FEN BİLİMLERİ ENSTİTÜSÜ

METİN MADENCİLİĞİ KULLANARAK İNGİLİZCE DOKÜMAN SINIFLAMA

Ahmet Görkem ÖZDOĞAN

Danışman
Dr. Öğr. Üyesi Metin TURAN

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
İSTANBUL – 2019

KABUL VE ONAY SAYFASI

Ahmet Görkem ÖZDOĞAN tarafından hazırlanan **"Metin Madenciliği Kullanarak İngilizce Doküman Sınıflama"** adlı tez çalışması 24/09/2019 tarihinde aşağıdaki jüri üyeleri önünde başarı ile savunularak, İstanbul Ticaret Üniversitesi Fen Bilimleri Enstitüsü **Bilgisayar Mühendisliği Anabilim Dalı**'nda **Yüksek Lisans Tezi** olarak kabul edilmiştir.

Danışman	**Dr. Öğr. Üyesi Metin TURAN** İstanbul Ticaret Üniversitesi	
Jüri Üyesi	**Prof. Dr. Selim AKYOKUŞ** İstanbul Medipol Üniversitesi	
Jüri Üyesi	**Doç. Dr. Berk AYVAZ** İstanbul Ticaret Üniversitesi	

Onay Tarihi : 23.10.2019

Prof. Dr. Necip ŞİMŞEK
Enstitü Müdürü

AKADEMİK VE ETİK KURALLARA UYGUNLUK BEYANI

İstanbul Ticaret Üniversitesi, Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- Tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- Görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- Başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- Atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- Kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

Beyan ederim.

Ahmet Görkem ÖZDOĞAN

İÇİNDEKİLER

ÖZET

Yüksek Lisans Tezi

METİN MADENCİLİĞİ KULLANARAK İNGİLİZCE DOKÜMAN SINIFLAMA

Ahmet Görkem ÖZDOĞAN

İstanbul Ticaret Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr.Öğr. Üyesi Metin TURAN

2019, 87 sayfa

Günümüzde metin tabanlı dokümanların sınıflandırılması özellikle kurumsal yazışmaların ve dijital dokümantasyonun çok yapıldığı durumlarda ciddi öneme sahiptir. Bu çalışmada bilinirliği yüksek olan kosinüs benzerliği ve Jaccard benzerliği ile Noktasal karşılıklı Bilgi (PMI) birliktelik ölçütü karşılaştırılarak sonuçlar gözlemlenmiştir. Özellik seçimi için, Helmholtz prensibi ile Gestalt teorisi kullanılmıştır. Bu yöntem metin madenciliğinde, özellik çıkarımı, özetleme gibi alanlarda kullanılmıştır. Çalışma için kullanılan doküman veri seti spor ve eğitim temalarında olup, toplam 14 alt kavram önceden belirlenmiştir. Önceden belirlenmiş kavramlara sahip dokümanlar için Kosinüs, Jaccard ve PMI benzerlik ölçütleri karşılaştırılmıştır. Her bir dokümanın benzerlik katsayılarının ortalamaları baz alınarak yapılan sınıflama ise anlamlı kelimelerin yüzdelik değerlerine göre farklı başarımlar elde edilmiştir. Bu bakımdan PMI benzerlik ölçütü anlamlı kelime dağılımlarına adaptif bir yaklaşım sergiler iken Kosinüs benzerlik ölçütünde ve Jaccard benzerliğinde herhangi bir iyileşme gözlemlenmemiştir. Çalışmanın sonraki kısmında, PMI benzerlik ölçütünü K-Means modeli üzerinde uygulayarak öbekleme sonuçları gözlemlenmiştir. Sonuçları iyileştirmek üzere benzerlik gösteren kelimelerin sonuçlara daha belirgin etki yapması amacıyla öbeklenen doküman vektörlerin temsilinde yüzdesel eşikler uygulanmıştır. Bu aralıkta yapılan öbekleme çalışmasında yaklaşık %70'lere varan başarı sağlanmıştır.

Anahtar Kelimeler: PMI benzerliği, Metin Öbekleme, Doğal dil işleme, Cosine benzerliği, Jaccard benzerliği

ABSTRACT

M.Sc. Thesis

TITLE OF THE THESIS

Ahmet Görkem ÖZDOĞAN

İstanbul Commerce University
Graduate School of Applied and Natural Sciences
Department of Computer Engineering

Supervisor: Assist.Prof. Dr. Metin TURAN

2019, 87 pages

Nowadays, the classification of text-based documents is of very import, especially when lots of corporate correspondence and digital documentation are done. Classification of similar texts from piles is a factor increases productivity. In text mining, various approaches to such problems are sought. In this study, we have compared the Cosine similarity and Jaccard similarity with PMI (Pointwise Mutual Information) criterion and the results are observed. The Gestalt theory with the Helmholtz principle was used to identify meaningful words. This method has been used in text mining in areas such as feature extraction, text summarization. The document data set used for the study was in the sports and educational themes and a total of 14 sub-concepts were pre-determined. Cosine Jaccard and PMI similarity criteria were compared for documents with predetermined concepts. On the basis of all of the documents with a similarity rate on average, the likeness of Cosine similarity was 75%, Jaccard similarity was 40% and PMI similarity was 55%. On the other hand, based on the accuracy values, the cosine similarity criterion was 80%, Jaccard similarity was 65%, and PMI similarity was 65%. According to the averages of the similarity coefficients of each document, different performances were obtained according to the percentage of meaningful words. In the point of view, while the PMI similarity criterion exhibits an adaptive approach to meaningful word distributions, no improvement was observed in the cosine similarity criterion and in the Jaccard Similarity. In the next part of the study, clustering results were observed by applying the PMI similarity criterion on K-Means model. In the clustering study for randomly selected classes, it was observed that the 20 randomly selected documents were assigned to different classes in the calculations, considering that the first random classes were assigned different topics. Percentage thresholds were applied to the document vectors of the clustered document vectors in order to have a more obvious effect on words with common similarities in order to improve the results. In the calculations of these threshold values between 25% and 75%, the most successful interval was 60-65%. In this range, the success of the clustering was achieved up to 70%.
Keywords: Pointwise mutual information similarity, Text classification, Natural language processing, Cosine similarity, Jaccard similarity,

TEŞEKKÜR

Bu araştırma için beni yönlendiren, karşılaştığım zorlukları bilgi ve tecrübesi ile aşmamda yardımcı olan değerli danışman hocam Dr. Öğr. Üyesi Metin Turan'a teşekkürlerimi sunarım.

Araştırmanın yürütülmesinde maddi ve manevi yardımlarını gördüğüm başta Ahmet Toprak ve İsmail Levent Utkusavaş'a olmak üzere tüm Türkiye Finans Katılım Bankası personeline teşekkür ederim.

Tez çalışmam boyunca beni yalnız bırakmayan ve desteğini esirgemeyen sevgili eşime ve eğitim hayatım boyunca her zaman desteklerini sunan ve yanımda olan aileme sonsuz sevgi ve saygılarımı sunarım.

A. Görkem Özdoğan

İSTANBUL, 2019

ŞEKİLLER

ÇİZELGELER

Sayfa

SİMGELER VE KISALTMALAR

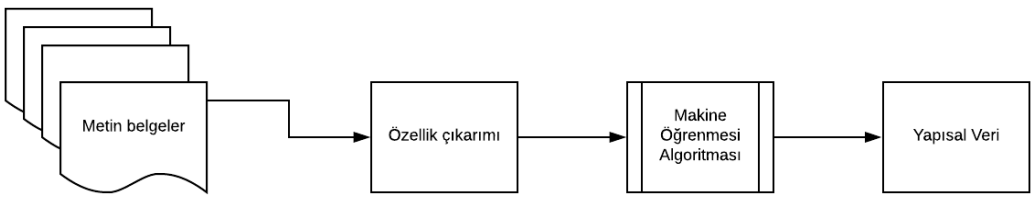
PMI	Pointwise Mutual İnformation
KNN	K Nearest Neighbour
AI	Artifical İntelligence
IR	Information Retrieval
cos	Cosinus
MI	Mutual İnformation
SSE	Sum of Squared Error
D	Document
a	Birliktelik değişkeni
Sim	Similarity
TF	Term Frequency
HTML	Hyper Text Markup Language
XML	Extensible Markup Language
SQL	Structured Query Language
NLTK	Natural Language Toolkit

1. GİRİŞ

Günümüzde gelişen teknoloji ile birlikte üretilen veri miktarı ciddi seviyede artmıştır. Bu durum bilgiye ulaşmayı zorlaştırmakta olup, çeşitli yöntemler ile veriler anlamlandırılıp, mümkünse sınıflara ayrılıp, kullanma ve raporlama için kolaylıklar sağlanmaya çalışılmıştır. Veri madenciliği, veri yığınları içerisinden çeşitli yöntemler kullanılarak kesin olmamak ile birlikte anlamlı sonuçlar çıkarmayı amaçlar (A. Peña-Ayala, 2014). Veri madenciliğinin bir alt dalı olan metin madenciliğinde ise yazılı dokümanlardan anlamlı bilgiler çıkarılması hedeflenir (S. M. Weiss, N. Indurkhya, T. Zhang, F. J. Damerau, 2005). Metin madenciliğini veri madenciliğinden ayıran fark, veri madenciliğinde çıkarımların veri tabanlarında yer alan anlamlandırılmış verileri üzerinde çalışılırken metin madenciliğinden doğal dil metinlerinden bu bilgilerin edinimidir. Yapılan çalışmalara göre, şirket bilgilerinin %80'i metin dokümanlarında tutulmaktadır. **(**A. Akilan, 2015). Bu bakımdan metin madenciliği çalışmalarının gelişen teknoloji ile birlikte önemli bir noktaya geldiği aşikârdır.

Temel olarak metin madenciliği 3 adıma dayanır (M. Sukanya ve S. Biruntha, 2012).

- Metnin ön işlenmesi
- Metin madenciliği tekniğinin uygulanması
- Metin ve sonuçların analizi

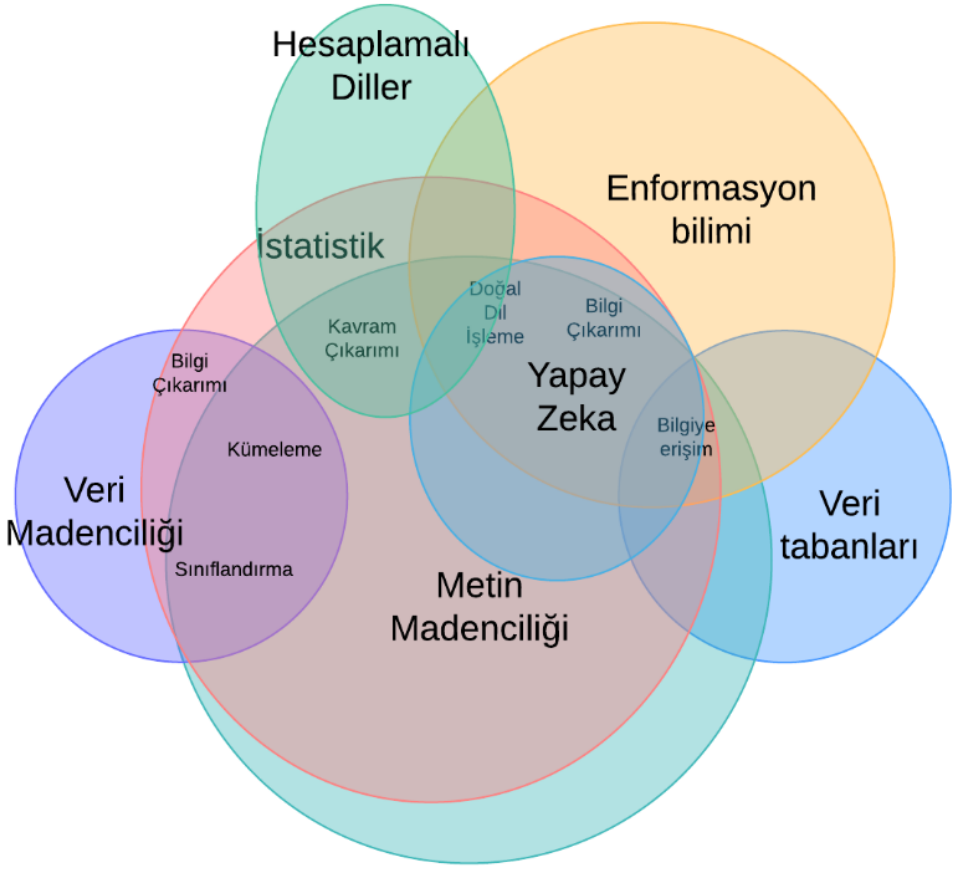


Şekil 1.1. Metin madenciliği çalışması

Bir metin madenciliği çalışması Şekil1.1'de şematik olarak gösterilmiştir. Şekilde görüldüğü üzere bir metin veri tabanından alınan veriler ön işlemeden geçirilerek özellik çıkarımı yapılır. Sonrasında çıkarılan özellikler seçilen makine

öğrenmesi algoritması ile, sınıflandırma, öbekleme ve tahmin gibi aşamaları geçer ve yapısal bir veri elde edilmiş olur.

Metinler, genellikle doğal dil ile yazılan kaynaklardır. Her dilinde kendine has gramer kuralları ve fonetik yapıları mevcuttur. Metin dokümanlarının çeşitliliği ve fazlalığı beraberinde bazı dezavantajlar getirmekte, aranılan nitelikli bilgiye ulaşmak da zorlaşmaktadır. Ulaşabileceğimiz bilginin çok çeşitli ve fazla olması yapılan araştırmaların sonuçlandırılmasına olumlu etki yaparken, bir diğer taraftan zaman kaybına ve aranılan bilgiden ziyade yanlış bir bilgiye ulaşmaya da neden olabilmektedir. Bir metnin sınıflandırılması da eldeki dokümanın önceden belirlenmiş sınıflardan hangisine dahil olduğunun belirlenmesi ile ilgilidir (Hartmann, Huppertz, Schamp, Heitmann, 2019). Metin madenciliği çalışmalarının ilişkili olduğu disiplinler ve yöntemler çok geniş bir kapsamda ele alınabilir. Bu bağlamda farklı disiplinleri de içine alarak çalışmalar yürütülmektedir. Şekil1.2 de bu ilişkiler şematik olarak ifade edilmeye çalışılmıştır.



Şekil 1.2. Metin madenciliğinin diğer disiplinler ile ilişkisi (Metin Madenciliği, 2019)

Şekil 1.2'den görülebileceği üzere metin madenciliği istatistik temelli ve diğer disiplinlerle de birlikte çalışma yapılması gereken kesişimlere sahiptir.

1.1. Problemin Tanımı

İnternetin gelişmesiyle birlikte veri üretimi ve veriye ulaşımın çok kolay hale gelmesi, yapılan araştırmalarda hızlıca bilgiye ulaşmak, kullanmak ve üretkenliği arttırmak gibi olumlu etkiler yapmıştır. Ancak bunun yanında aynı sebeplerden ötürü olumsuz birtakım etkilerde doğurmaktadır. Bu olumsuz etkiler; Araştırma yapılan konuda ihtiyacı karşılayan dokümanlar ile birlikte aslında o araştırma için gerek olmayan dokümanlara da ulaşılmasıdır. Bu da üretkenliğin düşmesine ve zaman kaybı gibi problemlere yol açmaktadır. Bu problemlerin çözümü için farklı yöntemler kullanılarak bilginin, belgenin, dokümanın kısaca çalışma yapılan materyalin sınıflandırılması ile ilgili çalışmalar yapılmaktadır. Bu çalışmanın ana konusu olan doküman öbekleme için de Naive Bayes, En Yakın Komşu, K-Means gibi çeşitli öbekleme yöntemleri örnek verilebilir.

1.2 Çalışma Konusu ve Amacı

Bu çalışmanın ana konusu metin öbekleme yöntemlerinin PMI benzerlik ölçütü ile kullanılarak konuları önceden belirlenmiş doküman kümelerinde öbeklemeyi ne kadar iyi yapabildiğinin tespitidir. Bu teknik bilgimiz dahilinde literatürde öbekleme amacı ile ilk kez uygulanmış bir çalışmadır. Deney amaçlı kullanılan doküman kümeleri bu dokümanlar için anlamlı olan kelimelere göre Kosinüs, Jaccard ve PMI benzerlik ölçütlerine göre karşılaştırılmış sonuçlar gözlemlenmiştir. Amacımız, K-Means veya KNN (K- Nearest Neighbour) öbekleme algoritmaları ile PMI benzerlik ölçütünü kullanarak daha iyi sonuçlar verecek bir öbekleme yöntemi modelinin geliştirilmesidir.

Çalışmanın ikinci bölümünde bu alanda yapılmış çalışmalar hakkında bilgiler verilmiştir, üçüncü bölümde metin madenciliğinde veri işleme adımlarından ve Kosinüs, Jaccard ve PMI benzerliklerinin tespiti için kullanılan yöntemlerden bahsedilmiştir, dördüncü bölümde öbekleme algoritmalarında ve PMI

benzerliğinin birlikte kullanımı için kullanılan yöntemden bahsedilmiştir. Beşinci bölümde geliştirilen yazılımdan bahsedilmiş, son bölümde de sonuçlar açıklanmıştır.

2. LİTERATÜR ÖZETİ

Veri kümeleme, veri madenciliğinin önemli tekniklerinden biridir ve temel olarak benzer veri nesnelerinin farklı gruplara sınıflandırılması işlemidir. Bir veri kümesi belirli bir uzaklık ölçütüne göre kümelere ayrılırken her bir altküme nesnesinin ortak bir özelliğe sahip olması beklenir. Kümeleme işleminde sınıflandırma yapılacak olan kategoriler önceden belli değildir. Bu nedenle kümeleme analizleri gözetimsiz öğrenme yöntemleri olarak adlandırılır (Aggarwal, Gates and Yu, 2004).

Metin sınıflandırması, metin belgelerinin kategorilere veya konulara göre sınıflandırılması, herhangi bir metin işleme sisteminin önemli bir bileşenidir. Doğru belge sınıflandırıcıları oluşturmak için içeriği- belgelerde görünen sözcükleri, belgelerin yapısını- ve dış kaynakları kullanan çok sayıda çalışma vardır (Olsson, 2009). Ek olarak, doküman sınıflandırma performansını arttırmak için dokümanlar arasındaki bağlantı yapısını kullanmaya çalışan yöntemler üzerine artan bir literatür var (Sinoara, Camacho-Collados, Rossi, Navigli ve Rezende, 2019). Metin belgeleri çeşitli yollarla birbirine bağlanabilir. En yaygın bağlantı yapısı alıntı grafiğidir: Örneğin, makaleler belgeleri ve web sayfalarını diğer web sayfalarına bağlar. Bunların hepsi birbirine bağlı bir metin belgeleri koleksiyonu oluşturmak için bir araya getirilebilir.

William ve Haym, 2000 yılında WHIRL isimli uygulama ile internetteki verileri kullanarak bir dizi tümevarımsal sınıflandırma görevinde değerlendirmiştir. WHIRL, daha genel benzerlik temelli muhakeme görevleri için tasarlanmış olmasına rağmen, sınıflandırma işlemlerinde tümevarımsal sınıflandırma sistemleri ile rekabet ettiğini gözlemlemişlerdir. Özellikle, WHIRL genellikle C4.5, RIPPER ve en yakın komşu yöntemlerden daha düşük genelleme hatası elde etmiştir. WHIRL aynı zamanda bazı kıyaslama problemlerinde C4.5'ten 500 kat daha hızlıdır. Ayrıca, WHIRL'nin büyük bir etiketlenmemiş veri havuzundan seçiminde, güvenle doğru şekilde sınıfları seçebileceğini de kanıtlamışlardır (William ve Haym,2000).

2004 yılında ise C. C. Aggarwal, S. C. Gates ve P. S. Yu yapmış oldukları çalışmada, denetimli kümelemeyi kullanarak kategorizasyon sistemleri oluşturmak için yöntemler önermişlerdir. Yahoo! Adlı web sitesinden bir dizi sınıfı kullanarak böyle bir sınıflandırma sistemi oluşturmuşlardır (Aggarwal, Gates ve Yu, 2004).

Metin öbekleme ise doküman setlerinin benzerliklerine göre kümelere ayrıştırılması işlemidir. Hedef olarak bir doküman seti içerisindeki dokümanların benzer bir konuda olmaları beklenir. 1980'lerin ortalarından itibaren metin kümeleme ilgili gelişmeler yaşanmıştır.

Metin öbekleme konusunda birçok çalışma yapılmış ve yapılmaya devam etmektedir. Metin öbekleme alanında en fazla bilinen ve yaygın kullanıma sahip ortalamaya göre merkez kümeleme algoritması K-Means algoritmasıdır (Han ve Kamber, 2006).

M. Işık ve A. Y. Çamurcu, c-means ve k-means algoritmalarını kullanarak belge madenciliği konusunda çalışmışlardır. Web belgeleri kullanılarak yapılan çalışmada her bir web belgesinin içerdikleri kelimeler çok boyutlu vektörler olarak temsil edilmiştir. Yapılan test sonuçlarına göre bulanık c-means algoritması kümeleme işlemlerinde daha az hata oranına sahip olduğu anlaşılmıştır. Oluşan kümelerinin saflıkları ve entropileri k-means ile oluşan kümelerin değerlerinden daha iyi çıktığı tespit edilmiştir. (Işık ve Çamurcu, 2010)

V. Adalı, çalışmasında iyileştirilmiş bir kümeleme yapısı tasarlamıştır. Çalışma kapsamında çok boyutlu doküman koleksiyonlarının yüksek başarımla ve verimli bir şekilde kümelenebilmesi amacıyla, K-Means algoritmasında değişiklik yapılmış olup, esnek kümeleme ile örtüşen kümeler oluşabilmesi fikrini K-Means algoritmasına uygulanmış ve dokümanların kümelere benzerliklerine göre belli ölçüde birden çok kümeye dahil olmasına olanak sağlayan özgün bir algoritma olan bulanık K-Means algoritması geliştirilmiştir (Adalı, 2011).

H. Nunez ve E. Ramos, K – En Yakın Komşuluk algoritmasını kullanarak bir sınıflandırma modeli geliştirmişlerdir. 10 katlı çapraz kontrol yöntemini

kullanarak %82 başarı oranında doğrulukla sınıflandırma yapmışlardır. Dokümanlar arasında belirgin olan iki kategoriyi dikkate alarak sınıflandırma yapıldığında %95 doğruluk oranı elde etmişlerdir. Bu sınıflandırıcı, bir uygulamaya entegre edilerek dokümanlar gözden geçirmelerini yapacak kişileri ilgi alanlarına göre otomatik olarak belirleyen bir sistem oluşturulmuştur (Nunez ve Ramos, 2012).

A. Kısayol ve M. Turan, yapmış oldukları çalışmada İngilizce dokümanlar üzerinde bir konuya ait birden fazla doküman içerisinden otomatik özet çıkarmayı amaçlamıştır. Özet çıkarma işlemi yapılırken kümeleme algoritmaları kullanılarak aynı bilgiyi içeren paragrafların seçimi minimuma indirilmeye çalışılmıştır. Geliştirilen sistemde özet oranı arttırıldıkça daha da iyi sonuçlar elde edilmiştir (Kısayol ve Turan, 2018).

