NameSurname=SelimCanYazar

StudentID= 152120191023

Pattern Recognition Hw6 Rapor

Adım 1: MI Hesaplama ve En İyi İki Kelimeyi Seçme

```python
import numpy as np
from collections import Counter
from math import log2

# Eğitim verisi
documents = [
    ("free free free buy discount combo pleasure", 'S'),
    ("free free free discount pleasure smile smile smile", 'S'),
    ("cat mouse", 'N'),
    ("cat cat dog dog dog dog", 'N'),
    ("mouse", 'N')
]

# Kelime sayısı
word_counts = Counter(word for doc, _ in documents for word in doc.split())

# Kelime sayısı ve sınıf dağılımı
class_word_counts = {'S': Counter(), 'N': Counter()}
for doc, label in documents:
    words = doc.split()
    for word in words:
        class_word_counts[label][word] += 1
```

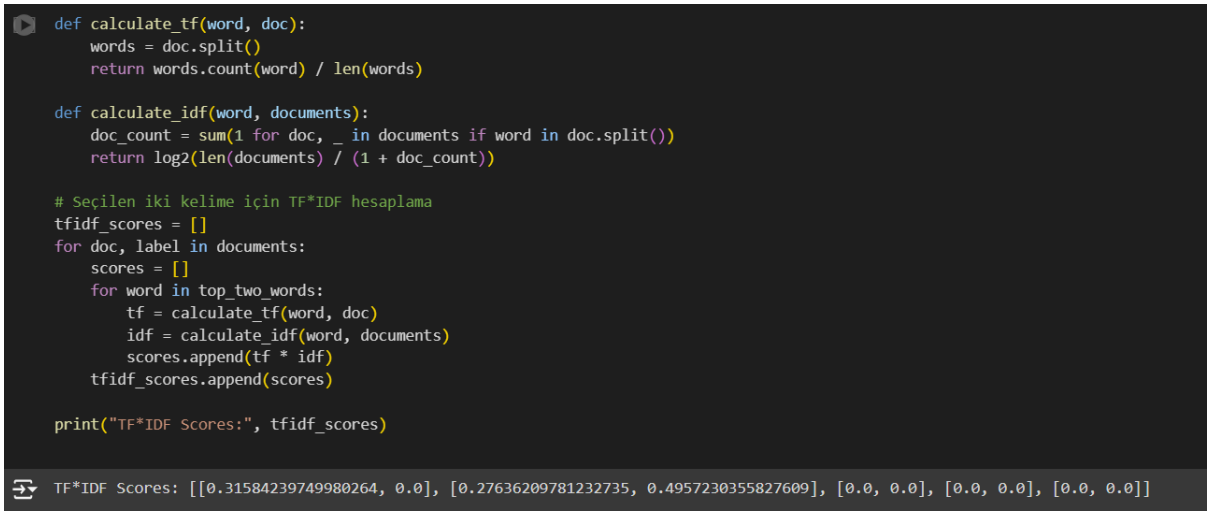
```python
# MI hesaplama
def calculate_mi(word, class_word_counts, word_counts, total_docs, class_docs):
    mi = 0.0
    for label in class_word_counts.keys():
        p_wc = (class_word_counts[label][word] + 1) / (total_docs + 1)
        p_w = (word_counts[word] + 1) / (total_docs + 1)
        p_c = class_docs[label] / total_docs
        mi += p_wc * log2(p_wc / (p_w * p_c))
    return mi

total_docs = len(documents)
class_docs = {label: sum(1 for _, l in documents if l == label) for label in class_word_counts.keys()}
mi_scores = {word: calculate_mi(word, class_word_counts, word_counts, total_docs, class_docs) for word in word_counts}

# En yüksek iki MI skoru
top_two_words = sorted(mi_scores, key=mi_scores.get, reverse=True)[:2]
print("Top two words by MI:", top_two_words)
```

```
Top two words by MI: ['free', 'smile']
```

Adım 1 de eğitim verileri üzerinden Mutual Information yani MI hesaplaması yapılmıştır. İlk olarak her belgedeki kelimelerin sayısı hesaplanmış ve ardından spam veya normal olmak üzere her sınıf için kelime dağılımı belirlenmiştir. Daha sonra calculate_mi fonksiyonu

kullanılarak her kelimenin MI skoru hesaplanmıştır. Bu fonksiyon sayesinde bir kelimenin belirli bir sınıfta bulunma olasılığı ile tüm sınıflarda bulunma olasılığını karşılaştırarak bağımlılıklarını ölçer. MI skorları hesaplandıktan sonra en yüksek MI skoruna sahip iki kelime seçilmiştir.

