerçekleşme olasılığı

Çalışma Prensibi :

Bayes sınıflandırma algoritması, verilerdeki özelliklerin sınıflara olan uyumunu hesaplamak için Bayes teoremini kullanır. Tembel(lazy) bir öğrenme algoritmasıdır. Dengesiz veri kümelerinde de çalışabilir. Önceki veri örneklerinin sınıfları ve özellikleri bilindiğinde, algoritma yeni veri örneklerini sınıflandırmak için en yüksek olasılık sınıfını tahmin eder.

Öğrenme zamanı yoktu, yani sınıflandırmadan önce bir başlangıç zamanı gerekmez. Her sınıflandırma için tüm veri kümesini işler.

Kullanım Alanları :

- **Tekstil Endüstrisi (Kumaş kalitesi ve dokuma desenlerini sınıflandırmak için kullanılır)**
- **Tıbbi Tanı(Hastalıkların tespitinde, teşhis koyarken veya tedavi planları oluştururken kullanılır)**
- **Spam Filtreleme(E-postaları spam veya gerçek olarak sınıflandırmak için kullanılır)**

Avantajları :

- **Basit ve Hızlı**

  Bayes sınıflandırması, basit ve hızlı bir algoritmadır. Büyük veri kümeleri üzerinde bile hızlı sonuçlar üretebilir.

- **Veri Eksikliğine Dirençli**
  Veri setinde eksik değerler veya gürültüler olsa bile, algoritma etkili bir şekilde çalışabilir.
- **Yüksek Başarı Oranı**
  Bayes sınıflandırma algoritması, sınıflandırma işlemlerinde yüksek başarı oranı sağlayabilir.

Dezavantajları :

- **Aşırı Uyma**
  Az veri setleri veya dengesiz sınıf dağılımlarında aşırı uyum sorunu ortaya çıkabilir.
- **Bağımsızlık Varsayımı**

  Bayes sınıflandırması, özellikler arasında tam bağımsızlık varsayımı yapar, bu da gerçek dünyadaki verilere uygulanabilirliğini sınırlar.

- **Hesaplama Yükü**
  Bazı durumlarda, Bayes sınıflandırma algoritması hesaplamZa açısından karmaşık olabilir.

KOD

```
import pandas as pd
import re
from collections import defaultdict
from math import log

class NaiveBayesClassifier:
    def __init__(self):
        self.class_probs = defaultdict(float)
        self.word_probs = defaultdict(lambda: defaultdict(float))
        self.vocabulary_size = 0

    def preprocess_text(self, text):
        # Metni küçük harfe çevirme ve gereksiz karakterleri temizleme
        text = text.lower()
        text = re.sub(r'[^a-z\s]', '', text)
        return text.split()

    def train(self, documents, labels):
        total_docs = len(documents)
        spam_docs = sum(1 for label in labels if label == 'spam')
        ham_docs = total_docs - spam_docs

        # Sınıf olasılıklarını hesaplama
        self.class_probs['spam'] = (spam_docs + 1) / (total_docs + 2)  # Laplace düzeltmesi
        self.class_probs['ham'] = (ham_docs + 1) / (total_docs + 2)  # Laplace düzeltmesi

        # Kelime olasılıklarını hesaplama
        word_counts = defaultdict(lambda: {'spam': 0, 'ham': 0})

        for doc, label in zip(documents, labels):
            words = self.preprocess_text(doc)
```

```
            for word in set(words):
                word_counts[word][label] += 1

        self.vocabulary_size = len(word_counts)  # Kelime dağarcığı boyutunu güncelle

        for word, counts in word_counts.items():
            self.word_probs[word]['spam'] = (counts['spam'] + 1) / (spam_docs +
self.vocabulary_size)  # Laplace düzeltmesi
            self.word_probs[word]['ham'] = (counts['ham'] + 1) / (ham_docs +
self.vocabulary_size)  # Laplace düzeltmesi

    def predict(self, document):
        words = self.preprocess_text(document)

        spam_prob = log(self.class_probs['spam'])
        ham_prob = log(self.class_probs['ham'])

        for word in words:
            spam_prob += log(self.word_probs[word]['spam'] + 1e-10)
            ham_prob += log(self.word_probs[word]['ham'] + 1e-10)

        return 'spam' if spam_prob > ham_prob else 'ham'


# Excel dosyasından verileri okuma
df = pd.read_excel('./veri.xlsx')

# Verileri eğitim ve etiket listelerine ayırma
train_documents = df['Document'].tolist()
train_labels = df['Label'].tolist()

# Naive Bayes sınıflandırıcıyı eğitme
classifier = NaiveBayesClassifier()
classifier.train(train_documents, train_labels)

# Test verisi
test_document = input('Mesajı giriniz : ')

# Tahmin yapma
prediction = classifier.predict(test_document)

# Sonucu görüntüleme
print(f"Test dokümanı: '{test_document}'")
print(f"Tahmin: {prediction}")
```