Türkçe dilinde de metin madenciliği alanında çalışmalar yapılmış ve yapılmaya devam etmektedir.

Benzerlik bulunması ile ilgili yapılan çalışmada Bünyamin Dursun ve A. Coskun Sönmez, Türkçe metin benzerliklerinin bulunması için yeni bir yöntem önermişlerdir (Dursun ve Sonmez, 2008). Çalışmada Türkçe dili için sık yapılan hatalar tespit edilmiş ve bu hataların düzeltilmesi için çözümler önerilmiştir. Uygulanan çözümlerin başarıya ulaşıp ulaşmadığının tespiti için uygulanan çözümler, Levenshtein Edit Distance benzerliği ve Jaro-Winkler benzerliği ile karşılaştırılmış ve sonuçlar değerlendirilmiştir.

Tuğberk Kocatekin ve Devrim Ünay ise metin madenciliği kullanarak Türkçe dilindeki radyoloji raporlarını analiz etmişlerdir (Kocatekin and Ünay, 2013).

G. İşgüder-Şahin, H. R. Zafer ve E. Adalı tarafından popüler bir Türk web sitesinden teknoloji markaları ile ilgili yorumlar toplanmış ve olumlu veya olumsuz olarak sınıflandırılmıştır (İşgüder-Şahin, Zafer ve Adalı, 2014).

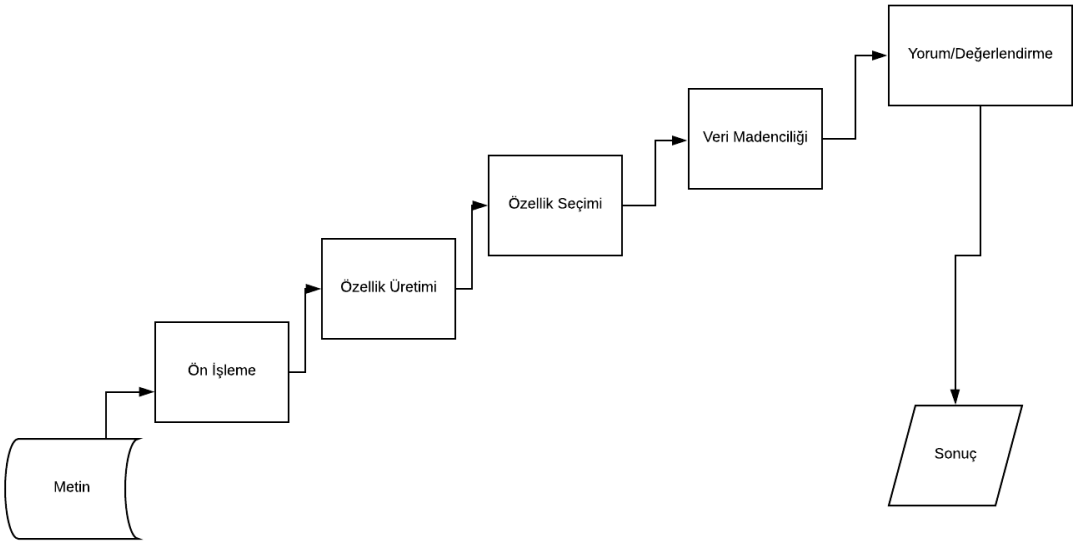
Gerek ülkemizde gerekse dünyada doğal dil işleminin alt disiplinleri olan metin sınıflandırma ve metin kümeleme alanlarında bilim insanları çalışmalarına devam etmekte, dijital verilerin yorumlanarak değerli bilgilere dönüşmesine katkıda bulunacak yöntem veya uygulamalar geliştirmektedirler.

3. VERİ İŞLEME ADIMLARI

İnternetin yaygınlaştırmasıyla birlikte veri üretiminin ve veriye ulaşmanın gelişmesiyle beraber araştırmalarda ve veri toplama işlemlerinde verilerin genellikle büyük olması, heterojen olması ve dağıtık olmalarından kaynaklı olarak birçok kirli, gürültülü ve tutarsız veri ortaya çıkmaktadır. Bu verilerin ham hâliyle işlenmesi sonuçların yetersiz ve yanlış olmasına sebebiyet verebilir ve bu da verilerin kalitesi düşürmektedir. İyi bir araştırma yapmak için kaliteli verilerle işlem yapılmalıdır. Bir verinin kalitesi o verilerle yapılacak olan çalışmanın başarı oranının artmasına olanak sağlamakta ve çalışmayı gelişime açık kılmaktadır. Verilerin özünün anlaşılması ve daha anlamlı bir veri analizi yapılabilmesi açısından veri ön işleme önemli bir kısımdır. Veri kümesinden anlamlı bir bilgi çıkarılmasında en büyük etkenlerden biri de veri ön işlemede anlamlı bilginin çıkarılmasıdır. Veri ön işlemesi ile ilgili birçok teknik bulunmaktadır. Özellikle metin madenciliğinde veri ön işleme adımları önem teşkil etmektedir. Bilgisayarlar, açık, kesin ve sık sık yapılandırılmış programlama dillerinde insanlarla etkileşime giriyor. Ancak, doğal (insan) dilin çok fazla belirsizliği vardır. Aynı anlamı taşıyan (eş anlamlılar), birden fazla anlamı olan kelimeler (çok yönlülük), bazıları doğada tam tersi olan kelimeler (oto-zıtlıklar) ve isim ve fiil olarak kullanıldığında farklı davranış gösteren kelimeler vardır. Bu kelimeler bağlamsal olarak, insanların kolayca anlayabileceği ve ayırt edebildiği doğal dilde anlamlıdır, ancak makineler bu ayrımı kolay kolay yapamaz. NLP'yi yapay zekâ (AI) alanındaki en zor ve ilginç alanlardan biri yapan da budur.

3.1 Ön İşleme Teknikleri

Büyük miktardaki metinsel verilerden potansiyel olarak yararlı ve önceden bilinmeyen belirli bir önemi olan bilginin çıkarılması olarak nitelendirilen Metin Madenciliği şekilde görüldüğü gibi temelde altı adımdan oluşmaktadır. Metin Madenciliği işlemleri, Veri Madenciliğine benzer olarak Şekil 3.1.'deki gibi özetlenebilir.



Şekil 3.1. Veri Madenciliği İşlem Adımları

Dokümanlar üzerinde metin madenciliği işlemleri yapılmadan önce verilerin temizlenmesi gerekir. Veri temizleme, ham olarak elde edilen verilerin içerisinde yer alan gürültülü, eksik ve tutarsız verilerin çıkarılması işlemidir. Toplanan verilerin ön işleme safhasındaki ilk aşamadır.

3.1.1. Cümle Segmentasyonu

Cümle ayrıştırma (cümle segmentasyonu da denir), bir yazı dilini bileşen cümlelerine bölme adımıdır. İngilizce ve diğer bazı dillerde, bir noktalama işareti gördüğümüzde cümleleri bölebiliriz. Ancak İngilizce'de bile, kısaltmalar için durma karakterinin, örneğin: U.S.A., S.M.A.R.T. gibi kısaltmalar kullanılması nedeniyle bu sorun önemsiz değildir. Düz metni işlerken, kısaltma kullanımları cümle sınırlarının yanlış atanmasını önlememize yardımcı olmaktadır.

3.1.2 Kelime Segmentasyonu

Kelime ayrıştırma (kelime segmentasyonu da denir), bir yazı dilini bileşen sözcüklerine ayırma problemidir. İngilizce'de ve bazı Latin alfabesi formlarını kullanan diğer dillerde, boşluk, sözcük ayırıcı için iyi bir yaklaşımdır. Bununla birlikte, istenen sonuçları elde etmek için yalnızca kullanımı yeterli olmayabilir. Bazı İngilizce bileşik isimler değişken bir şekilde yazılmıştır, örneğin, “back gammon”, “stock market”, “full stack” ve bazen boşluk içerirler. Diğer dil aileleri

için de kendi fonetik ve gramer yapılarına göre kısıtlar gözlenebilir. Örneğin, Türkçe'de önek yoktur. Almanca'daki gibi bir dizi isim kökü birbirine ekleyerek beraber yazılan birleşik isimler de bulunmaz gibi dillere özgü kurallar kısıtlar oluşturabilmektedir.

3.1.3 Lemmatizasyon ve Kök Bulma

Dilbilgisel nedenlerden dolayı, belgeler bir kelimenin farklı biçimlerini içerebilir. Ayrıca bazen benzer bir anlamdaki kelimelerle de ilişkili olabilir. Hem kök bulmanın hem de lemmatizasyonun amacı, çekim biçimlerini ve bazen bir kelimenin türevsel olarak ilişkili formlarını ortak bir temel forma indirmektir. Bu alanda yaygın olarak bilinen ve kullanılan algoritma Porter algoritmasıdır (Porter, 1980).

Kök bulma ve Lemmatizasyon normalizasyon yöntemleri olmasına rağmen aralarında farklar vardır. Stemming (kök bulma) genellikle, bu amaca ulaşmak için kelimelerin uçlarını kesen ve çoğu zaman türev eklerinin kaldırılmasını içeren kaba bir sezgisel süreci ifade ederken lematizasyon ise, genellikle kelimeleri kullanarak, kelimelerin kelime ve morfolojik analizlerini çıkarır, normal olarak sadece kelimelere eklenen son ekleri kaldırmayı ve lemma olarak bilinen bir kelimenin temelini veya sözlük biçimini döndürmeyi amaçlayan şeyleri doğru şekilde yapmak anlamına gelir. Aradaki fark, bir kökün bağlam bilgisi olmadan çalışması ve bu nedenle konuşmanın bir kısmına bağlı olarak farklı anlamları olan kelimeler arasındaki farkı anlayamamasıdır. Ancak kök bulma yöntemi de bazı avantajlara sahiptir, uygulanması kolaydır ve genellikle daha hızlı çalışırlar. Ayrıca, "doğruluk" bazı uygulamalar için önemli kriter olmayabilir.

3.1.4 Anlamsız Kelimelerin (Stop words) Atılması

Anlamsız kelimeler, metnin işlenmesinden önce veya sonra filtrelenen sözcüklerdir. Makine öğrenmesini metne uygularken, bu kelimeler metnin içerisinde çok fazla gürültüye sebep olabilmektedir. Bu nedenle bu anlam ifade etmeyen kelimeler metinin içinden atılır. Anlamsız sözcükler genellikle dile özgüdür ve örnek olarak "ve", "veya" gibi kelimelerdir, ancak evrensel bir

anlamsız sözcük listesi yoktur. Yazılım dilleri için çeşitli hazır stop word listeleri kullanılarak metinler daha sade ve semantik analizine ve sınıflandırmaya uygun hale getirilir. Çince, Japonca gibi sembollerin ağırlıkta olduğu dillerde jieba kütüphanesinden faydalanılabilir.

3.2 Kelimelerin Ağırlıklandırılması

Dokümanlar içerisinde yer alan kelimeler ve onların ağırlıkları ile temsil edilirler. Ağırlıklandırma işleminin başarısı, sınıflandırma ve kümeleme işlemlerinin başarısında önemli rol oynamaktadır. Kelime ağırlıklandırması için çeşitli yöntemler önerilmektedir.

- Terim Frekansı
- Ters Doküman Frekansı
- Terim Frekansı- Ters Doküman Frekansı
- Terim Ayrıştırma Değeri
- Olasılıksal Terim Ağırlıklandırma
- Tek Terim Doğruluğu
- Genetik Algoritmalar

Gibi teknikler kullanılarak kelime ağırlıklandırması yapılabilmektedir.
Ön işleme sonucunda çıkan tekilleştirilmiş kelimelerin, dokümanlardaki önem düzeyini bulmak için ağırlıklandırma işlemi uygulanır. Kelimelerin anlam değerlerinin bulunması bu kelimeler için anahtar kelime seçimlerinin yapılmasına olanak tanımaktadır. Ağırlıklandırma işleminde; Helmoltz tabanlı Gestalt insan algı teoreminin eğitimli anlamsal özellik seçiminden yola çıkılmıştır.

Gestalt, Almanya'da ortaya çıkan; algı ve kavrama süreçlerine odaklanarak, algıya yön veren temel yasaları tanımlayan bir psikoloji kuramıdır. Bu kuram basitçe, bütünün, onu oluşturan parçaların toplamı değil, daha fazlası olduğunu savunur. İstistiksel fizik ile doğrulanabilen bu yöntem beklenti değeri hesaplamaları kullanılarak elde edilmiştir. Bu teori rastgele oluşturulmuş bir görüntü içerisinde rahatlıkla algılanabilen bir geometrik yapının şans eseri olmayacağının, bunun bir anlamı olduğunu söylemektedir (Balinsky H., Balinsky A., Simske S., 2011).

3.3 Anlamlı Kelimelerin Tespiti

Doküman içerisinde geçen anlamlı kelimelerin tespiti için Helmholtz prensibi kullanılmıştır. Helmholtz prensibi, Gestalt insan algı teorisini kullanır. Gestalt teorisi, zihinsel figür gibi imgeleri nasıl düzenlediğimiz ve görüntüleri görsel, işitsel ve koku uyarıcıları gibi duyusal girdiler yoluyla nasıl algıladığımız açıklamaktadır. Tecrübelerimizi düzenli simetrik ve basit şekilde düzenleme eğiliminde olduğumuz fikrine dayanmaktadır. Bu çalışmada Helmholtz teorisi, her bir kelime için; doküman paragrafında, kelimenin m kez geçmesinin olası olup olmadığının belirlenmesinde kullanılmıştır. Anahtar kelimeler çıkarma sorununa uygulandığında, parametre içermeyen yöntemleri kullanarak anlamlı kelimeleri tanımlamak için hızlı ve etkili araçlar sunar. Ayrıca, farklı uygulama ihtiyaçları için seçilen anahtar kelime kümelerinin boyutunun kontrol edilmesini sağlayan dokümanların anlamlılık seviyeleri tanımlanmıştır (B. Dadachev, A. Balinsky, H. Balinsky ve S. Simske, 2012).

3.4 Doküman Vektörlerinin Oluşturulması

Makine öğrenmesi algoritmaları çoğu zaman sayısal özellik vektörlerini girdi olarak alır. Bu nedenle, metin belgeleriyle çalışırken, her belgeyi sayısal bir vektöre dönüştürmek gerekmektedir. Bu işlem, metin vektörleşmesi olarak bilinir. Yaygın bir vektörleştirme yaklaşımı, her belgeyi bir kelime sayımı vektörü olarak temsil etmektir. Bir metni vektörleştirmek için birçok farklı yol vardır ve kullanılan yöntem çoğu zaman çok önemli olabilir. Bir kelime sayımı vektörü oluşturmak yerine, her bir dizinin bir kelimenin varlığını veya yokluğunu gösterdiği bir ikili vektör de oluşturabilmektedir. Ayrıca her bir kelime önem derecesine göre de ağırlıklandırılabilmektedir. Vektör uzayı oluşturmak için kullanılan çeşitli yaklaşımlar vardır.

Bir dokümanın “ağaç, yeşil, bulut, doğa, dünya” kelimeleri ile temsil edildiğini varsayalım. Vektörün boyu temsil edilen kelime sayısı kadardır. Örneğin bu kelimelerin frekansları aşağıdaki gibi verilmiş olsun:

V [2,3,1,4,2]

Bu vektöre göre ağaç kelimesinin dokümanda 2 kere, yeşil kelimesinin 3 kere, bulut kelimesinin 1 kere, doğa kelimesinin 4 kere ve dünya kelimesinin de 2 kere geçtiği anlaşılmaktadır.

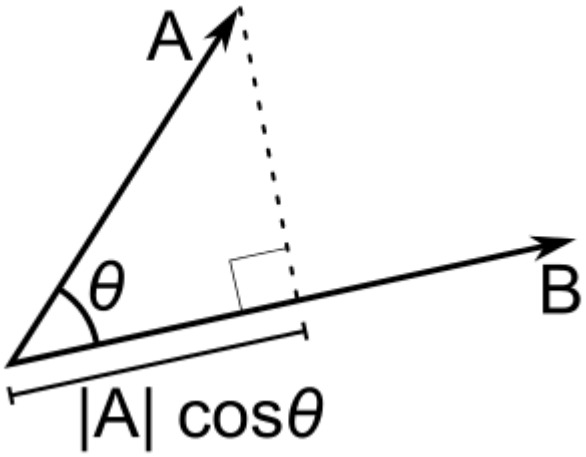
İkili vektör yaklaşımında, metin verileri 1 ve 0'lar ile ifade edilmektedir. Veri içinde barındırdığı kelimelerin metin içerisindeki varlıklarına göre bu değerleri almaktadır. Veri kümesi içerisindeki kelimelerin alacağı değerler binary vektör temsilinde V [1,1,1,1,1] şeklinde olmaktadır.

Frekans vektör yaklaşımında, binary tanımlamadan farklı olarak veri içinde bulunan kelime köklerinin kaç defa geçtiği bilgisi tutularak yapılmaktadır. Veri kümesi içerisindeki kelimelerin alacağı değerler frekans vektörü temsilinde [2, 0, 5, 6, ...] şeklinde olmaktadır.

TF-IDF vektör yaklaşımında her bir dokümandaki kelimelerin frekansları göz önüne alınmaktadır. TF değeri frekans bilgisini tutmaktadır. (Veri kümesi içinde kaç kez geçtiği bilgisi). IDF ise tüm dokümanlarda seyrek görülen kelimeler ile ilgili bir ölçü vermektedir. Kelime tüm eğitim dokümanları içerisinde sadece o dokümanda geçiyor ise o doküman için belirleyici özelliği olmaktadır.

3.5. Kosinüs Benzerliği Yöntemi ile Benzerlik Hesaplama

Kosinüs benzerliği, iki vektör arasındaki açı kosinüsünün ölçüsüdür; bu yaklaşımda iki vektör, tf-idf ağırlık vektörü olarak temsil edilen metin belgeleridir. Kosinüs açısı, dokümanlar arasındaki içeriği açısından üst üste binmenin ölçüsüdür. Nokta ürün denklemini kullanarak iki belge arasındaki kosinüs benzerliği hesaplanabilmektedir.



Şekil 3.2. İki Vektörün Açısal Görünümü

Sıfır olmayan iki vektörün kosinüsü, Öklid formülü kullanılarak hesaplanır:

$A * B = ||A||\ ||B||\ \cos \boldsymbol{\theta}$ (3.1)

İki özellik vektörü göz önüne alındığında A ve B için cos $\boldsymbol{\theta}$ bir nokta çarpımı olarak kullanılmıştır. Aşağıda Kosinüs benzerlik formülü verilmektedir.

$$\text{Benzerlik} = \cos(\boldsymbol{\theta}) = \frac{\boldsymbol{A.B}}{||\boldsymbol{A}||||\boldsymbol{B}||} = \frac{\sum_{i=0}^{n} \boldsymbol{AiBi}}{\sqrt{\sum_{i=1}^{n} \boldsymbol{A_i^2}}\ \sqrt{\sum_{i=1}^{n} \boldsymbol{B_i^2}}} \quad (3.2)$$

"Kosinüs benzerliği" terimi bazen yukarıda verilen benzerliğin farklı bir tanımını belirtmek için kullanılır. Bununla birlikte, "kosinüs benzerliği" IR (Information Retrieval) ve metin madenciliği alanlarında kullanılır. Aşağıda tanımlanan benzerlik ve mesafe ölçümleri sırasıyla "açısal benzerlik" ve "açısal mesafe" olarak adlandırılır. Vektörler arasındaki normalize açı, resmi bir mesafe ölçüsüdür ve yukarıda belirtilen benzerlik puanından. Bu açısal mesafe metriği, 0 ile 1 arasında sınırlandırılmış bir benzerlik fonksiyonunu hesaplamak için kullanılabilir.

Eğer A = [A1, A2] T o zaman $\bar{A} = \left[\frac{\boldsymbol{A1+A2}}{\boldsymbol{2}}, \frac{\boldsymbol{A1+A2}}{\boldsymbol{2}}\right]^{\boldsymbol{T}}$,

Yani $A- \bar{A} = \left[\frac{\boldsymbol{A1-A2}}{\boldsymbol{2}}, \frac{\boldsymbol{-A1+A2}}{\boldsymbol{2}}\right]^{\boldsymbol{T}}$ (3.3)

İki doküman arasındaki benzerliğin kosinüs yöntemi ile bulanabilmesi için Şekil 3.3'teki formül kullanılabilir: (Bhushan S.N. B, Danti A., 2018)

$$\text{BenzerlikCosine} = \frac{(\text{doc1} \,.\, doc2)}{(\text{doc1} \,.doc\, 1)^{1/2}\,(doc2\,.doc2)^{1/2}} \tag{3.4}$$

3.6. Jaccard Benzerliği Yöntemi ile Benzerlik Hesaplama

Jaccard Benzerliği veya Jaccard Katsayısı olarak da bilinen benzerlik ölçütü, tek terimli benzerliği hesaplamak için kullanılır. Kosinüs benzerliği ölçümü gibi, doğrudan belge vektör gösterimi üzerinden hesaplanır.

Genişletilmiş Jaccard benzerliği, iki nesnenin paylaşılan parçalarının, nesnelerin tüm parçalarına oranıdır. Nesneler belge vektörleri, parçalar ise kelimeler ile temsil edilir. Formül daha basite indirgendiğinde, iki vektörün kesişim kelimelerinin birleşim kelimelerine oranı olarak ifade edilebilir. Yüksek uzaklık değerlerinde Kosinüs'e, düşük değerlerde ise Öklid'e benzer özellik göstermektedir (Gover, J.C.,1971). Formül 3.4' de genişletilmiş Jaccard benzerliğinin kümeleme özelliği görülmektedir.

$$S^{(j)}(d, d^*) = \frac{\boldsymbol{d} \cap \boldsymbol{d}^*}{\boldsymbol{d} \cup \boldsymbol{d}^*} \tag{3.5}$$

İki doküman arasındaki benzerlik Şekil 3.8'da verilen formülle hesaplanabilmektedir. (Bhushan S.N. B, Danti A., 2018)

$$\text{Benzerlik}_{\text{Jaccard}} = \frac{\boldsymbol{doc_1}.\ \boldsymbol{doc_2}}{\mathbf{doc}_1.\ \boldsymbol{doc_1} + \boldsymbol{doc_2}\,.\,\boldsymbol{doc_2} - \boldsymbol{doc_1}.\ \boldsymbol{doc_2}} \tag{3.6}$$

Jaccard benzerlik ölçüsü, tekli benzerlik ölçütlerinden daha verimli sonuçlar üretmesine rağmen, basitlik ve düşük hesaplama süresi nedeniyle tekli benzerli ölçütleri daha fazla tercih edilmektedir. Zaman karmaşıklığı birçok yazılım uygulamasının çok önemli bir kriteri ve bu uygulamalar çevrimiçi veri organizasyonu içerdiğinde daha önemli hale gelir (Madylova A, 2009).