Adım 2: TF*IDF Hesaplama



```
def calculate_tf(word, doc):
    words = doc.split()
    return words.count(word) / len(words)

def calculate_idf(word, documents):
    doc_count = sum(1 for doc, _ in documents if word in doc.split())
    return log2(len(documents) / (1 + doc_count))

# Seçilen iki kelime için TF*IDF hesaplama
tfidf_scores = []
for doc, label in documents:
    scores = []
    for word in top_two_words:
        tf = calculate_tf(word, doc)
        idf = calculate_idf(word, documents)
        scores.append(tf * idf)
    tfidf_scores.append(scores)

print("TF*IDF Scores:", tfidf_scores)
```

```
TF*IDF Scores: [[0.31584239749980264, 0.0], [0.27636209781232735, 0.4957230355827609], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0]]
```

Adım 2: TF*IDF Hesaplama kısmında seçilen iki kelime için TF*IDF skorlarını hesaplıyoruz.Önce bir kelimenin bir belgedeki frekansını buluyoruz. Bu işlem bir kelimenin o belgedeki toplam kelime sayısına oranıdır. Örneğin, bir belgede "free" kelimesi üç kez geçiyorsa ve belgede toplamda on kelime varsa, "free" kelimesinin frekansı 3/10 yani 0.3 olur.

Frekansı bulduktan sonra bir kelimenin kaç farklı belgede geçtiğini hesaplıyoruz. Buna document frequency denir. Bu değeri kullanarak IDF hesaplıyoruz. IDF toplam belge sayısının bu kelimenin geçtiği belge sayısına bölünmesinin logaritmasıdır. IDF bir kelimenin nadirliğini ölçer; nadir olan kelimeler daha yüksek IDF değerine sahip olur. Örneğin, "free" kelimesi beş belgenin üçünde geçiyorsa IDF değeri log2(5/3) olur.

Son olarak TF ve IDF değerlerini çarparak her kelimenin TF*IDF skorunu elde ediyoruz. Bu skor, bir kelimenin o belgede ne kadar önemli olduğunu gösterir. Eğitim verilerindeki her belge için bu skorları hesaplıyoruz ve bir listeye ekliyoruz. Bu liste her belgenin bu iki kelime açısından ne kadar önemli olduğunu belirtir.

TF*IDF Scores: [[0.31584239749980264, 0.0], [0.27636209781232735, 0.4957230355827609], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0]]

Bu sonuçlar her belgenin seçilen iki kelimeye göre TFIDF skorlarını gösterir. Her alt liste, bir belgenin bu iki kelime için TFIDF skorlarını içerir.

İlk belge (d1) için free kelimesinin TF*IDF skoru yaklaşık 0.316 ve smile kelimesi geçmediği için skoru 0. Bu free kelimesinin bu belgede önemli olduğunun ancak smile kelimesinin bu belgede bulunmadığını gösterir. İkinci belge (d2) için free kelimesinin TF*IDF skoru yaklaşık 0.276 ve smile kelimesinin skoru yaklaşık 0.496. Bu, her iki kelimenin de bu belgede önemli olduğunu gösterir. Üçüncü, dördüncü ve beşinci belgelerde (d3, d4, d5) bu kelimeler geçmediği için TF*IDF skorları 0'dır.Bu şekilde, her belgenin belirlenen iki kelimeye göre önemini belirlemiş oluyoruz. Bu skorlar, belgelerin sınıflandırılması için kullanılacak.

Adım 3: Belgeleri Temsil Etme

```python
import pandas as pd

# Belgeleri TF*IDF değerleri ile temsil etme
df = pd.DataFrame(tfidf_scores, columns=top_two_words)
print("TF*IDF Representation:\n", df)
```

```
TF*IDF Representation:
       free     smile
0  0.315842  0.000000
1  0.276362  0.495723
2  0.000000  0.000000
3  0.000000  0.000000
4  0.000000  0.000000
```

Adım 3: Belgeleri Temsil Etme kısmında Hesapladığımız TF*IDF değerlerini kullanarak belgelerimizi iki kelime yani özellik ile temsil ederiz. Pandas DataFrame kullanarak bu değerleri bir tablo halinde düzenleriz. Bu tablo sayesinde her belgenin seçilen iki kelimeye göre nasıl temsil edildiğini gösteririr ve bu şekilde verilerimizi daha net bir şekilde görebiliriz.