AÇIKLAMALAR

```
import pandas as pd
import re
from collections import defaultdict
from math import log
```

Bu kod parçası, Python programlama dilinde yaygın olarak kullanılan bazı kütüphaneleri içermektedir.

1. **pandas:**

   - `import pandas as pd` satırı, pandas kütüphanesini `pd` takma adıyla içeri aktarır. Pandas, veri analizi ve manipülasyonu için güçlü bir kütüphanedir. Veri çerçeveleri (DataFrame) kullanarak veri analizi işlemlerini kolaylaştırır. Burda kullanım amacı ise Excell dosyasından verileri çekmektir.

2. **re:**

   - `import re` satırı, Python'un düzenli ifadeleri (regular expressions) işlemek için sağladığı `re` modülünü içeri aktarır. Düzenli ifadeler, metin içinde desen aramak ve eşleştirmek için kullanılır.

3. **defaultdict:**

   - `from collections import defaultdict` satırı, `collections` modülünden `defaultdict` sınıfını içeri aktarır. `defaultdict`, bir sözlük (dictionary) türüdür ve belirli bir anahtarın değeri yoksa varsayılan bir değerle otomatik olarak oluşturulmasını sağlar.

4. **math:**

   - `from math import log` satırı, `math` modülünden `log` fonksiyonunu içeri aktarır. Bu fonksiyon, bir sayının belirli bir tabandaki logaritmasını hesaplamak için kullanılır.

```
def __init__(self):
        self.class_probs = defaultdict(float)
        self.word_probs = defaultdict(lambda: defaultdict(float))
        self.vocabulary_size = 0
```

1. `self.class_probs`: Bu, sınıfların ("spam" ve "ham") olasılıklarını tutan bir `defaultdict(float)` nesnesidir. Naive Bayes sınıflandırıcısı, eğitim sırasında belge sınıflarının olasılıklarını hesaplar ve bu olasılıkları bu sözlükte saklar.

2. `self.word_probs`: Bu, kelimelerin sınıflara göre olasılıklarını tutan bir `defaultdict(lambda: defaultdict(float))` nesnesidir. Eğitim sırasında, her kelimenin "spam" veya "ham" sınıfına ait olma olasılıkları hesaplanır ve bu olasılıklar bu sözlükte saklanır.

3. `self.vocabulary_size`: Bu, eğitim sırasında belirlenen kelime dağarcığının boyutunu temsil eder. Kelime dağarcığı, eğitim verilerindeki farklı kelimelerin toplam sayısını belirtir.

`__init__` fonksiyonu, bu değerleri başlangıçta sıfıra ayarlar ve sınıflandırıcının eğitildikten sonra bu değerleri güncelleyeceği veri yapılarını oluşturur. Bu, sınıflandırıcının tahminler yapmak üzere hazır hale getirilmesine yardımcı olur.

```
def preprocess_text(self, text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    return text.split()
```

Bu `preprocess_text` fonksiyonu, verilen bir metni ön işleme adımlarından geçirerek, metni temizleyip analiz için uygun hale getirir.

1. `text.lower()`: Bu adım, metindeki tüm harfleri küçük harfe çevirir. Bu, büyük ve küçük harf farklılıklarını ortadan kaldırarak kelime analizini basitleştirir.

2. `re.sub(r'[^a-z\s]', '', text)`: Bu adım, regular expression (regex) kullanarak metinden yalnızca küçük harfleri (a-z aralığındaki harfler) ve boşlukları (whitespace) tutar. `[^a-z\s]` ifadesi, a-z aralığındaki harflerin dışındaki her karakteri ifade eder. Bu sayede noktalama işaretleri, rakamlar ve diğer özel karakterler temizlenir.