3.7. Noktasal Karşılıklı Bilgi Yöntemi ile Benzerlik Hesaplama

Noktasal karşılıklı bilgi (PMI) veya nokta karşılıklı bilgi, bilgi teorisi ve istatistik biliminde kullanılan bir birliktelik ölçüsüdür. PMI, üzerine inşa eden karşılıklı bilginin (MI) aksine, tek olaylara atıfta bulunur, MI ise ortalama olası tüm değerleri kapsamaktadır. X ve Y ayrık rastgele değişkenlere ait x ve y sonuçlarının noktasal olarak karşılıklı bilgileri, ortak dağılımları göz önüne alındığında, kendi ortak dağılımlarının tesadüf olasılığı ile bireysel dağılımları arasındaki tutarsızlığı ölçmektedir. Aşağıdaki formül ile hesaplanmaktadır.

$$pmi(x,y) = \log P(x, y) / P(x)P(y) \quad (3.7)$$

X ve Y rasgele değişkenlerinin karşılıklı bilgi (MI) PMI beklenen değeridir. PMI (x, y) ve PMI (y, x) aynı sonuçları verir. Pozitif veya negatif değerler alabilir ancak X ve Y bağımsızsa sıfırdır. PMI'nin negatif ya da pozitif olmasına rağmen, tüm ortak olaylara (MI) beklenen sonucunun pozitif olacağı göz ardı edilmemelidir.

Hesaplamalı dil biliminde, PMI kelimeler arasındaki sıralamaları ve benzerlikleri bulmak için kullanılmıştır. Örneğin, bir metin içerisinde yer alan kelimelerin tek başlarına kullanılmaları ve birlikte kullanımları göz önüne olarak benzerlik hesaplanabilmektedir (K. W. Church and H. Patrick, 1990).

3.8. Benzerlik Ölçütlerinin Karşılaştırılması

Önceden belirlenmiş sınıflara sahip dokümanlar için Kosinüs ve Jaccard benzerliği ve PMI benzerlik ölçütleri karşılaştırılmıştır. Benzerlik oranı ortalama üzerinde olan dokümanların tümü baz alındığında Kosinüs benzerlik ölçütü %75, Jaccard benzerliği %40, PMI benzerlik ölçütü ise %55 başarı sağlamıştır. Buna rağmen doğruluk değerleri baz alındığında Kosinüs benzerlik ölçütü %80, Jaccard benzerliği %65 ve aynı şekilde PMI benzerlik ölçütü de %65 başarı sağlamıştır. Her bir dokümanın benzerlik katsayılarının ortalamaları baz alınarak yapılan sınıflama ise anlamlı kelimelerin yüzdelik değerlerine göre farklı başarımlar elde edilmiştir. Bu bakımdan PMI benzerlik ölçütü anlamlı kelime dağılımlarına adaptif bir yaklaşım sergiler iken Kosinüs benzerlik ölçütünde ve Jaccard benzerliğinde herhangi bir iyileşme gözlemlenmemiştir.

Çizelge 3.1'de sisteme girdi olarak verilen 20 rastgele doküman için Kosinüs, Jaccard ve PMI benzerlik ölçütlerinde alınan sonuçlara göre tahmini sınıflandırma sonuçlar paylaşılmıştır. Anlamlı kelimelerin %100'ü kullanılarak hesaplama ve sınıflandırma kontrolü yapıldığında Kosinüs benzerlik ölçütü %70, Jaccard benzerlik ölçütü %75, PMI benzerlik ölçütü ise %60 doğruluk ile doküman sınıflandırma işlemini gerçekleştirmişlerdir.

DocumentID	Önceden Verilmiş Sınıf	EducationCosine	SportsCosine	EducationJaccard	SportsJaccard	EducationPMI	SportsPMI	Kosinus Sınıf Eşleşmesi	Jaccard Sınıf Eşleşmesi	PMI Sınıf Eşleşmesi
0	Education	**0.148990**	0.119535	0.745747	**0.762551**	**44,374625**	26,469980	DOĞRU	YANLIŞ	DOĞRU
1	Education	**0.185445**	0.086015	**0.714440**	0.685026	11,166056	**11,739412**	DOĞRU	DOĞRU	YANLIŞ
2	Education	**0.155747**	0.082146	**0.715165**	0.710039	11,618104	**12,719975**	DOĞRU	DOĞRU	YANLIŞ
3	Education	**0.196831**	0.112581	0.730259	**0.750354**	15,628816	**16,296350**	DOĞRU	YANLIŞ	YANLIŞ
4	Education	**0.175772**	0.078806	0.751890	**0.778283**	**20,722472**	19,401942	DOĞRU	YANLIŞ	DOĞRU
5	Education	**0.121591**	0.082397	**0.724264**	0.711810	18,134948	**30,589308**	DOĞRU	DOĞRU	YANLIŞ
6	Education	**0.152336**	0.119817	0.766477	**0.783921**	24,366328	**29,084808**	DOĞRU	YANLIŞ	YANLIŞ
7	Education	**0.196745**	0.097079	**0.753952**	0.743736	12,881452	**18,775050**	DOĞRU	DOĞRU	YANLIŞ
8	Education	**0.085504**	0.063208	**0.763954**	0.738666	15,762737	**18,257641**	DOĞRU	DOĞRU	YANLIŞ
9	Education	**0.121551**	0.040135	**0.649838**	0.636833	9,209429	**11,369741**	DOĞRU	DOĞRU	YANLIŞ
10	Sports	**0.158364**	0.154848	0.731515	**0.741891**	6,711039	**25,841193**	DOĞRU	DOĞRU	DOĞRU
11	Sports	0.036335	**0.128739**	0.694522	**0.698535**	2,372627	**7,111401**	YANLIŞ	DOĞRU	DOĞRU
12	Sports	0.086458	**0.119428**	0.748884	**0.790770**	8,832967	**25,971739**	YANLIŞ	DOĞRU	DOĞRU
13	Sports	0.094488	**0.070117**	0.667213	**0.671085**	9,563137	**14,050180**	YANLIŞ	DOĞRU	DOĞRU
14	Sports	0.081875	**0.083002**	0.722416	**0.756575**	9,056293	**23,187546**	DOĞRU	DOĞRU	DOĞRU
15	Sports	0.051908	**0.075485**	0.708461	**0.746956**	6,882267	**16,444478**	DOĞRU	DOĞRU	DOĞRU
16	Sports	0.080408	**0.067922**	0.740780	**0.803741**	11,908392	**20,782580**	YANLIŞ	DOĞRU	DOĞRU
17	Sports	0.135863	**0.128200**	**0.757721**	0.755943	9,351473	**15,076442**	YANLIŞ	YANLIŞ	DOĞRU
18	Sports	0.074422	**0.187578**	0.764437	**0.781579**	4,567182	**16,318355**	DOĞRU	DOĞRU	DOĞRU
19	Sports	0.072597	**0.115266**	0.765270	**0.770100**	8,344630	**29,925445**	YANLIŞ	DOĞRU	DOĞRU
								Doğruluk Oranı:**0,7%**	Doğruluk Oranı:**0,75%**	Doğruluk Oranı:**0,6%**

Çizelge 3.1: Anlamlı kelimelerin %100'ü ile yapılan hesaplama

Çizelge 3.2'de sisteme girdi olarak verilen 20 rastgele doküman için Kosinüs, Jaccard ve PMI benzerlik ölçütlerinde alınan sonuçlara göre tahmini sınıflandırma sonuçlar paylaşılmıştır. Anlamlı kelimelerin %50'ü kullanılarak hesaplama ve sınıflandırma kontrolü yapıldığında Kosinüs benzerlik ölçütü %70, Jaccard benzerlik ölçütü %75, PMI benzerlik ölçütü ise %50 doğruluk ile doküman sınıflandırma işlemini gerçekleştirmişlerdir.

DocumentID	Önceden Verilmiş Sınıf	EducationCosine	SportsCosine	EducationJaccard	SportsJaccard	EducationPMI	SportsPMI	Kosinus Sınıf Eşleşmesi	Jaccard Sınıf Eşleşmesi	PMI Sınıf Eşleşmesi
0	Education	**0.148990**	0.110535	0.745747	**0.762551**	**5,417832**	1,625349	DOĞRU	YANLIŞ	DOĞRU
1	Education	**0.185445**	0.086015	**0.714440**	0.685026	**3,232620**	0,572067	DOĞRU	DOĞRU	DOĞRU
2	Education	**0.155747**	0.082146	**0.715165**	0.710039	2,788176	**3,813781**	DOĞRU	DOĞRU	YANLIŞ
3	Education	**0.196831**	0.112581	0.730259	**0.750354**	**8,035875**	3,778957	DOĞRU	YANLIŞ	DOĞRU
4	Education	**0.175772**	0.078806	0.751890	**0.778283**	**20,722472**	19,401942	DOĞRU	YANLIŞ	DOĞRU
5	Education	**0.121591**	0.082397	**0.724264**	0.711810	4,920718	**5,147165**	DOĞRU	DOĞRU	YANLIŞ
6	Education	**0.152336**	0.119817	0.766477	**0.783921**	4,177256	**5,537637**	DOĞRU	YANLIŞ	YANLIŞ
7	Education	**0.196745**	0.097079	**0.753952**	0.743736	**4,802463**	1,559565	DOĞRU	DOĞRU	DOĞRU
8	Education	**0.085504**	0.063208	**0.763954**	0.738666	**3,313573**	1,641944	DOĞRU	DOĞRU	DOĞRU
9	Education	**0.121551**	0.040135	**0.649838**	0.636833	**3,480649**	1,010427	DOĞRU	DOĞRU	DOĞRU
10	Sports	**0.158364**	0.154848	0.731515	**0.741891**	6,711039	**25,841193**	DOĞRU	DOĞRU	DOĞRU
11	Sports	0.036335	**0.128739**	0.694522	**0.698535**	0,461917	**1,906267**	YANLIŞ	DOĞRU	DOĞRU
12	Sports	0.086458	**0.119428**	0.748884	**0.790770**	1,925660	**9,567742**	YANLIŞ	DOĞRU	DOĞRU
13	Sports	0.094488	**0.070117**	0.667213	**0.671085**	**2,372607**	2,200213	YANLIŞ	DOĞRU	YANLIŞ
14	Sports	0.081875	**0.083002**	0.722416	**0.756575**	**3,586787**	0,140943	DOĞRU	DOĞRU	YANLIŞ
15	Sports	0.051908	**0.075485**	0.708461	**0.746956**	**1,486624**	1,331948	DOĞRU	DOĞRU	YANLIŞ
16	Sports	0.080408	**0.067922**	0.740780	**0.803741**	**4,704320**	-0,298912	YANLIŞ	DOĞRU	YANLIŞ
17	Sports	0.135863	**0.128200**	**0.757721**	0.755943	**1,939874**	1,450643	YANLIŞ	YANLIŞ	YANLIŞ
18	Sports	0.074422	**0.187578**	0.764437	**0.781579**	**2,185982**	1,655232	DOĞRU	DOĞRU	YANLIŞ
19	Sports	0.072597	**0.115266**	0.765270	**0.770100**	**2,522206**	1,492643	YANLIŞ	DOĞRU	YANLIŞ
								Doğruluk Oranı:**0,7%**	Doğruluk Oranı:**0,75%**	Doğruluk Oranı:**0,5%**

Çizelge 3.2: Anlamlı kelimelerin %50'si ile yapılan hesaplama

Çizelge 3.3'de sisteme girdi olarak verilen 20 rastgele doküman için Kosinüs, Jaccard ve PMI benzerlik ölçütlerinde alınan sonuçlara göre tahmini sınıflandırma sonuçlar paylaşılmıştır. Anlamlı kelimelerin %25'i kullanılarak

hesaplama ve sınıflandırma kontrolü yapıldığında Kosinüs benzerlik ölçütü %70, Jaccard Katsayısı %75, PMI benzerlik ölçütü ise %89 doğruluk ile doküman sınıflandırma işlemini gerçekleştirmişlerdir.

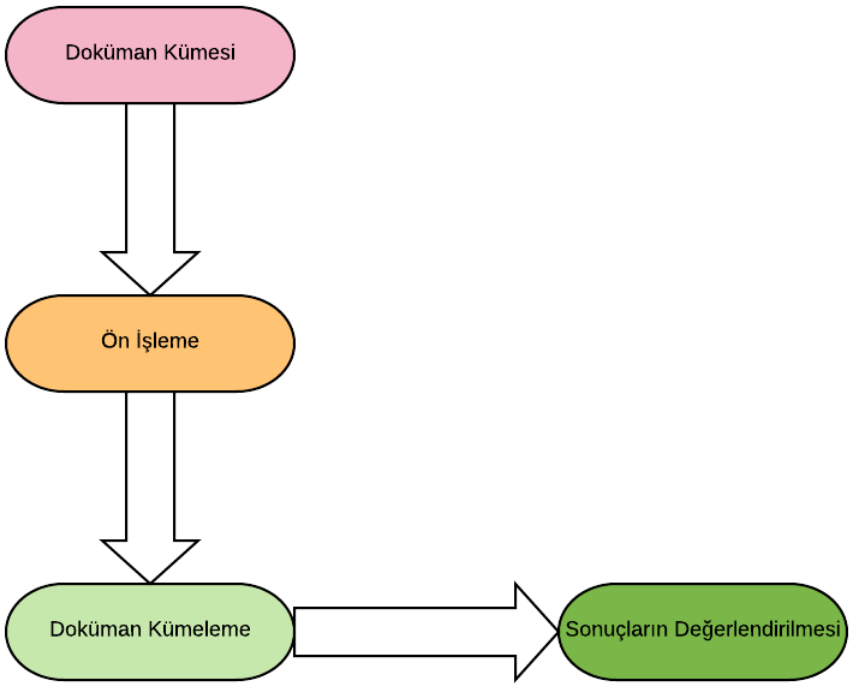
DocumentID	Given Class	EducationCosine	SportsCosine	EducationJaccard	SportsJaccard	EducationPMI	SportsPMI	Kosinus Sınıf Eşleşmesi	Jaccard Sınıf Eşleşmesi	PMI Sınıf Eşleşmesi
0	Education	**0.148990**	0.110535	0.745747	**0.762551**	**7,024364**	1,264385	DOĞRU	YANLIŞ	DOĞRU
1	Education	**0.185445**	0.086015	**0.714440**	0.685026	0,000000	0,000000	DOĞRU	DOĞRU	N/A
2	Education	**0.155747**	0.082146	**0.715165**	0.710039	1,968963	**7,678957**	DOĞRU	DOĞRU	YANLIŞ
3	Education	**0.196831**	0.112581	0.730259	**0.750354**	**5,422693**	1,728771	DOĞRU	YANLIŞ	DOĞRU
4	Education	**0.175772**	0.078806	0.751890	**0.778283**	**8,222094**	3,025349	DOĞRU	YANLIŞ	DOĞRU
5	Education	**0.121591**	0.082397	**0.724264**	0.711810	**1,247646**	0,532192	DOĞRU	DOĞRU	DOĞRU
6	Education	**0.152336**	0.119817	0.766477	**0.783921**	**8,702308**	6,482892	DOĞRU	YANLIŞ	DOĞRU
7	Education	**0.196745**	0.097079	**0.753952**	0.743736	**3,455721**	2,762756	DOĞRU	DOĞRU	DOĞRU
8	Education	**0.085504**	0.063208	**0.763954**	0.738666	**4,766372**	2,593156	DOĞRU	DOĞRU	DOĞRU
9	Education	**0.121551**	0.040135	**0.649838**	0.636833	**1,727860**	0,000000	DOĞRU	DOĞRU	DOĞRU
10	Sports	**0.158364**	0.154848	0.731515	**0.741891**	2,045763	**7,095904**	DOĞRU	DOĞRU	DOĞRU
11	Sports	0.036335	**0.128739**	0.694522	**0.698535**	0,864385	**2,881285**	YANLIŞ	DOĞRU	DOĞRU
12	Sports	0.086458	**0.119428**	0.748884	**0.790770**	0,000000	**5,665787**	YANLIŞ	DOĞRU	DOĞRU
13	Sports	0.094488	**0.070117**	0.667213	**0.671085**	3,010149	**3,021622**	YANLIŞ	DOĞRU	DOĞRU
14	Sports	0.081875	**0.083002**	0.722416	**0.756575**	**5,744809**	0,081885	DOĞRU	DOĞRU	YANLIŞ
15	Sports	0.051908	**0.075485**	0.708461	**0.746956**	2,045763	**6,680866**	DOĞRU	DOĞRU	DOĞRU
16	Sports	0.080408	**0.067922**	0.740780	**0.803741**	2,319460	**3,112042**	YANLIŞ	DOĞRU	DOĞRU
17	Sports	0.135863	**0.128200**	**0.757721**	0.755943	2,477956	**7,417081**	YANLIŞ	YANLIŞ	DOĞRU
18	Sports	0.074422	**0.187578**	0.764437	**0.781579**	3,068645	**8,033831**	DOĞRU	DOĞRU	DOĞRU
19	Sports	0.072597	**0.115266**	0.765270	**0.770100**	4,091527	**9,474473**	YANLIŞ	DOĞRU	DOĞRU
								Doğruluk Oranı: **0,7%**	Doğruluk Oranı:**0,75%**	Doğruluk Oranı:**0,89%**

Çizelge 3.3: Anlamlı kelimelerin %25'i ile yapılan hesaplama

4. K-MEANS ÖBEKLEME ALGORİTMASI

4.1 Doküman Kümeleme

Belge kümeleme düzeninin amacı, kümeler arası mesafeleri en üst düzeye çıkarırken (belgeler arasında uygun bir mesafe ölçüsü kullanarak) belgeler arasında küme içi mesafeleri en aza indirmektir. Böylece bir uzaklık ölçüsü (veya ikili olarak benzerlik ölçüsü) belge kümelemesinin merkezinde yer alır. Çok çeşitli belgeler, her tür veri kümesi için en iyi şekilde çalışabilecek genel bir algoritma oluşturmayı neredeyse imkânsız hale getirir.



Şekil 4.1 Doküman Kümeleme Adımları

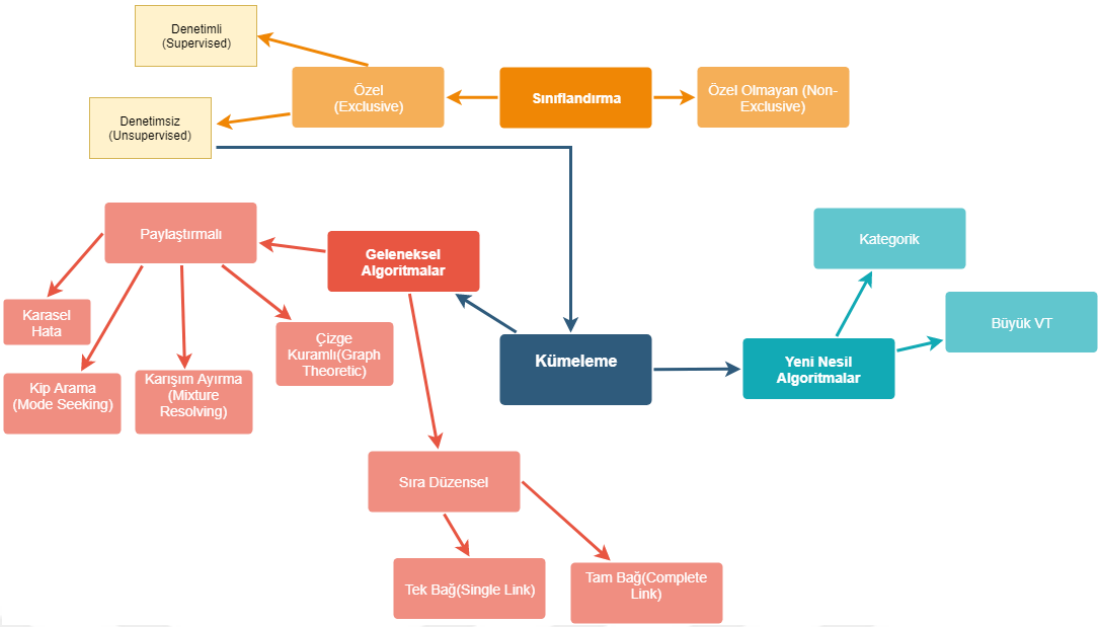
Doküman setlerinden kümelenmiş gruplar elde etmenin sadece tek bir işlem değil, daha çok aşamalı bir süreç olduğunu vurgulamak önemlidir. Bu aşamalar, tarama, indeksleme, ağırlıklandırma, filtreleme, vb. gibi daha geleneksel bilgi alma işlemlerini içerir. Bu diğer işlemlerin bazıları, çoğu kümeleme algoritmasının kalitesi ve performansı için merkezi bir noktadır ve bu aşamaları verilenlerle birlikte düşünmek gerekir.

Şekil4.1'de görsel olarak ifade edilen işlemlerden doküman kümesi içerisinde yer alan, kümelenmesi gereken belgeleri toplamak, daha iyi bir şekilde saklamak ve almak için dizine almak ve fazladan verileri kaldırmak üzere filtrelemek için kullanılan tarama, dizin oluşturma, filtreleme vb. işlemleri içerir. 2. Adımda verileri kümelemede kullanılabilecek bir biçimde göstermek için ön işleme yapılır. Vektör modeli, grafik modeli vb. dokümanları temsil etmenin birçok yolu vardır. Dokümanları ve benzerliklerini bulmak için birçok yöntem kullanılır. Doküman kümeleme işlemleri için seçilen algoritmalar ile veri seti işlenir, son işlem, doküman kümelemesinin kullanıldığı başlıca uygulamaları, örneğin, kullanıcılara haber makalelerini önermek için kümelemenin sonuçlarını kullanan öneri uygulamasını içerir.

4.2 Doküman Kümeleme Yöntemleri

Bu bölümde doküman kümeleme yöntemleri hakkında bilgiler verilecek ve doküman kümelemede kullanılan önemli ve bilinen yöntemlere değinilecektir. Doküman kümelemede birçok yöntem kullanılmaktadır. Her yaklaşımın vakaya göre avantajları ve dezavantajları bulunur, vakanın gerektirdiği yaklaşıma göre kümeleme yapmak sonuçların performansını ve doğruluğunu direkt olarak etkileyecektir.

Verilerin kümelere ayrılmasında kullanılan çok çeşitli algoritma ve yöntemler bulunur. Şekil 4.2'de paylaşıldığı üzere sınıflandırma ve kümelemeye ait birçok yöntem birbiriyle ilişkili veya ilişkisiz şekilde kullanılmaktadır.



Şekil 4.2 Sınıflandırma ve Kümeleme Çeşitleri

Özel sınıflandırma, nesnelerin parçalara ayrılmasıdır. Her bir nesne, tamamen bir tek kümeye veya bir altkümeye aittir. Özel-olmayan sınıflandırma ise, bir nesneyi birden fazla sınıfa dahil etmektir. Örneğin arabaları renklerine, motor hacimlerine göre sınıflandırmak özel sınıflandırmaya, evleri kat sayılarına göre ayırmak ise özel olmayan sınıflandırmaya örnek gösterilebilir. Dahili sınıflandırma da sadece yakınlık matrisi kullanılmaktadır. Bununla beraber, sınıflandırma yapılırken ön bilgiye sahip olunmadığından dolayı Öğreticisiz Öğrenme olarak adlandırılır. Harici sınıflandırma ise, yakınlık matrisi dışında, nesnelerin kategori özelliklerini de kullanmaktadır.

