Multinominal Naive Bayes ile Türkçe Cümlelerde Pozitiflik Sınıflaması

Kullanılan Yöntem

Projemiz bir text classification projesi olduğu için hem basit hem de efektif bir çözüm olarak Multinominal Naive Bayes kullanmayı ve baştan implente etmeyi seçtik.

Multinominal Naive Bayes ile ilgili Formüller

Bayes Teoremi

- A ve B verilen durumlar
- P(A|B), B doğru iken A'nın gerçekleşme ihtimali (Posterior Probability)
- P(B|A), A doğru iken B'nın gerçekleşme ihtimali (Likelihood)
- P(A) (Class Prior Probability) ile P(B) (Predictor Prior Probability) ise A ile B olaylarının gerçekleşme ihtimalleridir.

Bayes Teoreminin Text Classification İçin Kullanılan Hali

- nc, durum gerçekleşirkenki girdi olarak verilen kelime sayısı.
- n, durum gerçekleşirkenki bütün kelimelerin sayısı.
- vocabulary, eşsiz kelime sayısı.

Test ve Training İçin Kullanılan Dataset

Geliştirme yaparken Kaggle'da bulunan Mustafa Keskin'e ait olan Turkish Movie Sentiment Analysis Dataset'i kullandık.

Kod Açıklamaları

```
import pandas as pd

dataset = pd.read_csv('./dataset.csv',
                      header=None,
                      names=["Comment", "Movie" ,"Bool"])

dataset = dataset.tail(-1)
```

```
dataset = dataset.drop("Movie", 1)

dataset = dataset.sample(frac=.01, random_state=1).reset_index(drop=True)
```

Pandas kütüphanesini data frame'leri oluşturmak için kullanıyoruz. Data frame'ler elimizdeki csv dosyasını bir çeşit iki boyutlu array'e dönüştürmemizi sağlıyor ve her feature'a bir isim atamamızı sağlıyor. Elimizdeki datayı tek seferde hafızaya yüklediğimiz için sadece yüzde birlik bir kısmını rastgele yükleyerek çalıştırdık projemizi.

```
def map_points(x):
    x = float(x.replace(',', '.'))
    if x < 3.5:
        return False
    else:
        return True
def clean_values(x):
    x = x.replace('\n', ' ')
    x = x.replace('\W', ' ')
    x = x.lower()
    return x

dataset['Bool'] = dataset['Bool'].apply(map_points)
dataset['Comment'] = dataset['Comment'].apply(clean_values)
dataset['Comment'] = dataset['Comment'].str.split()
```

map_points ve clean_values fonksiyonları adlarından da anlaşılacağı üzere sıra ile kullandığımız dataset'teki continuous değerleri discrete değerlere map etmeye ve "Comment" column'u üzerindeki boşlukları silmeye yarıyor. Bundan sonraki kısımda da bu fonksiyonları gerekli kolonlara uygulayıp "Comment" column'undaki string değerine sahip olan kısmı string listine'e değiştirmeye yarıyor. Bunu frekans tablosu oluştururken kullanacağız.

```
vocabulary = []
for comment in dataset['Comment']:
    for word in comment:
        vocabulary.append(word)

vocabulary = list(set(vocabulary))
```

Sözlüğümüzü oluşturup "Comment" column'u listelerdeki her kelimeyi teker teker ekliyoruz. En sonunda da bu listeyi bir kümeye çevirip sadece unique değerlerin kaldığına emin olarak listeye geri çeviriyoruz. Bu noktada dataların temizlenmesi neredeyse bitiyor.

```
word_counts_per_comment = {unique_word: [0] * len(dataset['Comment']) for unique_w

for index, comment in enumerate(dataset['Comment']):
    for word in comment:
        word_counts_per_comment[word][index] += 1

word_counts = pd.DataFrame(word_counts_per_comment)
```

Frekans tablosu bir DataFrame şeklinde oluşturuluyor.

```
dataset_joined = pd.concat([dataset, word_counts], axis=1)
```