3. `return text.split()`: Bu adım, temizlenmiş metni boşluklara göre böler ve bir liste halinde döndürür. Bu sayede metin, ayrı ayrı kelimelere ayrılmış olur. Kelimelerin listesi daha sonra Naive Bayes sınıflandırıcısının eğitim ve tahmin aşamalarında kullanılır.

Sonuç olarak, `preprocess_text` fonksiyonu, bir metni küçük harfe çevirir, gereksiz karakterleri temizler ve kelimelere böler. Bu ön işleme adımları, metin verilerini daha anlamlı ve kullanışlı hale getirmeye yardımcı olur.

```
def train(self, documents, labels):
    total_docs = len(documents)
    spam_docs = sum(1 for label in labels if label == 'spam')
    ham_docs = total_docs - spam_docs

    self.class_probs['spam'] = (spam_docs + 1) / (total_docs + 2)
    self.class_probs['ham'] = (ham_docs + 1) / (total_docs + 2)

    word_counts = defaultdict(lambda: {'spam': 0, 'ham': 0})

    for doc, label in zip(documents, labels):
        words = self.preprocess_text(doc)
        for word in set(words):
```

```
                word_counts[word][label] += 1

        self.vocabulary_size = len(word_counts)

        for word, counts in word_counts.items():
            self.word_probs[word]['spam'] = (counts['spam'] + 1) / (spam_docs +
self.vocabulary_size)
            self.word_probs[word]['ham'] = (counts['ham'] + 1) / (ham_docs +
self.vocabulary_size)
```

Bu `train` fonksiyonu, Naive Bayes sınıflandırıcısını eğitmek için kullanılır. Eğitim süreci sırasında, belge koleksiyonu (documents) ve etiketler (labels) üzerinde istatistiksel hesaplamalar yaparak sınıf olasılıklarını ve kelime olasılıklarını belirler.

1. `total_docs = len(documents)`: Eğitim veri setindeki toplam belge sayısını alır.

2. `spam_docs = sum(1 for label in labels if label == 'spam')`: "spam" etiketine sahip belge sayısını hesaplar.

3. `ham_docs = total_docs - spam_docs`: "ham" etiketine sahip belge sayısını hesaplar.

4. `self.class_probs['spam']` ve `self.class_probs['ham']`: Sınıf olasılıklarını ("spam" ve "ham" sınıfları için) hesaplar. Laplace düzeltmesi (smoothing) uygulanmıştır (`+1` ve `+2` terimleri) toplam belge sayısı ve her sınıfa ait belge sayısını kullanarak.

5. `word_counts`: Kelime sayılarını tutan bir `defaultdict` nesnesi oluşturur. Her kelime için, "spam" ve "ham" sınıflarındaki belge sayısını takip eder.

6. `for doc, label in zip(documents, labels):`: Eğitim veri setindeki her belge ve etiket çifti için döngü başlatır.

7. `words = self.preprocess_text(doc)`: Belgedeki kelimeleri ön işleme geçirir (küçük harfe çevirme, gereksiz karakterleri temizleme, kelimeleri ayırma).

8. `for word in set(words):`: Her belgedeki benzersiz kelimeler için döngü başlatır.

9. `word_counts[word][label] += 1`: Kelimenin, ilgili etiket ("spam" veya "ham") altındaki belgelerde geçme sayısını artırır.

10. `self.vocabulary_size = len(word_counts)`: Kelime dağarcığının boyutunu günceller.

11. Kelime olasılıklarını hesaplar. Her kelimenin "spam" ve "ham" sınıflarına ait olma olasılıklarını Laplace düzeltmesi uygulayarak belirler. Bu olasılıklar, `self.word_probs` içinde saklanır.

Bu işlemler sonucunda, Naive Bayes sınıflandırıcısı eğitilmiş olur ve belirli bir belgenin "spam" veya "ham" sınıfına ait olma olasılığını tahmin etmek için kullanılmaya hazır hale gelir.

```
def predict(self, document):
    words = self.preprocess_text(document)

    spam_prob = log(self.class_probs['spam'])
    ham_prob = log(self.class_probs['ham'])

    for word in words:
        spam_prob += log(self.word_probs[word]['spam'] + 1e-10)
        ham_prob += log(self.word_probs[word]['ham'] + 1e-10)

    return 'spam' if spam_prob > ham_prob else 'ham'
```

Bu `predict` fonksiyonu, Naive Bayes sınıflandırıcısının eğitildikten sonra belirli bir belgenin ("spam" veya "ham" olarak etiketlenmiş) hangi sınıfa ait olduğunu tahmin etmek için kullanılır.