Sıra düzensel sınıflandırma, iç içe sıralanmış bölümlerden oluşmaktadır. Paylaştırmalı sınıflandırma ise tek bölümden oluşmaktadır. Bu nedenle sıradüzensel sınıflandırma, paylaştırmalı sınıflandırmanın özel bir durumudur [M. S. Durmuş, 2005].

Toplayıcı algoritmalar, başlangıçta her bir veriyi bir küme olarak kabul eder. Kademe kadem bu altkümeler, tek bir küme oluşuncaya kadar birleştirilmektedir. Bölücü algoritmalarda ise tüm veriler tek bir küme olarak kabul edilir ve bu küme kademe kademe altkümelere ayrılır.

Seri algoritmalar veriler üzerinde tek tek işlem yapmaktadır. Eşzamanlı algoritmalar ise, tüm veriler üzerinde aynı anda işlem yapar [M. S. Durmuş, 2005].

Verilerin kümelenmesi amacıyla sıkça kullanılan metotlardan biri de karışım-ayırma yöntemidir. Bu algoritmada amaç, her örneği doğru kısma yerleştirebilmektir. Yoğunluk tahmini veya yüksek yoğunluğua sahip bölge arama algoritmalarının tersine, bölgelerin şekli ve sayısı biliniyor kabul edilir.

Bulanık kümeleme, 1965 yılında Azerbaycanlı Lotfi Zadeh tarafından ortaya konulmuştur. Bu yönteme göre bir nesne üyelik derecesine göre farklı kümelere de ait olabilmektedir. Bulanık kümeleme teorisine dayanan kümeleme algoritmalarının çoğu paylaştırmalı kümeleme yapmaktadır. [M. S. Durmuş, 2005].

Bir diğer kümeleme yaklaşımı ise yapay sinir ağları ile kümelemedir. Yapay sinir ağları son dönemde yaygın olarak kümeleme ve sınıflandırma işlemlerinde kullanılmaktadır.

Burada bahsi geçen algoritmalar dışında farklı amaçlar da kullanılmak üzere birçok kümeleme algoritması bulunmaktadır. CURE Algoritması, DBSCAN algoritması, BIRCH algoritması bunlara örnek olarak verilebilir. Bir sonraki bölümde doküman kümelemede yaşanan zorluklara değinilecektir.

4.3 Doküman Kümelemenin Zorlukları

Doküman kümelemesi, onlarca yıldan beri çalışılmaktadır ancak yine de çözülmüş bir sorun olmaktan uzaktır. Araştırmacılar birtakım zorluklar ile karşılaşmaktadırlar. Bu zorluklar;

- Kümeleme için kullanılması gereken belgelerin uygun özelliklerinin seçilmesi
- Dokümanlar arasında uygun bir benzerlik ölçüsü seçilmesi
- Seçilen benzerlik ölçütünü kullanarak uygun bir kümeleme yöntemi seçmek.

- Kümeleme algoritmasının gerekli bellek ve CPU kaynakları açısından mümkün kılan etkin bir şekilde uygulanması
- Yapılan kümelenmenin kalitesini değerlendirmenin yollarını bulmak.

Bunların da ötesinde, orta ve büyük boyutlu doküman kümelerinde (10000'den fazla doküman), terim ilişkilerinin sayısı milyonlarca olmakta ve uygulanan kümeleme algoritmalarının gerçek dünya uygulamalarında kullanımı noktasında sıkıntılar oluşturmaktadır.

4.4. K-Means Öbekleme Algoritması ile Doküman Öbekleme

Kümeleme, verilerin yapısı hakkında bir sezgiyi elde etmek için kullanılan en yaygın keşifsel veri analizi tekniklerinden biridir. Farklı kümelerdeki veri noktaları çok farklıyken, aynı alt gruptaki (küme) veri noktalarının çok benzer olması nedeniyle verilerdeki alt grupların belirlenmesi görevi olarak tanımlanabilir. Başka bir deyişle, veri kümesindeki homojen alt grupları, her bir kümedeki veri noktalarının, Öklid esaslı mesafe veya korelasyona dayalı mesafe gibi benzerlik ölçütlerine göre mümkün olduğu kadar benzer olması için bulmaya çalışırız. Hangi benzerlik ölçütünün kullanılması kararı uygulamaya özeldir. Kümeleme analizi, özelliklere dayalı örnek alt gruplarını bulmaya çalıştığımız değerlere veya örneklere dayalı özellik alt gruplarını bulmaya çalıştığımız değerlere dayanarak yapılabilir. Kümeleme segment bölümlemesinde kullanılır; davranış veya nitelik açısından, görüntü segmentasyonu / sıkıştırma; birbirine benzer müşterileri bulmaya çalıştığımız yerlerde, Benzer bölgeleri bir arada gruplamaya çalıştığımız yerlerde, konulara göre doküman kümelemesi vb. Denetimli öğrenmenin aksine, kümelenme algoritmasının çıktısını gerçek etiketlerle karşılaştıracak temel gerçeğe sahip olmadığımız için kümelenme denetimsiz bir öğrenme yöntemi olarak kabul edilir.

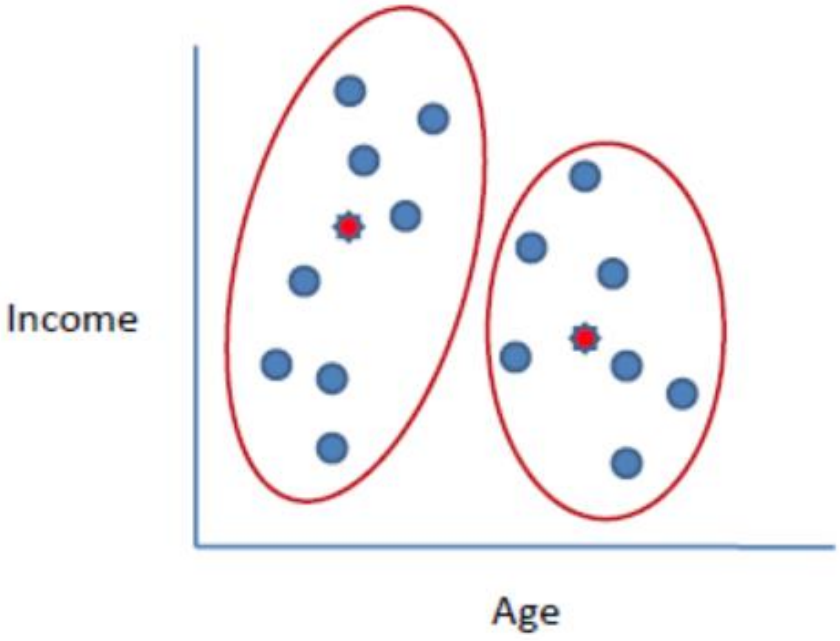
K-Means kümelenmesi, n nesnenin, her nesnenin kümeye en yakın ortalamaya ait olduğu k küme halinde bölünmesini amaçlamaktadır. Bu yöntem, mümkün olan en büyük ayrımın aynen farklı k kümelerini üretir.

En büyük ayrılmaya (mesafeye) yol açan, en iyi küme sayısı priori olarak bilinmektedir ve verilerden hesaplanmalıdır. K-Means kümelenmesinin amacı toplam küme içi varyansı veya kare hata fonksiyonunu en aza indirmektir.

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2 \quad (4.1)$$

4.5 K-Means Algoritması Adımları

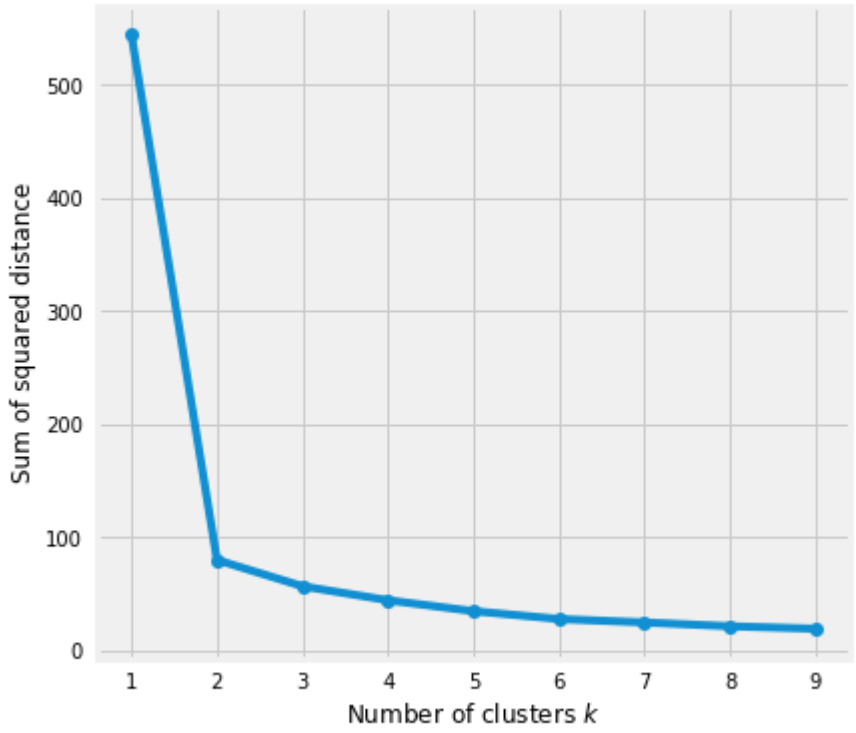
1. Veriler k'nin önceden tanımlandığı k gruplarına ayırır.
2. Küme merkezi olarak rastlantısal k noktaları seçilir.
3. Öklid mesafe işlevine göre nesneleri en yakın küme merkezine atanır.
4. Her kümedeki centroid veya tüm nesnelerin ortalamasını hesaplanır.
5. Her kümeye aynı noktalar ardışık olarak atanana kadar 2., 3. ve 4. adımları tekrarlanır.



Şekil 4.3. Kümeleme grafiği örneği (S. Sayad,2019)

4.6 Dirsek yöntemi

Dirsek yöntemi, iyi bir k sayısının kümelerin, veri noktaları ve atanmış kümelerinin centroidleri arasındaki kare mesafenin (SSE) toplamına bağlı olacağı hakkında bir fikir verir. SSE'nin düzleşmeye başladığı noktada bir dirsek oluşturulmaktadır. Geyser dataset'i kullanarak farklı k değerleri için dirsek eğrisinin nasıl olduğu Şekil 4. 'te verilmiştir (I. Dabbura, 2019).



Şekil 4.4. Elbow Dirseği (I. Dabbura, 2019)

Yukarıdaki grafik K değerinin 2 olarak belirlenmesi kötü sonuç vermeyeceğini göstermektedir. Bazen kullanılacak çok sayıda küme bulmak hâlâ zordur, çünkü eğri monoton bir şekilde azalır ve herhangi bir dirsek göstermeyebilir veya eğrinin düzleşmeye başladığı açık bir noktaya sahip olabilir.

4.7 Küme Kalitesini Değerlendirme

K-Means algoritmasını kullanmaktaki amaç sadece kümeler yapmak değil, aynı zamanda iyi ve anlamlı kümeler yapmaktır. Kalite kümelemesi, kümedeki veri noktalarının birbirine yakın olduğu ve diğer kümelerden uzak olduğu zamandır. Küme kalitesini ölçmek için iki tane yöntem tanımlanmıştır.

4.7.1 Süre-Durum

Sezgisel olarak atalet, bir küme içindeki noktaların ne kadar uzakta olduğunu gösterir. Bu nedenle küçük bir atalet hedefleniyor. Atalet değerinin değeri sıfırdan başlar ve artar.

4.7.2 Siluet Puanı

Siluet (Silhouette) puanı, bir kümedeki veri noktalarının başka bir kümedeki veri noktalarından ne kadar uzakta olduğunu söyler. Siluet puan aralığı -1 ile 1 arasındadır. Puan -1'den 1'e daha yakın olmalıdır.

4.8 K-Means Algoritması Kullanım alanları

K-Means nispeten etkili bir yöntemdir. Bununla birlikte, küme sayısını önceden belirtmemiz gerekir ve nihai sonuçlar başlatma işlemine karşı hassastır ve genellikle yerel olarak optimum bir şekilde sona erer. Ne yazık ki, optimum küme sayısını bulmak için global bir teorik yöntem yoktur. Pratik bir yaklaşım, birden fazla koşunun sonuçlarını farklı k ile karşılaştırmak ve önceden tanımlanmış bir kriteri temel alarak en iyisini seçmektir. Genel olarak, büyük bir k muhtemelen hatayı azaltır, ancak fazla uydurma riskini arttırır.

K-Means algoritması kullanılan birçok alan bulunmaktadır:

- Resim segmentasyonu
- Kümeleme gen bölümleme verileri
- Haber makaleleri kümeleme
- Dil kümeleme
- Türlerin kümelenmesi
- Anomali tespiti

Bunun yanısıra K-means algoritması, verilerde açıkça etiketlenmemiş grupları bulmak için kullanılır. Bu, hangi tür grupların var olduğu hakkındaki işletim varsayımlarını doğrulamak veya karmaşık veri setlerinde bilinmeyen grupları tanımladıktan sonra, herhangi bir yeni verinin doğru gruba atanabilmesini sağlamaktadır.

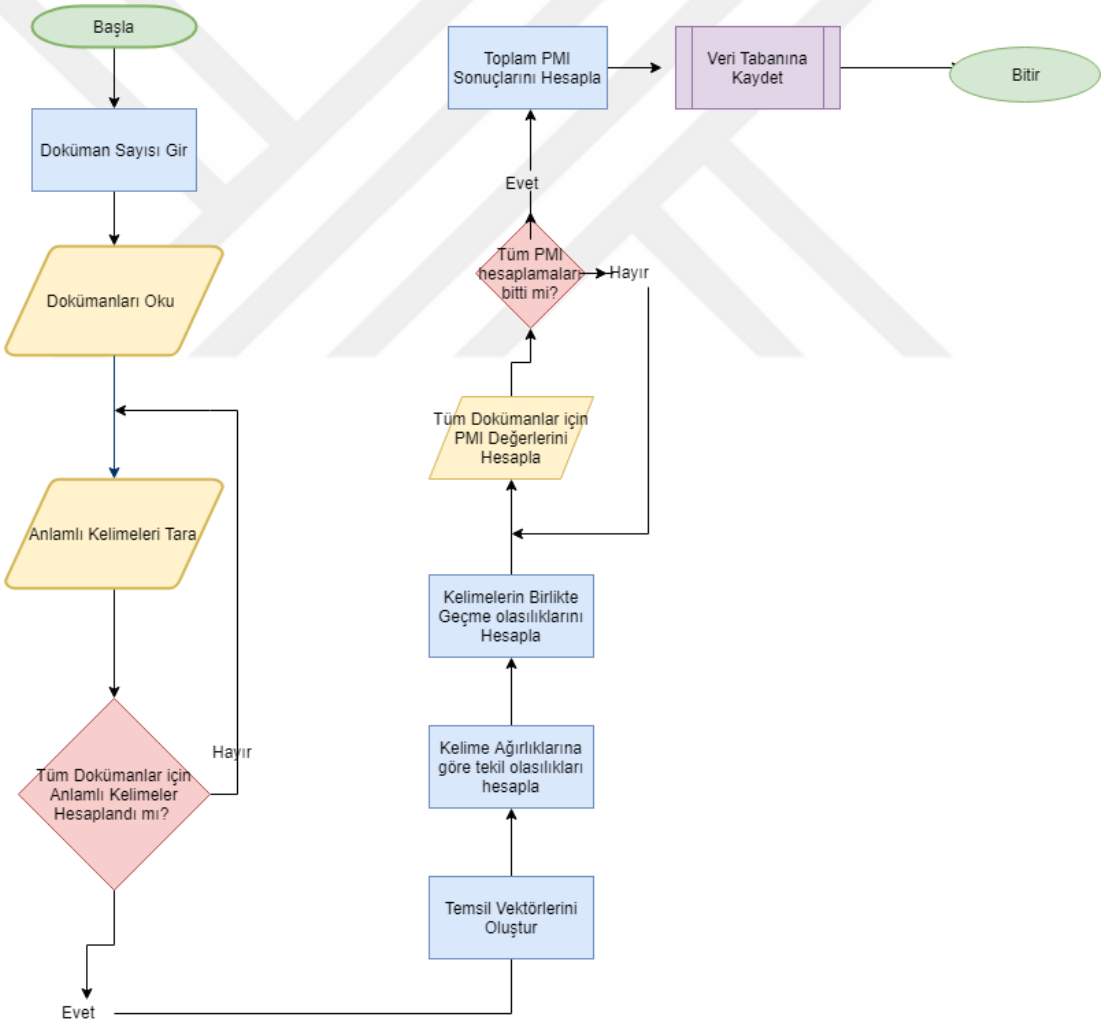
Bu çok yönlü algoritmanın diğer kullanım alanları ise şöyledir:

- Davranışsal Segmentasyon
    - Satın alma geçmişine göre ayırma
    - Uygulama, web sitesi veya platformlardaki etkinliklere göre segmentlere ayırma
    - İlgi alanlarına göre kişileri tanımlama
    - Etkinlik izlemeye dayalı profiller oluşturma
- Envanter Kategorizasyonu
    - Satış etkinliğine göre grup envanter
    - Üretim metriklerine göre grup envanteri
- Sensör ölçümlerini sıralama
    - Hareket sensörlerinde aktivite türlerini tespit etmek
    - Grup görüntüleri
    - Ses ayırma
    - Sağlık taramalarında grup tanımlamak
- Botları veya anomalileri tespit
    - Geçerli etkinlik gruplarını botlardan ayırmak
    - Aykırı değerleri temizlemek

Algoritmanın hesaplama karmaşıklığı gerçek yaşam uygulamaları için uygun olmayabilmektedir.

5. YAZILIMIN UYGULANMASI

Tez çalışmasının bu bölümünde, önceden sınıfları belirlenmiş dokümanların farklı benzerlik ölçütlerine göre benzerliklerin tespiti ve K-Means algoritması kullanılarak öbekleme yapılması için geliştirilen programdan bahsedilecektir. Uygulama Python 3.7.2 64-bit versiyonu ile geliştirilmiştir. Veri tabanı işlemleri için MS SQL Server 2017 sürümü kullanılmıştır. Doğal dil işleme adımlarını gerçekleyen çeşitli açık kaynak kodlu kütüphanelere projeye dâhil edilmiştir. Projedeki tüm hesaplama ve metin işleme işlemleri basit bir komut arayüzünden yapılmaktadır. Geliştirilen yazılıma dair algoritma şeması aşağıda paylaşılmıştır.

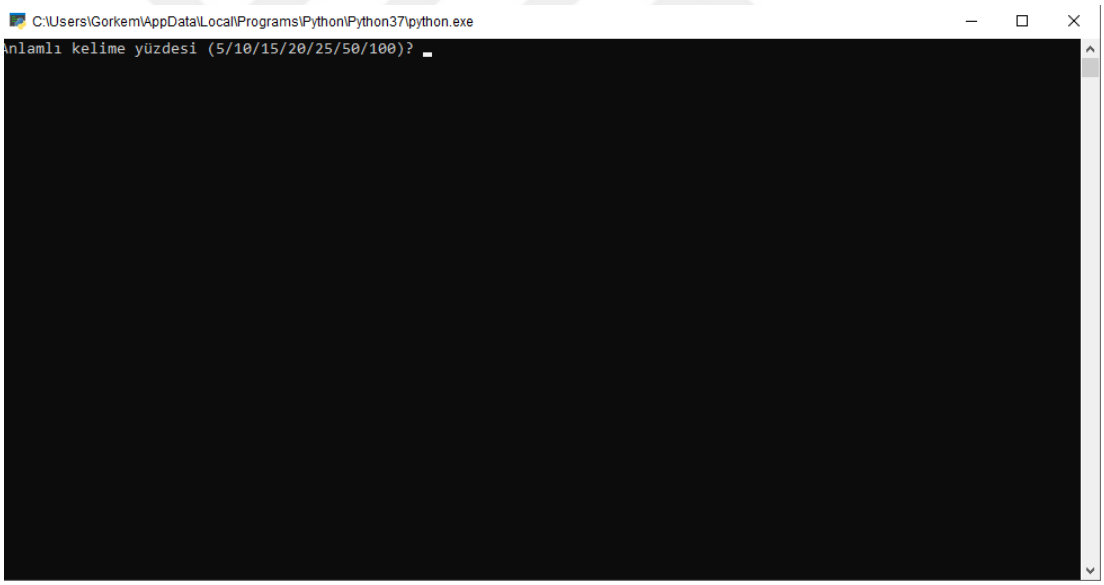


Şekil 5.1 Uygulama Algoritması

Uygulamada kullanılan açık kaynak kodlu Python kütüphanelerinden BeautifulSoap HTML veya XML dosyalarını işlemek için oluşturulmuştur.

Nltk.stem kütüphanesi yaygın olarak kullanılan Porter stemming algoritması (veya "Porter stemmer"), ortak morfolojik ve son eklerin İngilizce'deki kelimelerden çıkarma işlemi için kullanılmıştır. Pypyodbc kütüphanesi SQL server ile bağlantı kurulması ve SQL betiklerinin işletilmesi için kullanılmıştır. Düzenli ifadeler ile işlem yapılabilmesi için “re” kütüphanesi kullanılmıştır. Diziler ve lineer cebir işlemlerini yapabilmek için numpy kütüphanesinden istifade edilmiştir. Veri analizi ve işlenmesi için panda, spacy, sklearn kütüphaneleri kullanılmıştır. Bunun yanı sıra birçok yerleşik python kütüphanesi yazılımın geliştirilmesinde yardımcı araç olarak kullanılmıştır. Doğal dil işleme uygulamalarından yaygın olarak kullanılan ve bilinen NLTK kütüphanesi de ön işleme adımlarında kullanılmıştır.

Program çalıştırıldığında girdi olarak hesaplamalarda kullanılacak anlamlı kelime oranı girdi olarak alınmaktadır.



Şekil 5.2 Program Arayüzü

Bu programda 3 alt program parçacığı yer almaktadır. Bu alt programlardan ilki paragrafların tespiti, paragraflarda bulunan kelimelerin tespiti ve işlenmeye uygun hâle getirilmesi, paragraflarda bulunan terimlerin geçiş sayılarının hesaplanması ve terimlerin ağırlıklandırılması işlemlerini gerçekleştirmektedir (Ögtelik, S. & Turan, M. 2018).

Diğer alt programda Kosinüs, Jaccard ve PMI benzerliği anlamlı kelime ağırlıklarına göre hesaplanmaktadır. Son olarak üçüncü program parçasında, benzerlik ölçütü olarak K-Means ile kullanılabileceğini düşündüğümüz PMI benzerlik ölçütü ile doküman kümelemesi yapılmaya çalışılmıştır.

5.1 Veri Kümesi

Uygulamada sınanmak üzere 140 adet doküman rastgele seçim yöntemi ile programa girdi olarak verilmiştir. Dokümanların sınıfları önceden belirlenmiştir. Eğitim, Spor ana başlıkları altında 2 sınıf altında fiziksel ve mantıksal olarak tutulan kayıtlar üzerinde çalışmalar yapılmıştır. Anlamlı kelimelerin %25, %50 ve %100'ünü kullanarak veri kümesinde öbekleme algoritması çalıştırılmıştır. Uygulama için kullanılan veri setine dair liste Çizelge 5.1'de aşağıdaki gibi listelenmiştir (Ögtelik ve Turan, 2018).