Adım 4: Test Verileri İçin TF*IDF Hesaplama

```
test_documents = ["dog cat mouse cat", "Free free smile"]

# Test verileri için TF*IDF hesaplama
test_tfidf_scores = []
for doc in test_documents:
    scores = []
    for word in top_two_words:
        tf = calculate_tf(word, doc)
        idf = calculate_idf(word, documents)
        scores.append(tf * idf)
    test_tfidf_scores.append(scores)

print("Test TF*IDF Scores:", test_tfidf_scores)
```

```
Test TF*IDF Scores: [[0.0, 0.0], [0.24565519805540206, 0.440642698295787 45]]
```

Adım 4: Test Verileri İçin TF*IDF Hesaplama test belgeleri için TFIDF (Term Frequency - Inverse Document Frequency) skorlarını hesaplıyoruz. Test belgeleri bir liste içinde tanımlanır. Burada iki test belgesi vardır. ilki "dog cat mouse cat" ve ikincisi "Free free smile" test belgeleridir. Test belgeleri için TFIDF skorlarını saklamak üzere boş bir liste oluşturulur. Her bir test belgesini işlemek için bir döngü başlatılır ve her belge için TFIDF skorlarını saklamak üzere boş bir liste oluşturulur. Daha önce seçilen iki önemli kelime (örneğin, "free" ve "smile") üzerinde bir döngü başlatılır. Bu döngüde içerisinde mevcut kelimenin belgede ne kadar sıklıkla geçtiğini hesaplayan TF (Term Frequency) fonksiyonu kullanılır ve IDF (Inverse Document Frequency) fonksiyonu ile kelimenin nadirliğini belirler. Bu iki değerin çarpımı ile TFIDF skoru elde edilir ve bu skorlar listeye eklenir. Her test belgesi için bu işlemler tamamlandıktan sonra, sonuçlar ana listeye eklenir. Bu sayede test belgelerinin TF*IDF skorları hesaplanmış olur.

Test TF*IDF Scores: [[0.0, 0.0], [0.24565519805540206, 0.44064269829578745]]

İlk test belgesi ("dog cat mouse cat") için her iki kelimenin (örneğin, "free" ve "smile") TF*IDF skorları sıfırdır. Bu kelimelerin bu belgede

bulunmadığını gösterir. Dolayısıyla bu belgenin "free" ve "smile" kelimeleri açısından bir önemi yoktur.

İkinci test belgesi ("Free free smile") için "free" kelimesinin TFIDF skoru yaklaşık 0.246 ve "smile" kelimesinin TFIDF skoru yaklaşık 0.441'dir. Bu, bu iki kelimenin bu belgede önemli olduğunu gösterir. "Free" kelimesi belgede iki kez geçtiği için frekansı yüksektir ve "smile" kelimesi de bu belgede birkaç kez geçtiği için önemi yüksektir. Bu skorlar bu kelimelerin belgede önemli bir yere sahip olduğunu belirtir.

Adım 5: KNN Algoritması ile Sınıf Tahmini

Adım 5: KNN Algoritması ile Sınıf Tahmini

```
[ ]  from sklearn.neighbors import KNeighborsClassifier

     # Eğitim verisi ve sınıf etiketleri
     X_train = np.array(tfidf_scores)
     y_train = np.array([label for _, label in documents])

     # KNN model oluşturma ve eğitme
     knn = KNeighborsClassifier(n_neighbors=3)
     knn.fit(X_train, y_train)

     # Test verisi ve tahmin sonuçları
     X_test = np.array(test_tfidf_scores)
     predictions = knn.predict(X_test)

     # Test verileri ve tahmin edilen sınıflar
     test_documents_with_predictions = {
         'd6': predictions[0],
         'd7': predictions[1]
     }

     for doc, prediction in test_documents_with_predictions.items():
         print(f"Prediction for {doc}: {prediction}")
```

```
Prediction for d6: N
Prediction for d7: S
```

Adım 5: KNN Algoritması ile Sınıf Tahmini kısmında KNN algoritması kullanılarak test verilerinin sınıflarını tahmin ediyoruz. İlk aşamada eğitim verisi ve sınıf etiketleri numpy dizilerine dönüştürülür. TFIDF skorları X_train olarak, belgelerin sınıf etiketleri ise y_train olarak adlandırılır. KNN modelini oluşturmak için scikit-learn kütüphanesinden KNeighborsClassifier sınıfını kullanıyoruz. Modeli üç komşu kullanacak şekilde n_neighbors=3 olarak ayarlayıp, eğitim verileri ile eğitiyoruz. Eğitim tamamlandıktan sonra test verilerinin TFIDF skorları X_test olarak adlandırılır ve modelimiz bu veriler üzerinden sınıf tahmini yapar. Tahmin edilen sınıflar predictions dizisinde saklanır. Son olarak test belgeleri ile tahmin edilen sınıflar eşleştirilir ve her test belgesi için tahmin edilen sınıf ekrana yazdırılarak her test belgesinin hangi sınıfa ait olduğunu gösterir.

Prediction for d6: N

Prediction for d7: S

Örneğin yukarıda gözüktüğü üzere ödevimizin sonucu olarak d6 belgesi N sınıfına (yani normal) ve d7 belgesi ise S sınıfına (yani spam) olarak tahmin edilmiştir.