1. `words = self.preprocess_text(document)`: Verilen belgedeki kelimeleri ön işleme geçirir. `preprocess_text` fonksiyonu, metni küçük harfe çevirme, gereksiz karakterleri temizleme ve kelimeleri ayırma işlemlerini gerçekleştirir.

2. `spam_prob = log(self.class_probs['spam'])` ve `ham_prob = log(self.class_probs['ham'])`: Belgenin "spam" ve "ham" sınıflarına ait olma olasılıklarını başlangıçta, eğitim sürecinde hesaplanan sınıf olasılıkları üzerinden belirler. Olasılıkların logaritmik alınması, sayılar arasındaki büyük farkları dengeleyerek hesaplamaların daha kararlı olmasını sağlar.

3. `for word in words: ...`: Belgedeki her kelime için döngü başlatır.

4. `spam_prob += log(self.word_probs[word]['spam'] + 1e-10)`: Her kelimenin "spam" sınıfına ait olma olasılığını, eğitim sürecinde hesaplanan kelime olasılıkları üzerinden günceller. Logaritmik alınan olasılıkların toplanması, çarpma işleminin toplama işlemine dönüşmesini sağlar.

5. `ham_prob += log(self.word_probs[word]['ham'] + 1e-10)`: Her kelimenin "ham" sınıfına ait olma olasılığını, eğitim sürecinde hesaplanan kelime olasılıkları üzerinden günceller.

6. `return 'spam' if spam_prob > ham_prob else 'ham'`: Son olarak, belgeye ait "spam" ve "ham" sınıflarına ait olma olasılıkları karşılaştırılır. Eğer "spam" olasılığı, "ham" olasılığından büyükse, belge "spam" olarak tahmin edilir; aksi halde "ham" olarak tahmin edilir.

Bu fonksiyon, eğitilen Naive Bayes sınıflandırıcısını kullanarak verilen bir belgenin hangi sınıfa ait olduğunu tahmin eder.

```
df = pd.read_excel('./veri.xlsx')

train_documents = df['Document'].tolist()
train_labels = df['Label'].tolist()

classifier = NaiveBayesClassifier()
classifier.train(train_documents, train_labels)

test_document = input('Mesajı giriniz : ')

prediction = classifier.predict(test_document)

print(f"Test dokümanı: '{test_document}'")
print(f"Tahmin: {prediction}")
```

1. `df = pd.read_excel('./veri.xlsx')`: Pandas kütüphanesini kullanarak 'veri.xlsx' isimli bir Excel dosyasını okur. Bu dosya içinde belge (document) ve etiket (label) bilgileri bulunmalıdır.

2. `train_documents = df['Document'].tolist()`: Eğitim veri setinden belge bilgilerini alır ve bir liste olarak `train_documents` değişkenine atar.

3. `train_labels = df['Label'].tolist()`: Eğitim veri setinden etiket bilgilerini alır ve bir liste olarak `train_labels` değişkenine atar.

4. `classifier = NaiveBayesClassifier()`: `NaiveBayesClassifier` sınıfından bir örnek oluşturur. Bu örnek, Naive Bayes sınıflandırıcısını temsil eder.

5. `classifier.train(train_documents, train_labels)`: Oluşturulan sınıflandırıcıyı eğitir. Eğitim veri seti (`train_documents` ve `train_labels`) kullanılarak sınıf olasılıkları ve kelime olasılıkları hesaplanır.

6. `test_document = input('Mesajı giriniz : ')`: Kullanıcıdan bir test belgesi girmesini isteyen bir giriş isteği yapar.

7. `prediction = classifier.predict(test_document)`: Girilen test belgesi üzerinde sınıflandırıcıyı kullanarak bir tahmin yapar.

8. `print(f"Test dokümanı: '{test_document}'")`: Girilen test belgesini ekrana yazdırır.

9. `print(f"Tahmin: {prediction}")`: Yapılan tahmini ekrana yazdırır. Bu, test belgesinin "spam" veya "ham" sınıfına ait olup olmadığını gösterir.

Bu kod parçacığı, Naive Bayes sınıflandırıcısının eğitim ve test aşamalarını içerir. Eğitilen sınıflandırıcı, kullanıcıdan alınan bir test belgesi üzerinde tahmin yapar ve sonucu ekrana basar.