Çizelge 5.1: Uygulamada kullanılan doküman listesi

ID	Doküman Adı	Konu	Alt Konu
0	Education_Elearning_7 good reasons why eLearning is so important for your organization.txt	Education	Elearning
1	Education_Elearning_Definition of 'E-learning'.txt	Education	Elearning
2	Education_Elearning_Definition of e-Learning.txt	Education	Elearning
3	Education_ElemantrySchool_Early schooling matters most for children.txt	Education	ElemantrySchool
4	Education_Homeschool_Education's biggest trend Why home taught kids are doing better.txt	Education	Homeschool
5	Education_Language_Importance of Language – Why Learning a Second Language is Important.txt	Education	Language
6	Education_Mathematics_The UK needs a revolution in the way maths is taught. Here's why….txt	Education	Mathematics

7	Education_Preschool_Preschool education boosts children's academic success, research finds .txt	Education	Preschool
8	Education_Scholarship_Institutional Repositories Essential Infrastructure For Scholarship In The Digital Age.txt	Education	Scholarship
9	Education_Scholarship_Professionalising Teaching Practice in Higher Education A study of disciplinary variation and teaching scholarship.txt	Education	Scholarship
10	Sports_Badminton_Badminton Basics For Beginners.txt	Sports	Badminton
11	Sports_Badminton_Badminton Rules (doubles and singles).txt	Sports	Badminton
12	Sports_Badminton_Physical Treatments by Playing Tennis.txt	Sports	Badminton
13	Sports_Bicycling_The benefits of cycling for children and families.txt	Sports	Bicycling
14	Sports_Football_Football Association still to commission scientific research into links between dementia and football.txt	Sports	Football
15	Sports_Football_Welsh FA will appeal FIFA fine over poppy tributes against Serbia.txt	Sports	Football
16	Sports_Gymnastics_The Effect Of Gymnastics On Height And Bone Density.txt	Sports	Gymnastics
17	Sports_Swimming_The Benefits of Long, Unbroken Swimming.txt	Sports	Swimming
18	Sports_Tennis_Basic rules of tennis.txt	Sports	Tennis
19	Sports_Tennis_origin and early years.txt	Sports	Tennis

5.2 PMI Sonuçlarının hesaplanması

Dokümanlara ait anlamlı kelimeler tespit edilip veri tabanına kayıt edildikten sonra PMI hesaplama adımlarına geçilir. Anlamlı kelimelerin bir veri kümesi içerisinde tekrarlanmasından ziyade frekansları yok ederek sadece doküman içerisinde geçip geçmediğinin tespiti ile bir hesaplama yapılmak istendiği için her bir doküman için ikili sistem vektör dizileri oluşturulmuştur. Vektörler oluşturulurken anlamlı kelimeler teker teker dokümanlar içerisinde taranıp doğrusal bir vektör oluşturulduğu için kelimelerin veri tabanında tutulan indeks'lerinin birebir aynı olması gerekmektedir. Bu indeks sıralaması bozulduğu takdirde vektör üzerinde tutulan ikili sistem indis'lerde kayma yaşanacağı için hatalı sonuçlar üretilecektir. Her bir doküman için oluşturulan vektörler birliktelik hesaplamalarında kullanılmıştır.

Doküman seti için anlamlı olarak değerlendirilen kelimelerin, doküman kümesi içerisinde geçme olasılıkları:

$$P(x) = \sum V(x) / D \qquad (5.1)$$

Formülüne göre hesaplanmaktadır. Bu adımda her bir anlamlı kelime çifti için birlikte bulunma olasılıkları hesaplanmaktadır. Bu değerler veri tabanına kaydedilmiş ve PMI formülünün payı olarak kullanılmıştır. Oluşturulan doküman vektörleri taranarak iki dokümanının indeksi için true(doğru) olan indislerde birlikte geçme sayısı bir arttırılarak formülün payı hesaplanmıştır. Payda ise doküman sayısı olarak belirlenmiştir.

Birlikte bulunma olasılıklarının hesaplanması için kullanılan formül:

$$P(x, y) = a / D \qquad (5.2)$$ şeklindedir.

Formülde a değişkeni birliktelik değişkeni yani seçilen iki anlamlı kelimenin herhangi bir dokümanda bir arada kullanılma sayısı, olarak kullanılmaktadır. Doküman kümesi içinde yer alan anlamlı kelimelerin, dokümanlar içerisinde geçme durumları ve de birlikte geçme olasılıkları hesaplanıp veri tabanına kayıt edildikten sonra her bir kelime çifti için PMI hesapları yapılmıştır. PMI hesaplarını yapan yordam kullanılan PMI formülü aşağıdaki gibidir:

$$pmi(x, y) = \log_2 P(x, y) / P(x)P(y) \qquad (5.3)$$

Tüm dokümanlar için PMI değerleri hesaplandıktan sonra aşağıdaki benzerlik formülü ile doküman ikilileri için benzerlik sonuçları hesaplanmaktadır.

$$\text{Sim (Di, Dj)} = \sum_{i=1}^{k} \sum_{j=1}^{l} PMI(w_i, w_j) \quad (5.4)$$

Bulunan benzerlik sonuçları doküman çiftlerinin tekil değerleri, önceden belirlenmiş sınıfları ve benzerlik sonucu ile birlikte veri tabanında kayıt altına alınmaktadır.

5.3. Kosinüs Benzerliği Yöntemi ile Doküman Benzerliklerinin Hesaplanması

Doküman vektörlerinin hesaplanabilmesi için sklearn.feature extraction.text python kütüphanesinden istifade edilmiştir. Kodlanan yordama, doküman kümesinde yer alan dokümanlar birer çift halinde parametre olarak eklenmiş ve kosinüs benzerliğine göre sonuçlar alınıp, veri tabanına kayıt edilmiştir.

5.4 Jaccard Katsayısı Yöntemi ile Doküman Benzerliklerinin Hesaplanması

Her bir doküman çifti Jaccard katsayısını hesaplayan yordama parametre olarak eklenerek bulunan benzerlik sonuçları veri tabanında kayıt altına alınmıştır. Jaccard Katsayı formülünü programatik olarak ifade eden yordam her bir doküman çifti için çağrılarak hesaplanır.

5.5 Benzerlik Sonuçlarının Değerlendirilmesi

Veri tabanına kayıt edilen benzerlik sonuçları tablolarından, seçilen metriklere göre sorgular çalıştırılmıştır. Bu metrikler, benzerlik ortalamalarının üzerindeki dokümanların sınıflandırmasının değerlendirilmesi, Doğruluk değerleri ve Her bir dokümanın benzerlik katsayılarının ortalamaları baz alınarak sınıflandırmaların değerlendirilmesi şeklindedir.

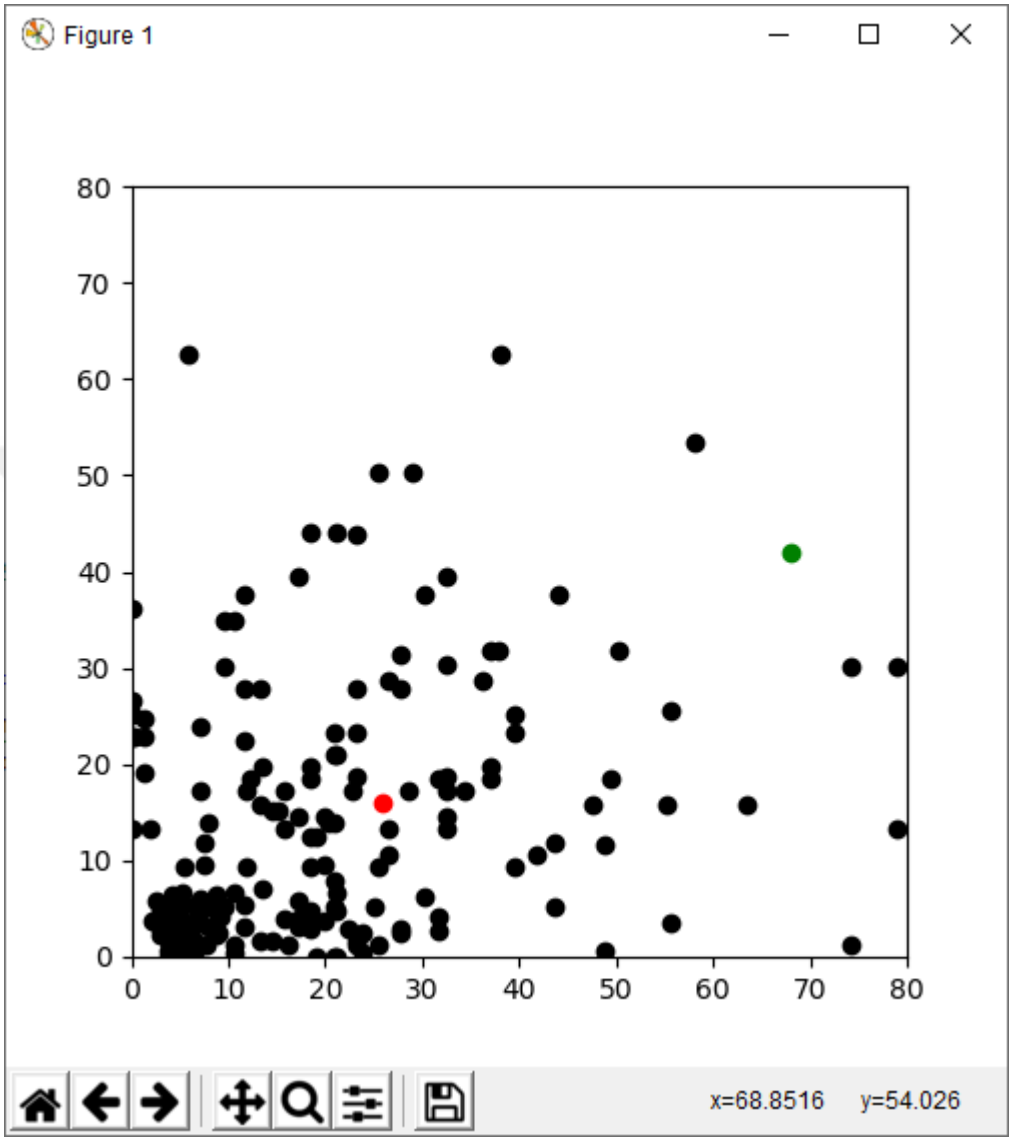
Her bir dokümanın benzerlik katsayılarının ortalamaları ile hesaplamalar yapılırken, anlamlı kelimelerin ağırlık oranı değiştirilmiş ve sonuçlar irdelenmiştir.

Bu sonuçlar irdelendiğinde PMI benzerlik ölçütü anlamlı kelime dağılımlarına adaptif bir yaklaşım sergiler iken Kosinüs benzerlik ölçütünde ve Jaccard benzerliğinde herhangi bir iyileşme gözlemlenmemiştir.

5.6 K-Means Kümeleme Algoritmasının PMI Yöntemiyle Gerçeklenmesi

PMI benzerlik ölçütü ile yapılan çalışmadan üretilen bulgular ile K-means algoritmasında kullanılmak üzere bir adet sonuç verisi dokümanı hazırlanmıştır.

Sonuçlar bir uzaklık vektörü olarak sisteme verilmiş ve başlangıç centroid değeri 2 seçilmiştir. K-means algoritması işletildikten sonra sonuçların dağılımı Şekil 5.3'deki gibi olmuştur.



Şekil 5.3: K-means kümeleme sonuçları

Çalışmanın devamında PMI benzerlik ölçütünü de kullanarak K-Means algoritmasının gerçeklenmesi üzerine çalışılmıştır.

K ortalama küme algoritması, veri madenciliğinde küme analizi için en popüler makine öğrenmesi algoritmalarından biridir. K ortalama kümelemesi, gözlemleri, kümelerin bir prototipi olarak belirtilen, her iterasyonun kümeye en yakın ortalamaya ait olduğu k kümelerine bölünmesini amaçlamaktadır.

K ortalama küme algoritması denetimsiz bir öğrenme algoritmasıdır. Yani eğitim verisine ihtiyacı yoktur, gerçek veri seti üzerinde hesaplamalar yapılmaktadır. Bu algoritma, sınıflandırma için popüler bir makine öğrenme tekniği olan en yakın komşu sınıflandırıcı ile benzerlikler içermektedir.
K-means algoritması iterasyon tabanlıdır, küme centroidlerini tekrar tekrar hesaplar ve sonuçlar değişmeyene kadar değerleri iyileştirir.

K-means algoritması, kaç tane küme oluşturulacağını belirten bir tamsayı parametresiyle birlikte bir veri kümesini alır. Çıktı olarak ise, bir 'K' küme centroid kümesidir ve her veri noktasını benzersiz bir kümeye eşleyen veri kümesini üretir.
Algoritmanın gerçeklenmesi için ilk adım olarak veri noktaları için K küme merkezleri belirlenmektedir. Örneğin, x_1, x_2, x_3, ..., x_n merkezleri belirlenmiş olsun.

$X= x_1, x_2, x_3, ..., x_n$ (5.5)

X seçilen tüm merkez noktalarını ifade etmektedir.

Sonraki adım olarak, girdi olarak verilen veri noktalarını en yakın merkez ile ilişkilendirmemiz gerekmektedir. Bunun için verilen veri noktası ile her bir merkez noktasına olan uzaklık hesaplanır.

$$\boldsymbol{arg\ \min_{xi \in X} uzaklık(Xi, X)^2} \quad (5.6)$$

Bu adımda, kümeye atanan tüm veri noktalarının ortalamasını alarak yeni küme merkezleri bulunmalıdır.

Bu uygulamada K-means kümeleme algoritmasında, Öklid uzaklık hesaplaması için aşağıdaki formül kullanılmıştır:

$$\text{Uzaklık (i)} = \sqrt{\sum_{x=1}^{p} (\boldsymbol{xi} - \boldsymbol{C})^2} \quad (5.7)$$

C değişkeni Centroid'i, Xi değişkeni Öklid uzaklığı hesaplanacak veri noktasını temsil etmektedir.

Bu adımlar atanabilecek küme kalmayana kadar devam eder ve kümeleme işlemi tamamlanır.

Bu çalışmada uzaklık hesaplamasında Öklid uzaklık ölçütü yerine, dokümanlara ait PMI toplamlarını benzerlik ölçütü olarak kullandık ve sonuçları kıyasladık.

5.7 PMI Yönteminin Bir Benzerlik Ölçütü Olarak Öbekleme İçin Kullanılması

Bu bölümde sınıfları önceden sistem tarafından bilinmeyen dokümanların PMI yöntemini bir benzerlik ölçütü olarak öbeklenmesine çalışılmıştır. Bunun için 10 eğitim ve 10 spor olmak üzere 20 doküman kullanılmıştır.

Dokümanlar içerisinden rastgele seçilen iki tanesi C1 ve C2 kümelerine konumlandırılmıştır. Bu aşamada her bir sınıf için eklenen bu dokümanların kelime vektörleri de o anki merkez vektörü olarak kabul edilmiştir. Sonrasında yeni bir rastgele doküman seçilerek bu dokümanın hangi sınıfa dahi olması gerektiğine karar veren algoritma çalıştırılmıştır. Buna göre, seçilen dokümana ait kelime vektörü ile o anki merkez vektörü arasındaki PMI değerleri kümülatif olarak hesaplanmıştır. C1 ve C2 sınıfına ait merkez vektörlerinden hangisinin PMI değeri büyük hesaplanıyor ise yeni seçilen doküman o sınıfa dahil edilerek merkez vektörü güncellenmiştir.

Sisteme girilen tüm dokümanlar yukarıda anlatılan yöntem ile hesaplamalara tutularak C1 ve C2 sınıflarına dağıtılmış ve sonuçlar gözlemlenmiştir.

Örneğin; Elimizde 5 dokümanlık bir doküman seti bulunsun ve bu dokümanlara ait doküman vektörleri, içerdikleri anlamlı kelimelere göre aşağıdaki gibi oluşturulmuş olsun.

D1(1,1,0,0,0)

D2(0,0,1,1,1)

D3(1,1,0,1,1)

D4(0,0,0,1,1)

D5(1,1,1,0,1)

Program çalıştırıldığında iki adet vektör rastgele seçilir ve ön tanımlı sınıf değişkenlere atanır. Program çalıştırıldığında iki adet vektör rastgele seçilir ve ön tanımlı sınıf değişkenlere atanır. Bu durumda Şekil 5.4 ‘te şema olarak ifade

edildiği gibi her bir sınıf için oluşan merkez vektörleri de o kümeye atanan doküman vektörü ile aynı vektör olmaktadır.

Şekil 5.4: Rastgele seçim ile doküman öbekleme

Sonraki adımda iteratif olarak yeni bir doküman ele alınır. Ele alınan doküman hem sınıf 1 için hem de sınıf 2 için PMI benzerlik hesaplamasına uygulanır. PMI benzerlik ölçütü hangi sınıf için daha büyük hesaplanmışsa, doküman bu sınıfa atanır ve merkez vektörü güncellenir.

Örneğin, D1 dokümanın Sınıf1 için hesaplanan PMI değeri 30,2, Sınıf2 için hesaplanan PMI değeri ise 11,4 olsun. Bu durumda D1 dokümanı Sınıf1 içine atanır.

Bu aşamada PMI değerleri şu şekilde hesaplanır.

D1(1,1,0,1,1)

D2(1,0,0,0,1)

Şeklindeki iki vektörü ele alacak olursak. $D_t = 2$, V=5 D_t Toplam doküman sayısını ifade eder. V ise anlamlı kelime sayılarını ifade etmektedir. Her bir doküman için öncelikle anlamlı kelimelerin dokümanlardaki görülme olasılıkları hesaplanır.

D1(0) = 1/5, D1(1) = 1/5, D1(2) = 0, D1(3) = 1/5, D1(4) = 1/5

D2(0) = 1/5, D2(1) = 0, D2(2) =0, D2(3) =0, D2(4) = 1/5

Daha sonra dokümanlarda yer alan kelimelerin birlikte görülme olasılıkları hesaplanır. D1(0,1) = 1, D1(0,2) = ½, D1(0,3) = ½ ... şeklinde tüm kelime çiftleri için olasılıklar hesaplanır.

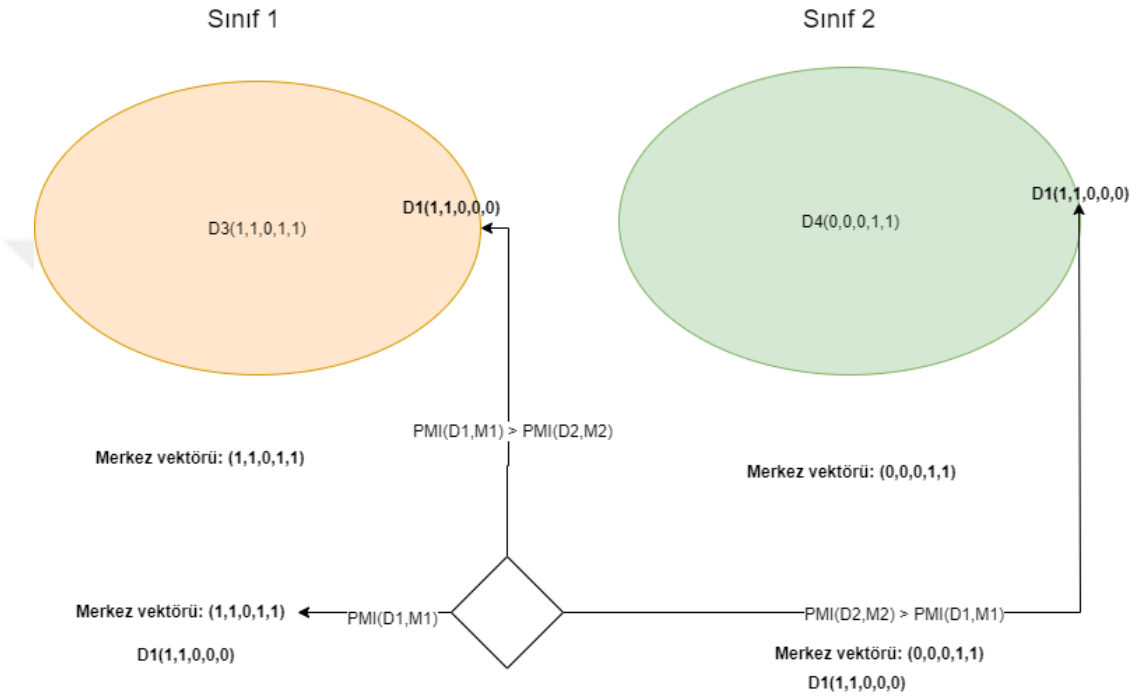
Sonrasında bulunan bu değerler PMI formülüne uygulanmaktadır.

PMI = $\log_2$ (P(D1(0)) * P(D1(1)) / P(d1(o) * d1(1))

Bu durumda yukarıdaki kelime çifti için D1 dokümanı için PMI değeri = 4,64 olarak hesaplanır. Temsil vektörlerindeki tüm kelime çiftleri için bu PMI değerleri kümülatif olarak hesaplanır. Hesaplama yapılan doküman çiftleri için

Sim (Di, Dj) =$\sum_{i=1}^{k} \sum_{j=1}^{l} PMI(w_i, w_j)$ (5.8)

5.8 formülünde verilen toplam formülüyle bir PMI toplamı elde edilmektedir.



Şekil 5.5 Merkez vektörü ile D_n PMI karşılaştırması

Tüm dokümanlar bu işlemlerden geçirilerek uygun sınıfa atanması amaçlanmıştır.

Anlamlı kelimelerin öbekleme üzerindeki etkisinin artıp atmadığını kontrol edebilmek adına "Eşik değeri" yöntemi uygulanmış olup, eşik değeri doküman vektörleri içerisinde anlamlı kelimenin toplam görülme sayısının, tüm dokümanlara oranıyla elde edilmiştir.

Örneğin, Sınıf1'e atanmış doküman örneklemi aşağıdaki gibi olsun.