Dataset'imizi ve frekans tablosunu birleştirip tek bir DataFrame haline getiriyoruz.

```
import numpy as np

positive_values = dataset_joined[dataset_joined['Bool'] == np.bool_(True)]
negative_values = dataset_joined[dataset_joined['Bool'] == np.bool_(False)]

#p(durum)
def find_class_prior_prob(positive_values, negative_values):
    positive_percentage = positive_values.shape[0] / len(dataset_joined)
    negative_percentage = negative_values.shape[0] / len(dataset_joined)

    return positive_percentage, negative_percentage

positive_percentage, negative_percentage = find_class_prior_prob(positive_values,

#n değeri
def n(positive_values, negative_values):
    n_words_per_positive_message = positive_values['Comment'].apply(len)
    n_positive = n_words_per_positive_message.sum()
```

```
    n_words_per_negative_message = negative_values['Comment'].apply(len)
    n_negative = n_words_per_negative_message.sum()

    return n_positive, n_negative

n_positive, n_negative = nc(positive_values, negative_values)

#vocabulary uzunluğu
n_vocabulary = len(vocabulary)
```

Class Prior Probability, n ve vocabulary değerlerini hesaplıyoruz. Buradaki hesaplamalar sırasında Numpy bool değerlerini kullanmamızın sebebi Python bool değerleri ile karşılaştırmada sıkıntı çıkması idi.

```
alpha = 1 # laplace smoothing

parameters_positive = {unique_word:0 for unique_word in vocabulary}
parameters_negative = {unique_word:0 for unique_word in vocabulary}

# datasetteki her kelime için p(kelime|durum hesaplanması)

for word in vocabulary:
    #nc
    n_word_given_positive = positive_values[word].sum()
    #m estimate
    p_word_given_positive = (n_word_given_positive + alpha) / (n_positive + alpha*
    parameters_positive[word] = p_word_given_positive
    #nc
    n_word_given_negative = negative_values[word].sum()
    #m estimate
    p_word_given_negative = (n_word_given_negative + alpha) / (n_negative + alpha*
    parameters_negative[word] = p_word_given_negative
```

Raporun matematiksel kısmındaki formülü kullanarak (laplace smoothing kullanıyoruz) gerekli değerleri hesaplıyoruz.

```
import re
```

```
import nltk
from stop_words import stop_words


WPT = nltk.WordPunctTokenizer()
# Yukarıdaki rowları normalize etme işlemine de bu method uygulanabilir
# bir ara implente et.
def norm_doc(single_doc):
    # TR: Dokümandan belirlenen özel karakterleri ve sayıları at
    # EN: Remove special characters and numbers
    single_doc = re.sub(" \d+", " ", single_doc)
    pattern = r"[{}]".format(",.;")
    single_doc = re.sub(pattern, "", single_doc)
    # TR: Dokümanı küçük harflere çevir
    # EN: Convert document to lowercase
    single_doc = single_doc.lower()
    single_doc = single_doc.strip()
    # TR: Dokümanı token'larına ayır
    # EN: Tokenize documents
    tokens = WPT.tokenize(single_doc)
    # TR: Stop-word listesindeki kelimeler hariç al
    # EN: Filter out the stop-words
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # TR: Dokümanı tekrar oluştur
    # EN: Reconstruct the document
    single_doc = ' '.join(filtered_tokens)
    return single_doc

def norm_values(perc_positive, perc_negative):
    sum = perc_positive + perc_negative
    norm_positive = perc_positive / sum
    norm_negative = perc_negative / sum

    return norm_positive, norm_negative

def classify(message):
    message = re.sub('\W', ' ', message)
    message = norm_doc(message).split()

    p_positive_given_message = positive_percentage
    p_negative_given_message = negative_percentage
```

```
    for word in message:

        if word in parameters_positive:
            p_positive_given_message *= parameters_positive[word]

        if word in parameters_negative:
            p_negative_given_message *= parameters_negative[word]

    confidence = norm_values(p_positive_given_message, p_negative_given_message)

    if p_negative_given_message < p_positive_given_message:
        return True, confidence[0], confidence[1]
    elif p_negative_given_message > p_positive_given_message:
        return False, confidence[0], confidence[1]
```

- norm_doc fonksiyonu, test olarak verilen string'i temizleyip tokenize etmeye;
- norm_values fonksiyonu, çıktı olarak verilen son yüzdeleri 0 ile 1 arasına map etmeye yarıyor ki confidence değerlerini hesaplayabilelim.
- classify fonksiyonu, bütün değerleri alarak son hesaplamayı yapan fonksiyon olarak kodumuzda yer alıyor.