D1(1,1,0,0,1)

D2(0,0,0,1,1)

D5(1,0,0,0,1)

D6(1,1,1,1,1)

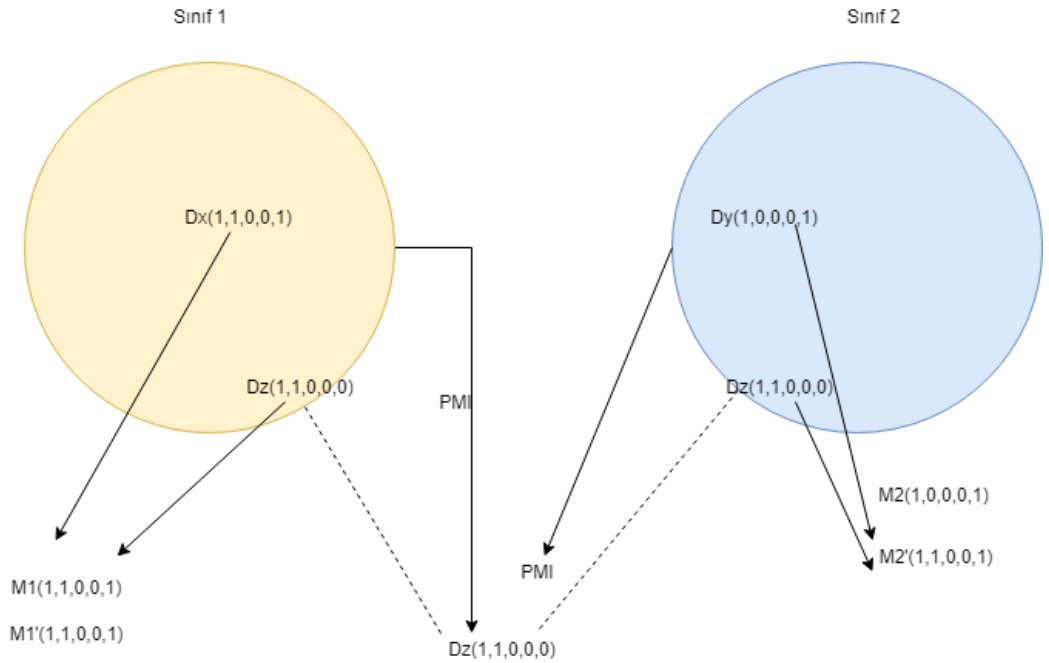
D9(0,0,0,1,1)

Ele alacağımız dokümanın ise D4(0,0,1,0,0) şeklinde olduğunu düşünürsek, bu dokümanda sadece 3. İndiste yer alan kelime için PMI benzerliğini hesaba katabiliriz. Fakat %50 eşik değeri uygulayarak geçmiş dokümanlarda yer alan fakat D4 dokümanında yer almayan 1. 4. ve 5. İndisteki kelimeler için de PMI benzerlik hesabını kullanabiliriz. Böylece merkez vektörü anlamlı kelimeleri daha yoğun olarak içerecek ve sınıflama belirginliği daha doğru sonuçlar verecektir. Bu yöntem uygulandığında oluşan merkez vektörü aşağıdaki gibi olacaktır.

M(1,0,0,1,1)

Oysa ki sadece sınıfa dahil olmuş dokümanların oluşturduğu merkez vektörü hiçbir eşik değeri uygulanmadan hesaplandığında merkez vektörü M(0,0,0,0,1) ve en nihayetinde de ya tamamen 0'lardan ya da tamamen 1'lerden oluşacak bir merkez temsil vektörüne dönüşecek ve anlamlı kelimelerin birbirleriyle olan anlamsal yakınlıkları tespit edilemeyecektir.

Şekil 5.6'da programın öbekleme mantığı şematik olarak verilmiştir. Doküman setinin tüm öğeleri Şekil 5.6'daki hesaplama zincirinden geçtikten sonra tahmin edilen sınıfa atanmaktadır. Bu çalışma ile ilgili yapılan deney sonuçları 6. Bölümde paylaşılmıştır.



Şekil 5.6: PMI Benzerlik Ölçütü ve Eşik Değeri Kullanılmadan Öbeklemenin Şematik Gösterimi

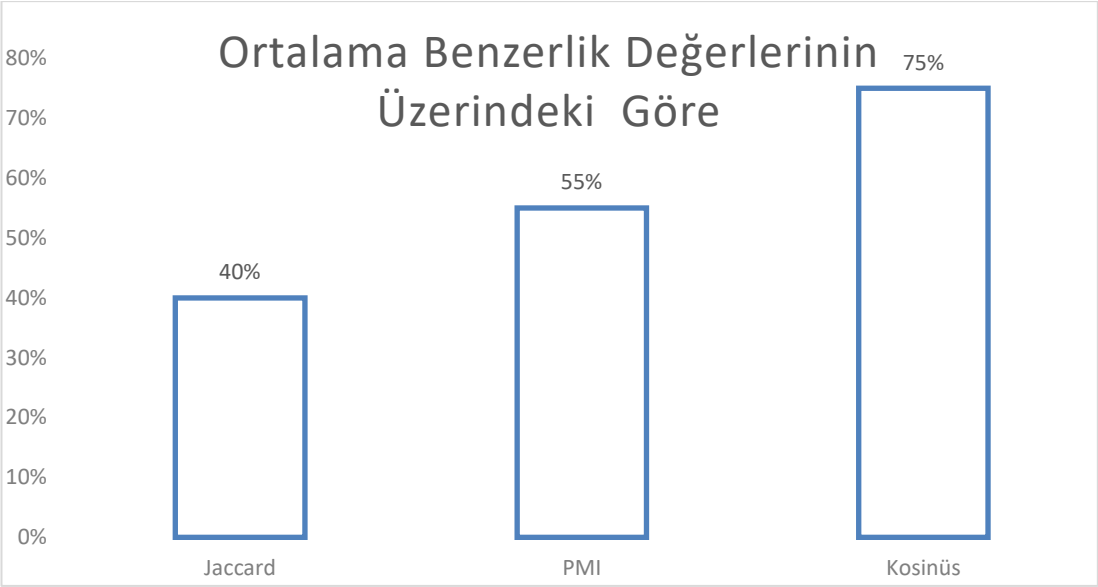
6. SONUÇ VE ÖNERİLER

Bu çalışmada benzerlik ölçütü karşılaştırılması için Kosinüs, Jaccard ve Noktasal Karşılık Bilgi yöntemleri kullanılmış ve sonuçlar değerlendirilmiştir.

Kosinüs benzerliği, metinlerin vektör olarak ifade edildiğinde aralarındaki açıların bir benzerlik ölçütü olabileceğini bir metrik öne sürer. Kosinüs benzerliği, aralarındaki açının kosinüsünü ölçen, iç çarpım uzayının sıfır olmayan iki vektörü arasındaki benzerliğin bir ölçüsüdür. 0 ° 'lik kosinüs 1'dir ve (0, π] radyanlar arasındaki herhangi bir açı için 1'den azdır. Kosinüs benzerliği, sonucun [0,1] arasında net bir şekilde sınırlandığı pozitif alanda özellikle kullanılır. Bu ad, "kosinüs yönü" teriminden türetilir: bu durumda, birim vektörler paralel ise maksimum "benzer" ve dik (dikey) ise maksimum "farklı" olabilirler. Bu, farklar sıfır açıya döndüğü zaman birlik (en yüksek değer) olan ve kosinüse dik olduğunda sıfıra (ilişkisiz) benzerdir.

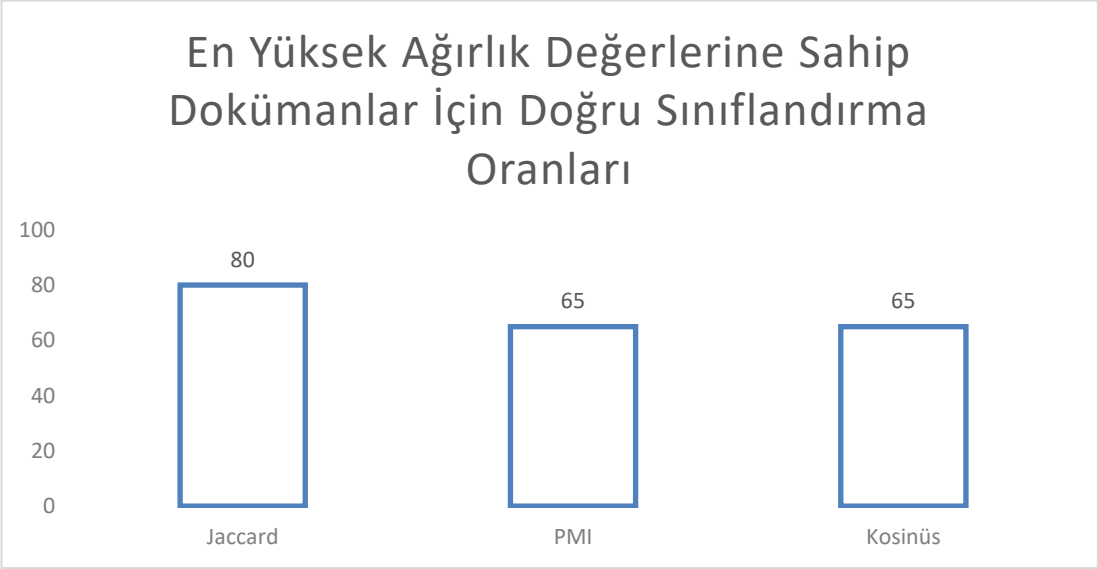
Birlik üzerine kesişim (Intersection Over Union) ve Jaccard Benzerlik Katsayısı olarak da bilinen Jaccard benzerliği, sonlu örnek kümeler arasındaki benzerliği ölçer ve örnek kümelerin birliğinin büyüklüğüne bölünmesiyle kesişim ölçüsü olarak tanımlanır.

Noktasal karşılıklı bilgi (PMI), p (x, y) olaylarının belirli bir eşzamanlı oluşumunun gerçek olasılığının, bireysel olayların olasılıkları P(x), P(y) temelinde olmasını beklediğimizden ne kadar farklı olduğunun bir ölçüsüdür.
Her dokümanın diğer dokümanlarla olan benzerliklerinin ortalaması üzerinde kalanların ait oldukları sınıfların çoğunluğuna göre sınıf tespiti yapıldığında elde edilen metrik değerleri Çizelge 1'de görüleceği gibi Kosinüs benzerliğinde %75, PMI benzerliğinde %55 ve Jaccard benzerliğinde %40 olmuştur. Biz bu metriği ortalama üstü çoğunluk olarak adlandırdık ve bu yaklaşım k komsu öbekleme icin neredeyse benzer biçimde uygulanmaktadır.



Çizelge 6.1: Benzerlik ağırlığının üzerindeki kayıtlar için dağılımlar

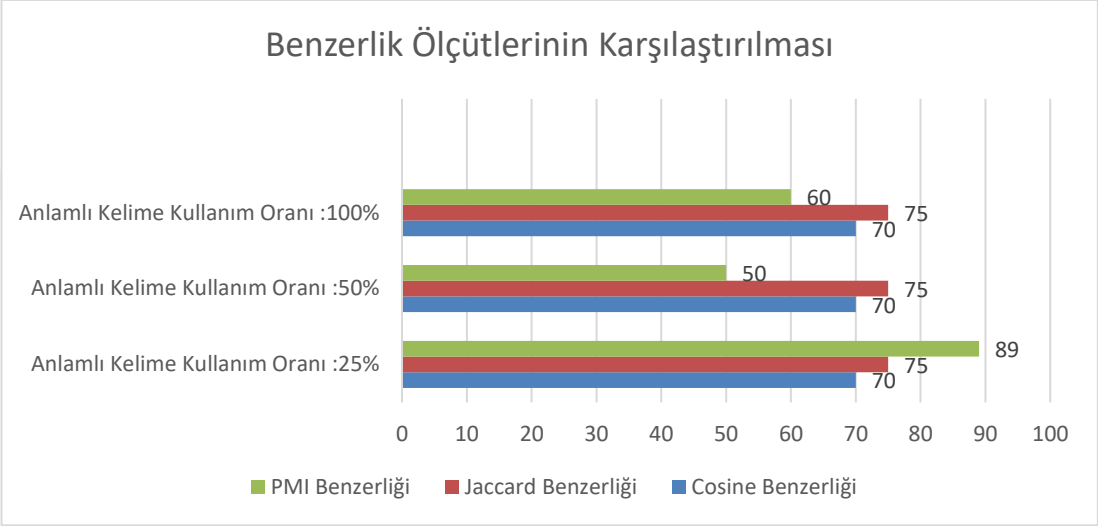
Buna rağmen 20 dokümanın her biri için en yüksek benzerlik değerine sahip kaydın, önceden verilen sınıf değerleri ile örtüşmesi karşılaştırıldığında Çizelge 2'de görülebileceği gibi Kosinüs benzerliği ile yapılan hesaplamalarda %80, Jaccard benzerliği ile yapılan hesaplamalarda %65 ve aynı şekilde PMI benzerliği ile yapılan hesaplamalarda ise %65 doğruluk olduğu tespit edilmiştir. Metrik olarak en yüksek benzerlik değerlerinin doğru olarak tespiti baz alındığında grafik aşağıdaki gibi oluşmaktadır.



Çizelge 6.2: Yüksek ağırlık dokümanlar için doğru sınıflandırma oranları

Bu çalışmada temel olarak metin benzerliği konusunda kullanılan Kosinüs ve Jaccard benzerlik ölçütleri, PMI yaklaşımı ile karşılaştırılmıştır. Sınıfları önceden belirlenmiş dokümanlar üzerinde bu metriklere göre sonuçları değerlendirilmiştir.

PMI, özellikle anahtar kelime seçim oranlarına duyarlı olduğu, bununda taşıdığı bilgi miktarıyla alakalı olduğu görülmektedir. Doğal olarak %25 anlamlı kelime oranında da diğer metriklere göre çok daha iyi sonuçları vermiştir. Anlamlı kelime kullanım oranlarına göre kullanılan 3 yöntemin başarı oranları aşağıdaki Çizelge 3'te sunulmuştur.



Çizelge 6.3: Anlamlı kelimelerin kullanım yüzdelerine göre dağılım

Temel olarak metin benzerliği konusunda kullanılan Kosinüs ve Jaccard benzerlik ölçütleri, PMI yaklaşımı ile karşılaştırılmıştır. Sınıfları önceden belirlenmiş dokümanlar üzerinde bu metriklere göre sonuçları değerlendirilmiştir.

PMI, özellikle anahtar kelime seçim oranlarına duyarlı olduğu, bununda taşıdığı bilgi miktarıyla alakalı olduğu görülmektedir. Doğal olarak %25 anlamlı kelime oranında da diğer metriklere göre çok daha iyi sonuçları vermiştir.

Anlamlı kelime kullanım oranlarına göre kullanılan 3 yöntemin başarı oranları yukarıdaki Grafik 3'te sunulmuştur. Bu durum göz önüne alındığında PMI'ın öbeklemede benzerlik ölçütü olarak kullanımının iyi sonuçlar vereceği gözlemlenmiştir.

	Tahmin Sonucu		
	Olumsuz	Olumlu	Toplam
Kelime Ağırlığı 25%	14	43	57
Kelime Ağırlığı 50%	21	36	57
Kelime Ağırlığı 100%	19	38	57
Toplam	**35**	**79**	**171**

Çizelge 6.4: Tüm tahminler için karmaşıklık matrisi

Kosinüs, Jaccard ve PMI benzerlik ölçütleri ile tüm tahminler sonucu Çizelge 6.4'teki gibi oluşan karmaşıklık matrisine göre, anlamlı kelimelerim kullanım ağırlıklarına göre ayrı ayrı sonuçlar elde edilmiştir. Tüm hesaplamalarda doğruluk oranı %69 olarak gerçekleşmiştir.

	Tahmin Sonucu		
	Olumsuz	Olumlu	Toplam
Kelime Ağırlığı 25%	3	16	19
Kelime Ağırlığı 50%	10	9	19
Kelime Ağırlığı 100%	8	11	19
Toplam	**21**	**36**	**57**

Çizelge 6.5: PMI benzerlik ölçütü tahminleri için karmaşıklık matrisi

Sadece PMI benzerlik ölçütü baz alınarak oluşturulan karmaşıklık matrisi Çizelge 6.5'te verilmiştir. Buna göre anlamlı kelimelerin kullanım ağırlıklarına göre ayrı ayrı hesaplanan benzerlik sonuçlarında doğruluk oranı %63 olarak gerçekleşmiştir.

K-means kümeleme algoritmasının entegrasyonundan sonra iki uzaklık ölçüm yaklaşımı kullanılarak gibi çeşitli metriklere (doğruluk ve hassasiyet) göre kümeleme sonuçları gözlemlenmiştir.

20 doküman ile yapılan çalışmada sonuçlar aşağıdaki gibi oluşmuştur.

Gerçek Sınıf	K-Means Öklid UzaklıkYöntemi	K-Means PMI yöntemi
Education	40%	90%
Sports	50%	20%
Toplam	**45%**	**55%**

Çizelge 6.6: Kümeleme sonuçların doğruluk yüzdeleri (20 Doküman)

40 doküman ile yapılan çalışmada sonuçlar aşağıdaki gibi oluşmuştur.

Gerçek Sınıf	K-Means Öklid UzaklıkYöntemi	K-Means PMI yöntemi
Education	50%	95%
Sports	55%	15%
Toplam	**52%**	**55%**

Çizelge 6.7: Kümeleme sonuçların doğruluk yüzdeleri (40 Doküman)

Seçilen 20 doküman için Öklid uzaklığı kullanılarak yapılan kümeleme işlemi sonuçların karmaşıklık matrisi aşağıdaki gibi oluşmuştur.

	Tahmin Sonucu		
	Olumsuz	Olumlu	Toplam
Kelime Ağırlığı 25%	10	10	20
Kelime Ağırlığı 50%	12	8	20
Kelime Ağırlığı 100%	11	9	20
Toplam	**33**	**27**	**60**

Çizelge 6.8: Öklid Uzaklık Uzaklığı ile K-means Kümeleme (20 Doküman)

Seçilen 20 doküman için PMI değerleri kullanılarak yapılan kümeleme işlemi sonuçların karmaşıklık matrisi aşağıdaki gibi oluşmuştur.

	Tahmin Sonucu		
	Olumsuz	Olumlu	Toplam
Kelime Ağırlığı 25%	10	10	20
Kelime Ağırlığı 50%	9	11	20
Kelime Ağırlığı 100%	9	11	20
Toplam	**28**	**32**	**60**

Çizelge 6.9: PMI Değerler kullanan K-means Kümeleme (20 Doküman)

Seçilen 40 doküman için Öklid uzaklığı kullanılarak yapılan kümeleme işlemi sonuçların karmaşıklık matrisi aşağıdaki gibi oluşmuştur.

	Tahmin Sonucu		
	Olumsuz	Olumlu	Toplam
Kelime Ağırlığı 25%	24	16	40
Kelime Ağırlığı 50%	18	22	40
Kelime Ağırlığı 100%	19	21	40
Toplam	**61**	**59**	**120**

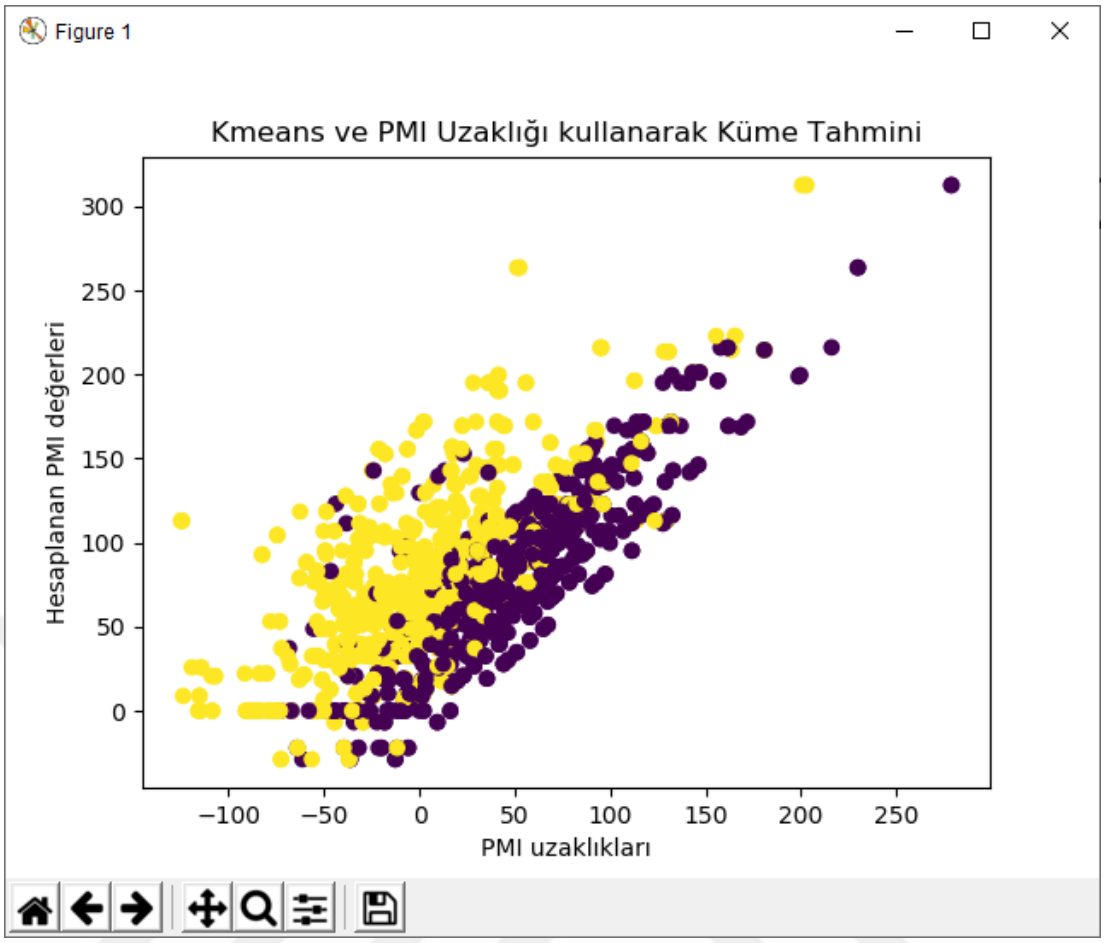
Çizelge 6.10: Öklid Uzaklık Uzaklığı ile K-means Kümeleme (40 Doküman)

Seçilen 40 doküman için PMI değerleri kullanılarak yapılan kümeleme işlemi sonuçların karmaşıklık matrisi aşağıdaki gibi oluşmuştur.

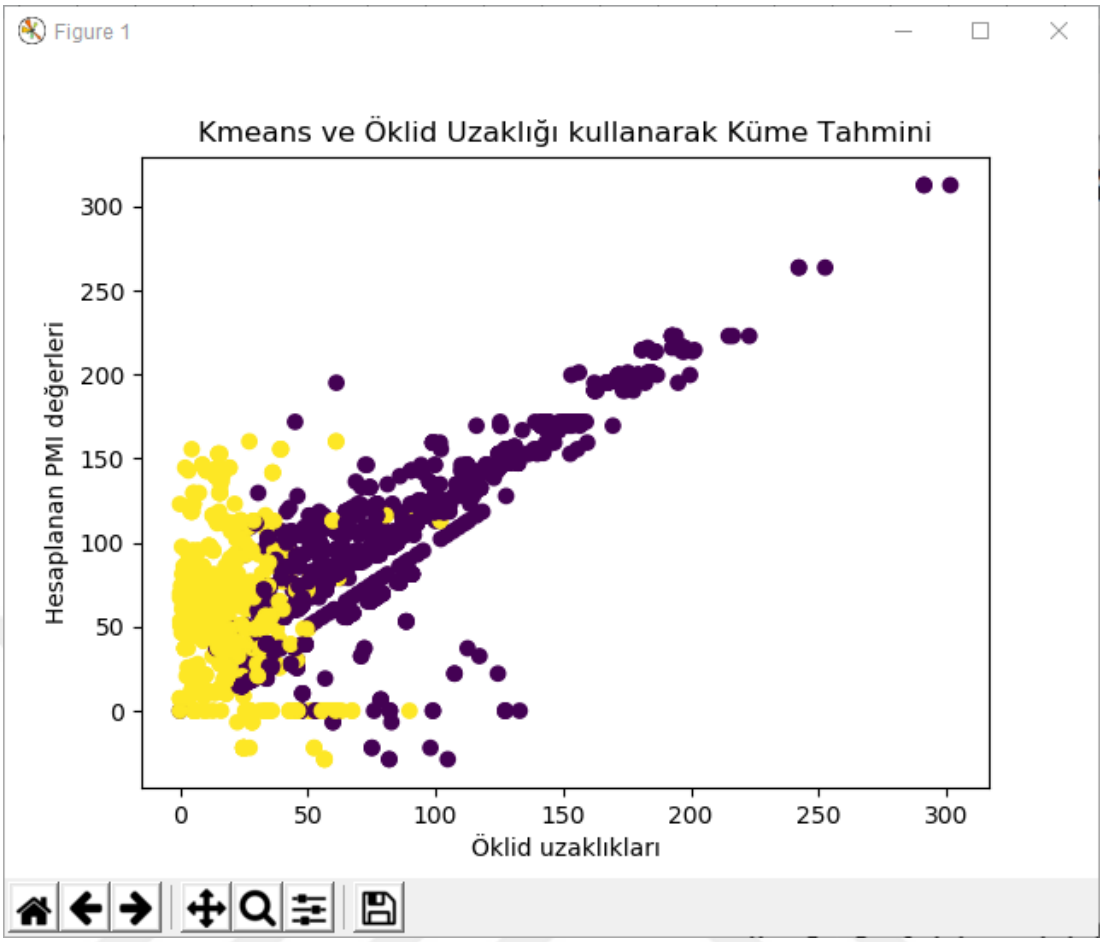
	Tahmin Sonucu		
	Olumsuz	Olumlu	Toplam
Kelime Ağırlığı 25%	18	22	40
Kelime Ağırlığı 50%	21	19	40
Kelime Ağırlığı 100%	18	22	40
Toplam	**57**	**63**	**120**

Çizelge 6.11: PMI Değerler kullanan K-means Kümeleme (40 Doküman)

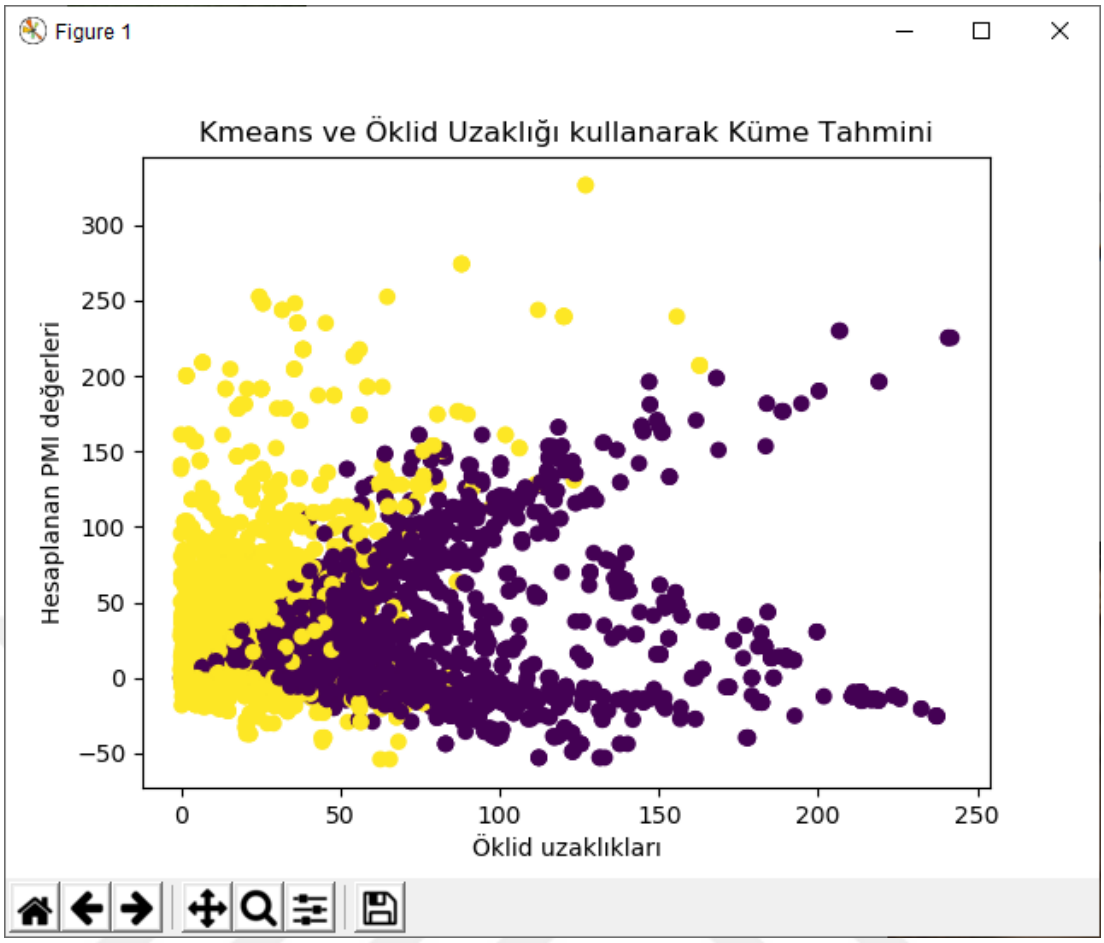
20 doküman ve anlamlı kelimelerin tamamı kullanılarak K-means algoritması öklid ve PMI uzaklık ölçütleri kullanılarak, program çalışma anında seçilen 2 küme ve 100 iterasyon için kümeleme algoritmaları çalıştırılmış ve Python grafik kütüphaneleri kullanılarak çizimler yapılmıştır. Plot grafikleri paylaşılmıştır.



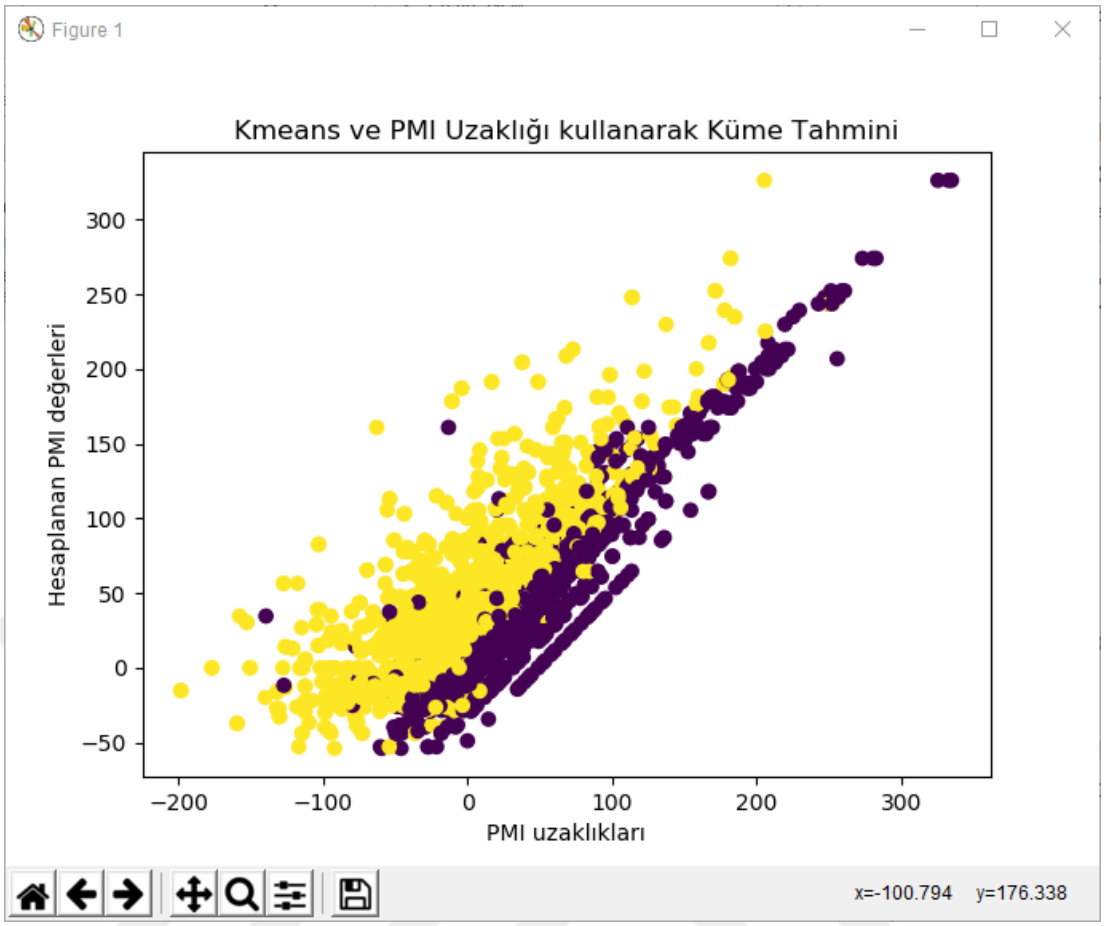
Çizelge 6.12: PMI Değerler kullanan K-means Kümeleme Grafiği (20 Doküman)



Çizelge 6.13: Öklid Değerler kullanan K-means Kümeleme Grafiği (20 Doküman)



Çizelge 6.14: Öklid kullanan K-means Kümeleme Grafiği (40 Doküman)



Çizelge 6.15: PMI kullanan K-means Kümeleme Grafiği (40 Doküman)

PMI kullanarak doküman öbeklemesi yapabilmek için bir başka yaklaşım olarak da doküman vektörlerinin birbirlerine olan uzaklıklarını K-means yaklaşımına benzer adımlar ile fakat PMI hesaplamaları yapmak olmuştur.

Bunu yapabilmek için doküman kümelerinden her biri farklı sınıfta olacak şekilde dokümanlar sınıflara atanmış ve bu doküman vektörleri merkez vektörü olarak kabul edilmiştir. Aynı konuya sahip dokümanların seçilmemesinin nedeni başlangıçta benzer konulara sahip iki doküman seçildiği takdirde ele alınan doküman hesaplamalarında doğru sonuçları elde edilememiş olmasıdır. Farklı konulara ait olduğu bilinen her bir yeni doküman için merkez vektörü ile arasındaki PMI değerleri bulunmuş ve karşılaştırma sonucu daha büyük PMI değerine sahip olan sınıfa ele alınan doküman atanmıştır. Atanan bu sınıftaki ağırlık merkez vektörü değiştiğinden ötürü merkez vektörü güncellenmiştir. Her bir adımda ele alınan bir diğer doküman aynı fonksiyonellik kullanılarak PMI hesaplaması yapılmış ve uygun olduğu ön görülen sınıfa atanmıştır.

Öncelikle örneklem olarak seçilen 20 adet doküman (10 Eğitim, 10 Spor) %25 ila %75 değerleri arasındaki eşik değerine göre çalıştırılmış ve sonuçları

gözlemlenmiştir. Buna göre %60 eşik değeri ile yapılan hesaplamada dokümanlar belirlenen sınıflara eşit dağılmış ve doğru sınıflama oranı %70 olarak gerçekleşmiştir. Bu sonuçları veren eşik değeri aralığı %60'tır.

Sonrasında sisteme farklı bir doküman sınıfında test verileri ilave edilmiş ve hesaplamalar 30 doküman ve 3 sınıf olacak şekilde tekrar gözlemlenmiştir. Buna göre 10 Doğa, 10 Eğitim ve 10 spor kategorisinde olmak üzere yapılan hesaplamalarda eşik değeri olarak %63 belirlendiğinde en yüksek doğrulukla sınıflamanın yapıldığı gözlemlenmiştir. Bu 3 dokümanın genel sınıflandırılma başarısı %70,26 olarak gerçekleşmiştir.

Her biri farklı sınıflarda olmak kaydıyla ve %25 ile %75 arasında değişen eşik değerleri kullanılarak yapılan hesaplamalara dair sonuçlar Çizelge 6.16'da paylaşılmıştır.

Eşik Değeri	C1' Atanan Doküman Sayısı	C2 'ye atanan doküman sayısı	Doğru Hesaplanan Class1 doküman yüzdesi	Doğru Hesaplanan Class2 doküman yüzdesi
25	1	19	100%	47%
26	1	19	100%	47%
27	1	19	100%	47%
28	19	1	47%	100%
29	19	1	47%	100%
30	15	5	46%	40%
31	18	2	100%	55%
32	19	1	47%	100%
33	19	1	47%	100%
34	1	19	100%	47%
35	19	1	47%	100%
36	19	1	47%	100%
37	13	7	53%	57%
38	19	1	47%	100%
39	1	19	100%	47%
40	1	19	47%	100%
41	2	18	100%	55%
42	15	5	46%	40%
43	19	1	47%	100%
44	11	9	54%	55%
45	7	13	71%	61%
46	8	12	50%	50%
47	7	13	42%	46%
48	12	8	58%	62%
49	6	14	66%	42%
50	14	6	71%	100%
51	10	10	60%	60%
52	12	8	50%	50%
53	7	13	100%	76%
54	6	14	66%	57%
55	7	13	57%	54%
56	12	8	50%	50%
57	7	13	57%	54%
58	8	12	50%	50%
59	1	19	100%	47%
60	10	10	70%	70%
61	10	10	50%	50%
62	11	9	72%	77%
63	8	12	50%	50%
64	12	8	58%	63%
65	11	9	54%	55%
66	8	12	50%	50%
67	7	13	71%	38%
68	10	10	50%	50%
69	10	10	50%	50%
70	9	11	55%	54%
71	19	1	47%	100%
72	8	12	37%	41%
73	1	19	100%	47%
74	13	7	61%	71%
75	9	11	44%	45%

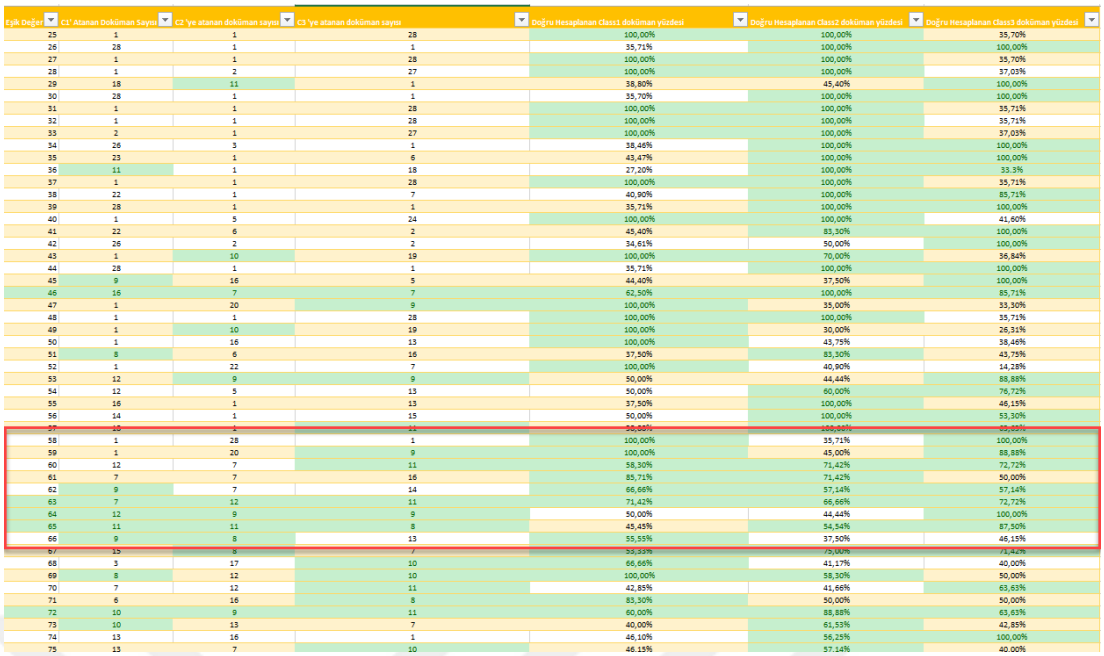
Çizelge 6.16: İki farklı sınıfa dair öbekleme sonuçları

Her biri tamamıyla rastgele sınıflarda olmak kaydıyla ve %25 ile %75 arasında değişen eşik değerleri kullanılarak yapılan hesaplamalara dair sonuçlar Çizelge 6.17'da paylaşılmıştır.

Eşik Değeri	C1' Atanan Doküman Sayısı	C2 'ye atanan doküman sayısı	Doğru Hesaplanan Class1 doküman yüzdesi	Doğru Hesaplanan Class2 doküman yüzdesi
25	19	1	47,37%	100,00%
26	19	1	47,37%	100,00%
27	1	19	100,00%	47,37%
28	1	19	100,00%	47,37%
29	19	1	47,37%	100,00%
30	19	1	47,37%	100,00%
31	1	19	100,00%	47,37%
32	1	19	100,00%	47,37%
33	1	19	100,00%	47,37%
34	1	19	100,00%	47,37%
35	1	19	100,00%	47,37%
36	19	1	47,37%	100,00%
37	19	1	47,37%	100,00%
38	1	19	100,00%	47,37%
39	1	19	100,00%	47,37%
40	1	19	100,00%	47,37%
41	19	1	47,37%	100,00%
42	19	1	47,37%	100,00%
43	19	1	47,37%	100,00%
44	1	19	100,00%	47,37%
45	19	1	47,37%	100,00%
46	1	19	100,00%	47,37%
47	19	1	47,37%	100,00%
48	7	13	57,10%	46,10%
49	1	19	100,00%	47,37%
50	1	19	100,00%	47,37%
51	9	11	66,60%	36,30%
52	6	14	50,00%	50,00%
53	1	19	100,00%	47,37%
54	10	10	60,00%	40,00%
55	11	9	45,40%	55,50%
56	11	9	55,50%	45,40%
57	11	9	45,40%	55,50%
58	10	10	40,00%	60,00%
59	8	12	37,50%	58,30%
60	13	7	46,10%	57,10%
61	12	8	41,60%	62,50%
62	14	6	64,20%	16,60%
63	12	8	58,30%	37,50%
64	10	10	40,00%	60,00%
65	12	8	58,30%	37,50%
66	11	9	54,50%	44,40%
67	7	13	57,10%	46,10%
68	6	14	83,30%	35,70%
69	9	11	44,40%	54,50%
70	9	11	44,40%	54,50%
71	1	19	100,00%	47,37%
72	7	13	42,80%	53,80%
73	1	19	100,00%	47,37%
74	13	7	46,10%	57,10%
75	19	1	47,37%	100,00%

Çizelge 6.17: Rastgele seçilen iki dokümanın öbeklenmesine dair sonuçlar

Her biri 3 farklı sınıfta olmak kaydıyla ve %25 ile %75 arasında değişen eşik değerleri kullanılarak yapılan hesaplamalara dair sonuçlar Çizelge 6.18'da paylaşılmıştır.



Eşik Değeri	C1' Atanan Doküman Sayısı	C2 'ye atanan doküman sayısı	C3 'ye atanan doküman sayısı	Doğru Hesaplanan Class1 doküman yüzdesi	Doğru Hesaplanan Class2 doküman yüzdesi	Doğru Hesaplanan Class3 doküman yüzdesi
25	1	1	28	100,00%	100,00%	35,70%
26	28	1	1	35,71%	100,00%	100,00%
27	1	1	28	100,00%	100,00%	35,70%
28	1	2	27	100,00%	100,00%	37,03%
29	18	11	1	38,80%	45,40%	100,00%
30	28	1	1	35,70%	100,00%	100,00%
31	1	1	28	100,00%	100,00%	35,71%
32	1	1	28	100,00%	100,00%	35,71%
33	2	1	27	100,00%	100,00%	37,03%
34	26	3	1	38,46%	100,00%	100,00%
35	23	1	6	43,47%	100,00%	100,00%
36	11	1	18	27,20%	100,00%	33,3%
37	1	1	28	100,00%	100,00%	35,71%
38	22	1	7	40,90%	100,00%	85,71%
39	28	1	1	35,71%	100,00%	100,00%
40	1	5	24	100,00%	100,00%	41,60%
41	22	6	2	45,40%	83,30%	100,00%
42	26	2	2	34,61%	50,00%	100,00%
43	1	10	19	100,00%	70,00%	36,84%
44	28	1	1	35,71%	100,00%	100,00%
45	9	16	5	44,40%	37,50%	100,00%
46	16	7	7	62,50%	100,00%	85,71%
47	1	20	9	100,00%	55,00%	33,30%
48	1	1	28	100,00%	100,00%	35,71%
49	1	10	19	100,00%	30,00%	26,31%
50	1	16	13	100,00%	43,75%	38,46%
51	8	6	16	37,50%	83,30%	43,75%
52	1	22	7	100,00%	40,90%	14,28%
53	12	9	9	50,00%	44,44%	88,88%
54	12	5	13	50,00%	60,00%	76,72%
55	16	1	13	37,50%	100,00%	46,15%
56	14	1	15	50,00%	100,00%	53,30%
57	15	1	14	53,33%	100,00%	57,14%
58	1	28	1	100,00%	35,71%	100,00%
59	1	20	9	100,00%	45,00%	88,88%
60	12	7	11	58,30%	71,42%	72,72%
61	7	7	16	85,71%	71,42%	50,00%
62	9	7	14	66,66%	57,14%	57,14%
63	7	12	11	71,42%	66,66%	72,72%
64	12	9	9	50,00%	44,44%	100,00%
65	11	11	8	45,45%	54,54%	87,50%
66	9	8	13	55,55%	37,50%	46,15%
67	15	8	7	53,33%	75,00%	71,42%
68	3	17	10	66,66%	41,17%	40,00%
69	8	12	10	100,00%	58,30%	50,00%
70	7	12	11	42,85%	41,66%	63,63%
71	6	16	8	83,30%	50,00%	50,00%
72	10	9	11	60,00%	88,88%	63,63%
73	10	13	7	40,00%	61,53%	42,85%
74	13	16	1	46,10%	56,25%	100,00%
75	13	7	10	46,15%	57,14%	40,00%

Çizelge 6.18: Üç farklı sınıfa dair öbekleme sonuçları

Gelecek çalışmalarda n küme içinde benzer PMI hesaplamaları yapılarak sonuçları gözlemlenmelidir. Mevcut sistem ilk atamalarda kümelere benzer dokümanların atanması durumunda başarılı olamamaktadır. Kümelere başlangıçta farklı türlerden doküman atamanın bir yöntemi geliştirilmelidir. Ayrıca benzer algoritma üzerinde aynı doküman atamalarında Öklid hesaplamaları ile oluşacak kümelerin başarımları karşılaştırılarak daha somut sonuçları elde edilmesi mümkün olacaktır.

KAYNAKLAR

A. Peña-Ayala, 2014, Educational data mining: A survey and a data mining-based analysis of recent works, Expert Systems with Applications, Volume 41, Issue 4, Part 1, Pages 1432-1462, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2013.08.042.

A. Akilan, 2015, "Text mining: Challenges and future directions," 2nd International Conference on Electronics and Communication Systems (ICECS), Coimbatore, pp. 1679-1684. doi: 10.1109/ECS.2015.7124872

A.Kısayol, M. Turan, 2018, Paragraf Tabanlı Çıkarımsal Özetlemede Öbekleme Kullanan İki Yeni Yöntemin Kıyaslanması. Düzce Üniversitesi Bilim ve Teknoloji Dergisi, 6 (4), 1047-1057. DOI: 10.29130/dubited.418453

Balinsky H., Balinsky A., Simske S., 2011. Document sentences as a small world. 2011 IEEE International Conference on Systems Man and Cybernetics (SMC). Anchorage. ABD.

B. Dursun and A. C. Sonmez, "Türkçe metin benzerlik hesaplamasi için yeni bir yöntem," 2008 IEEE 16th Signal Processing, Communication and Applications Conference, Aydin, 2008, pp. 1-4.doi: 10.1109/SIU.2008.4632581

Bhushan S.N. B, Danti A., 2018, "Classification of compressed and uncompressed text documents", Future Generation Computer Systems, Volume 88, 2018, Pages 614-623, ISSN 0167-739X, https://doi.org/10.1016/j.future.2018.04.054.

C. C. Aggarwal, S. C. Gates and P. S. Yu, 2004, "On using partial supervision for text categorization," in IEEE Transactions on Knowledge and Data Engineering,vol. 16, no. 2, pp. 245-255, doi: 10.1109/TKDE.2004.1269601

Dadachev B., Balinsky A., Balinsky H., Simske S., 2012. On the helmholtz principle for data mining. IEEE In Emerging Security Technologies (EST). 2012 Third International Conference, pp 99-102.

G. G. İşgüder-Şahin, H. R. Zafer and E. Adah, "Polarity detection of Turkish comments on technology companies," 2014 International Conference on Asian Language Processing (IALP), Kuching, 2014, pp. 136-139. doi: 10.1109/IALP.2014.6973514

Gover, J.C.,1971, : "Discussion of a paper by R.M. Cormack", J. Roy. Statist. Soc. Ser. A 134, 360-365,

H. Núñez and E. Ramos, 2012, "Automatic classification of academic documents using text mining techniques," XXXVIII Conferencia Latinoamericana En Informatica (CLEI), Medellin, pp. 1-7.doi: 10.1109/CLEI.2012.6427167

I. Dabbura, 2019, https://imaddabbura.github.io/post/kmeans_clustering/

J. Han; M. Kamber, 2006, "Data Mining: Concepts and Techniques ", Morgan Kaufmann Publishers, USA.

J. Hartmann, J. Huppertz, C. Schamp, M. Heitmann, 2019, Comparing automated text classification methods, International Journal of Research in Marketing, Volume 36, Issue 1, Pages 20-38, ISSN 0167-8116, https://doi.org/10.1016/j.ijresmar.2018.09.009.

K. W. Church and H. Patrick (1990). "Word association norms, mutual information, and lexicography". Comput. Linguist. 16 (1): 22–29.

Madylova A, 2009, "Kosinüs Benzerliğini Kullanarak Belgeler Arası Anlamsal Benzerliği Kavramsal Sözlüğe Dayalı Hesaplama Yöntemi", İTU, İstanbul, Türkiye.

Metin Madenciliği.com, Erişim Tarihi:24.05.2019, www.metinmadenciligi.com

M. Işık ve A. Y. Çamurcu, 2010, "K-MEANS VE AŞIRI KÜRESEL C-MEANS ALGORITMALARI İLE BELGE MADENCİLİĞİ", Fen Bilimleri Enstitüsü Dergisi, 22 1-18, Türkiye

M. S. Durmuş, 2005, "Veri Kümeleme Algoritmalarının Performansları Üzerine Karşılaştırmalı Bir Çalışma", Yüksek Lisans Tezi, Pamukkale Üniversitesi, Fen Bilimleri Enstitüsü,

M. Sukanya and S. Biruntha, 2012. "Techniques on text mining," IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), Ramanathapuram, 2012, pp. 269-271. doi: 10.1109/ICACCCT.2012.6320784

Olsson F. , 2009. A Literature Survey of Active Machine Learning in the Context of Natural Language Processing. Technical report, Swedish Institute of Computer Science

Ögtelik, S. & Turan, M. 2018. "İngilizce Dokümanlarda Tema ve Alt Kavramlar Tespit Modeli" Topic and Sub-Topic Detection Model in English Documents Düzce Üniversitesi Bilim ve Teknoloji Dergisi.

Porter, M. F.1980. An algorithm for suffix stripping. Program 14 (3): 130-137.

R. A. Sinoara, J. Camacho-Collados, R. G. Rossi, R. Navigli, Solange O. Rezende, 2019, Knowledge-enhanced document embeddings for text classification, Knowledge-Based Systems, Volume 163, 2019, Pages 955-971, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2018.10.026.

S. M. Weiss, N. Indurkhya, T. Zhang, F. J. Damerau, 2005, Text Mining

Predictive Methods for Analyzing Unstructured Information, Springer-Verlag New York, DOI https://doi.org/10.1007/978-0-387-34555-0

S.Sayad, 2019 , https://www.saedsayad.com/clustering_kmeans.htm

T. Kocatekin and D. Ünay, "Text mining in radiology reports," 2013 21st Signal Processing and Communications Applications Conference (SIU), Haspolat, 2013, pp. 1-4. doi: 10.1109/SIU.2013.6531400

V. Tunalı, 2011, Metin madenciliği için iyileştirilmiş bir kümeleme yapısının tasarımı ve uygulaması", Doktora Tezi, Marmara Üniversitesi, Fen Bilimleri Enstitüsü,

W. C., William & H., Haym. 2000, "Joins that generalize: Text classification using WHIRL." Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining.

Zohar.E, Y. Introduction to Text Mining. Supercomputing, 2002. http://algdocs.ncsa.uiuc.edu/PR-20021008-1.ppt

EKLER

EK A. Kodlar
EK B. Veri tabanı Diyagramı

EK A. Kodlar

```
from bs4 import BeautifulSoup
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from nltk.corpus import wordnet as wn
from nltk.tokenize import word_tokenize
from nltk import pos_tag
import nltk.data
import pypyodbc
import re
import os
import nltk
import string
from string import punctuation
import numpy as np
import itertools
import math
import datetime
import spacy
# Importing the libraries for K-Means
import matplotlib.pyplot as plt
import pandas as pd
# Using the elbow method to find  the optimal number of clusters
from sklearn.cluster import KMeans

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
MeaningPercent = input("Anlamlı kelime yüzdesi (5/10/15/20/25/50/100)? ")

start = datetime.datetime.now()
spacy.load('en_core_web_sm')
stopWords = stopwords.words('english')
DocumentSet = "D:\\Git
Repos\\MultinominalNaiveBayes\\MultinominalNaiveBayes\\SmallWord Eğitim
Verileri"
connection = pypyodbc.connect('Driver={SQL Server};'
                                                'Server=DESKTOP-
TMO951D\GORKEMPC;'

'Database=SmallWordsEducation;'
                                                    'uid=sa;pwd=1')
cursor = connection.cursor()
def ClearDatabase():
    SQLCommand = ("EXEC sp_MSForEachTable 'TRUNCATE TABLE ?'")
    cursor.execute(SQLCommand)
    connection.commit()
#Get Content 1-gram, 2-gram, 3-gram Freqs
def GetContentFreq(content):
    #Read Words
    translator = str.maketrans('', '', string.punctuation)
    words = nltk.word_tokenize(content)
    words = [word.translate(translator) for word in words]
    #Remove one letters
```

```
    words = [word for word in words if len(word) > 1]
    #Remove numbers
    words = [word for word in words if not word.isnumeric()]
    #Convert lowercase
    words = [word.lower() for word in words]
    #Remove stop-words
    words = [word for word in words if word not in
stopwords.words('english')]

    tempWords = []
    #for word in words:
    #    word = re.sub(r'(\w)\1+', r'\1', word)
    #    tempWords.append((word))
    #words = tempWords
    tempWords = []
    for word in words:
        if word != "" and len(word) > 1:
            tempWords.append((word))
    words = tempWords
    return words

tallyTableCount = 10000
```

```
def InsertTallyTable():
    SQLCommand = ("EXEC InsertTally ?")
    Values = [tallyTableCount]
    cursor.execute(SQLCommand,Values)
    connection.commit()

def SeparateWordAndSaveDB():
    counts = dict()
    paraghraphId = 0
    ClearDatabase() #TruncateSelectedDatabase()
    InsertTallyTable()
    path = DocumentSet
    sortlist = sorted(os.listdir(path))
    if len(sortlist) == 0:
        FillPathIfFolderPathIsEmpty()
        sortlist = sorted(os.listdir(path))
    i = 0
    while(i < len(sortlist)):
        dna = open(path + "\\" +
sortlist[i],encoding='utf8',errors='ignore')
        soup = BeautifulSoup(dna,"html.parser")
        paragraphs = soup.find_all("p")
        paraghraphId = 1
        stemmer = PorterStemmer()
        for paragraph in paragraphs:
            tokens = GetContentFreq(paragraph.text)
            tagged = pos_tag(tokens)
            nouns = [word for word,pos in tagged \
                if (pos == 'NN' or pos == 'NNP' or pos == 'NNS' or pos ==
'NNPS')]
            for word in nouns:
                originalWord = word
                stems = stemmer.stem(word)

                if stems in counts.keys():
                    shortest,count = counts[stems]
                    counts[stems] = (shortest,count + 1)
                else:
                    counts[stems] = (stems,1)
```

```
            for kok in counts:
                try:
                    shortest,count = counts[kok]
                    SQLCommand = ("INSERT INTO Words (DocumentId,Word,
Count,StemWord,Paragraph)  VALUES (?,?,?,?,?)")
                    Values = [i,shortest,count,kok,paraghraphId]
                    cursor.execute(SQLCommand,Values)
                    connection.commit()
                except:
                    pass
            counts.clear()
            tokens.clear()
            nouns.clear()
            paraghraphId+=1
        try:
            SQLCommand = ("INSERT INTO Documents
(Id,DocumentName,Topic,SubTopic)  VALUES (?,?,?,?)")
            Values =
[i,sortlist[i],sortlist[i].split("_")[0],sortlist[i].split("_")[1]]
            cursor.execute(SQLCommand,Values)
            connection.commit()
            i+=1
        except:
            pass
```

```
def LastCheckMeaningWords():
    cursor.execute("SELECT StemWord FROM
[SmallWordsEducation].[dbo].[MeaningWord] (nolock)")
    meaningWordList = cursor.fetchall()
    for meaningWord in meaningWordList:
        if not wordnet.synsets(meaningWord[0]):
            SQLCommand = ("DELETE FROM
[SmallWordsEducation].[dbo].[MeaningWord] WHERE StemWord=?")
            Values = [meaningWord[0]]
            cursor.execute(SQLCommand,Values)
            connection.commit()
SeparateWordAndSaveDB()
LastCheckMeaningWords()
SQLCommand = ("EXEC InsertMeaningValue")
cursor.execute(SQLCommand)
connection.commit()
```

```
if MeaningPercent == '50':
   SQLCommand = ("DELETE FROM [SmallWordsEducation].[dbo].[MeaningWord]
WHERE StemWord IN  (SELECT TOP (50) PERCENT StemWord FROM
[SmallWordsEducation].[dbo].[MeaningWord] ORDER BY MeaningValue asc)")
   cursor.execute(SQLCommand)
   connection.commit()
   DropWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] DROP Column WordId")
   cursor.execute(DropWordIdCommand)
   connection.commit()
   AddWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] ADD  WordId INT IDENTITY(0,1)")
   cursor.execute(AddWordIdCommand)
   connection.commit()

if MeaningPercent == '25':
```

```
    SQLCommand = ("DELETE FROM [SmallWordsEducation].[dbo].[MeaningWord]
WHERE StemWord IN (SELECT TOP (75) PERCENT StemWord FROM
[SmallWordsEducation].[dbo].[MeaningWord] order by MeaningValue asc)")
    cursor.execute(SQLCommand)
    connection.commit()
    DropWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] DROP Column WordId")
    cursor.execute(DropWordIdCommand)
    connection.commit()
    AddWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] ADD  WordId INT IDENTITY(0,1)")
    cursor.execute(AddWordIdCommand)
    connection.commit()

if MeaningPercent == '20':
    SQLCommand = ("DELETE FROM [SmallWordsEducation].[dbo].[MeaningWord]
WHERE StemWord IN (SELECT TOP (80) PERCENT StemWord FROM
[SmallWordsEducation].[dbo].[MeaningWord] order by MeaningValue asc)")
    cursor.execute(SQLCommand)
    connection.commit()
    DropWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] DROP Column WordId")
    cursor.execute(DropWordIdCommand)
    connection.commit()
    AddWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] ADD  WordId INT IDENTITY(0,1)")
    cursor.execute(AddWordIdCommand)
    connection.commit()

if MeaningPercent == '15':
    SQLCommand = ("DELETE FROM [SmallWordsEducation].[dbo].[MeaningWord]
WHERE StemWord IN (SELECT TOP (85) PERCENT StemWord FROM
[SmallWordsEducation].[dbo].[MeaningWord] order by MeaningValue asc)")
    cursor.execute(SQLCommand)
    connection.commit()
    DropWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] DROP Column WordId")
    cursor.execute(DropWordIdCommand)
    connection.commit()
    AddWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] ADD  WordId INT IDENTITY(0,1)")
    cursor.execute(AddWordIdCommand)
    connection.commit()

if MeaningPercent == '10':
    SQLCommand = ("DELETE FROM [SmallWordsEducation].[dbo].[MeaningWord]
WHERE StemWord IN (SELECT TOP (90) PERCENT StemWord FROM
[SmallWordsEducation].[dbo].[MeaningWord] order by MeaningValue asc)")
    cursor.execute(SQLCommand)
    connection.commit()
    DropWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] DROP Column WordId")
    cursor.execute(DropWordIdCommand)
    connection.commit()
    AddWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] ADD  WordId INT IDENTITY(0,1)")
    cursor.execute(AddWordIdCommand)
    connection.commit()

if MeaningPercent == '5':
```

```
    SQLCommand = ("DELETE FROM [SmallWordsEducation].[dbo].[MeaningWord]
WHERE StemWord IN (SELECT TOP (95) PERCENT StemWord FROM
[SmallWordsEducation].[dbo].[MeaningWord] order by MeaningValue asc)")
    cursor.execute(SQLCommand)
    connection.commit()
    DropWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] DROP Column WordId")
    cursor.execute(DropWordIdCommand)
    connection.commit()
    AddWordIdCommand = ("ALTER TABLE
[SmallWordsEducation].[dbo].[MeaningWord] ADD  WordId INT IDENTITY(0,1)")
    cursor.execute(AddWordIdCommand)
    connection.commit()

SelectMeaningWordCommand = ("SELECT COUNT(*) FROM MeaningWord")
cursor.execute(SelectMeaningWordCommand)
V = list(map(int,cursor.fetchall()[0]))
print("Toplam Meaning Word Sayısı:" + str(V))
#Toplam kelime sayısını bul
TotalWordsCommands = ("SELECT COUNT(*) FROM Words")
cursor.execute(TotalWordsCommands)
total_words = list(map(int,cursor.fetchall()[0]))
print("Toplam Kelime Sayısı:" + str(total_words))
```

```
def GetDocumentList():
    AllDocumentCommand = ("SELECT * FROM Documents")
    cursor.execute(AllDocumentCommand)
    return cursor.fetchall()
document_list = GetDocumentList()

def CheckContains(document,word):
    document = document[0]
    word = str(word[1]) # Stemword'ü alabilmek için word index'i 2
seçiliyor.
    IsContains = "SELECT Count(*) FROM Words WHERE DocumentId = ? and
StemWord =" + "'" + word + "'"
    documentId = [document]
    cursor.execute(IsContains,documentId)
    founded = cursor.fetchone()
    if(founded[0] > 0):
        return 1
    else:
        return 0
#Doküman vektörleri veri tabanına yazılıyor.
def WriteDocumentVectorToDB(document,document_vector,meaning_words):
    b = ",".join(str(i) for i in document_vector)
    InsertDocumentVector = ("INSERT INTO DocumentVector(DocumentId,Vector)
VALUES(?,?) ")
    params = [int(document[0]),str(b)]
    cursor.execute(InsertDocumentVector,params)

def VectorizeDocument(document_list):
     SelectMeaningWordLoop = ("SELECT * FROM MeaningWord")
     cursor.execute(SelectMeaningWordLoop)
     meaning_words = cursor.fetchall()
     for document in document_list:
         IsDocumentContainWord = []
         for word in meaning_words:
             IsDocumentContainWord.append(CheckContains(document,word))

WriteDocumentVectorToDB(document,IsDocumentContainWord,meaning_words)
```

```
def CalculateSingleWordProbabilityForDocuments(document):
    documentId = document[0]
    query = "SELECT TOP (1000) [DocumentId],[StemWord],[MeaningValue] FROM
[SmallWordsEducation].[dbo].[MeaningWord] WHERE DocumentId=" +
str(documentId)
    cursor.execute(query)
    results = cursor.fetchall()
    for r in results:
        CalculateSingleProp(r,documentId)

def CalculateSingleProp(r,documentId):
    singleProp = 0
    document_list = GetDocumentList()
    vectors = GetAllVectors()
    for vector in vectors:
        if vector[r[0]] == '1':
            singleProp = singleProp + 1

    InsertSingleWordProb = ("INSERT INTO
SingleWordProp(DocumentId,Word,Probability) VALUES(?,?,?) ")
    SingleWordValues = [int(documentId),r[1], singleProp /
len(document_list)]
    cursor.execute(InsertSingleWordProb,SingleWordValues)
```

```
def GetDocumentVector(document):
    query = "SELECT Vector FROM [SmallWordsEducation].[dbo].[DocumentVector]
WHERE DocumentId=" + str(document)
    cursor.execute(query)
    results = cursor.fetchall()
    vector = str(results[0])
    vector = vector[2:-3]
    vector_elements = vector.split(',')
    return vector_elements

def GetAllVectors():
    query = "SELECT Vector FROM [SmallWordsEducation].[dbo].[DocumentVector]
"
    cursor.execute(query)
    results = cursor.fetchall()
    vector_array = []
    for idx in results:
        vector = str(idx)
        vector = vector[2:-3]
        vector_elements = vector.split(',')
        vector_array.append(vector_elements)
    return vector_array

def GetWord(index):
    query = "SELECT StemWord FROM MeaningWord WHERE WordId=" + str(index)
    cursor.execute(query)
    results = cursor.fetchall()
    return str(results)
meaningwordlist = []
def GetMeainingWords():
    query = "SELECT StemWord FROM MeaningWord"
    cursor.execute(query)
    results = cursor.fetchall()
    meaningwordlist = results
    return meaningwordlist

def GetWordBag(index):
    query = "SELECT * FROM MeaningWord WHERE WordId=" + str(index)
```

```
    cursor.execute(query)
    results = cursor.fetchall()
    return results

def WriteAssociationPropToDB(word1index,word2index,prop):
    word1 = GetWord(word1index)
    word1 = word1[3:-4]
    word2 = GetWord(word2index)
    word2 = word2[3:-4]
    InsertProbs = ("INSERT INTO
ProbabilitiesOfAssociation(Word1,Word1Index,Word2,Word2Index,AssociationProp
) VALUES(?,?,?,?,?) ")
    InsertionValues = [word1,word1index,word2,word2index,prop]
    cursor.execute(InsertProbs,InsertionValues)

def CalculateTwoWordAssociation(i,j,documentCount):
    association = 0
    vector_list = GetAllVectors()

    for vector in vector_list:
        if vector[i] == "1" and vector[j] == "1":
            association = association + 1
    #   association = 0 ise laplace düzeltmesi uygulanacak
    association_prop = association / documentCount
    WriteAssociationPropToDB(i,j,association_prop)

def GetSingleProb(word):
    query = "SELECT Probability FROM SingleWordProp WHERE Word=" + "'" +
word + "'"
    cursor.execute(query)
    prop = cursor.fetchall()
    return prop[0][0]

def GetAssociatiedProb(word1,word2):
    query = "SELECT AssociationProp FROM ProbabilitiesOfAssociation WHERE
Word1=" + "'" + word1 + "'" + "AND Word2=" + "'" + word2 + "'"
    cursor.execute(query)
    prop = cursor.fetchall()
    return prop[0][0]

def CalculatePMIFinalization(i,j,document):
    print("PMI Similarity Calculation Started...")
    documentId = document[0]
    word1 = GetWord(i)
    word1 = word1[3:-4]
    word2 = GetWord(j)
    word2 = word2[3:-4]

    TotalWordsCommands = ("SELECT COUNT(*) FROM MeaningWord")
    cursor.execute(TotalWordsCommands)
    V = list(map(int,cursor.fetchall()[0]))
    word1Bag = GetWordBag(i)
    word2Bag = GetWordBag(j)
    p1 = GetSingleProb(word1)
    p2 = GetSingleProb(word2)
    p12 = GetAssociatiedProb(word1,word2)
    PMI2 = math.log2(((float(p12)) / ((float(p1) * float(p2)))))
    print("P(" + word1 + "," + word2 + ") =" + str(PMI2))
    InsertPMI = ("INSERT INTO
DocumentsPMI(WordOne,WordOneId,WordTwo,WordTwoId,DocumentId,PMI)
VALUES(?,?,?,?,?,?) ")
```

```
    InsertionPMIValues =
[word1,int(word1Bag[0][3]),word2,int(word2Bag[0][3]),documentId,float(PMI2)]
    cursor.execute(InsertPMI,InsertionPMIValues)

def WriteSimilarityToDB(document1,document2,result,document_list):
    topic1 = document_list[document1][2]
    topic2 = document_list[document2][2]
    subtopic1 = document_list[document1][3]
    subtopic2 = document_list[document2][3]

    InsertSimilarity = ("INSERT INTO
DocumentSimilarity(DocumentOne,DocumentOneTopic,DocumentOneSubTopic,Document
Two,DocumentTwoTopic,DocumentTwoSubTopic,SimilarityResult)
VALUES(?,?,?,?,?,?,?) ")
    InsertionSimilarityValues =
[document1,topic1,subtopic1,document2,topic2,subtopic2,float(result)]
    cursor.execute(InsertSimilarity,InsertionSimilarityValues)
```

```
def Calculate(i,j,document_list):
    s = 0
    print("D(" + str(i) + "," + str(j) + ")")
    i_vals = []
    D1Vector = GetDocumentVector(document_list[i][0])
    D2Vector = GetDocumentVector(document_list[j][0])
    #Dokümanlara ait vektörler çekilip, iki dokümanda da yer alan kelimeleri
    #toplam benzerliğe ekliyoruz.
    for index in range(0,len(D1Vector)):
        if(int(D1Vector[index]) == 1 and int(D2Vector[index]) == 1):
            query = "SELECT PMI FROM
[SmallWordsEducation].[dbo].[DocumentsPMI] WHERE WordOneId = ?  OR WordTwoId
= ?"
            params = [i,j]
            cursor.execute(query,params)
            results = cursor.fetchone()
            i_vals.append(float(results[0]))
            s = sum(i_vals)
    WriteSimilarityToDB(i,j, s,document_list)

def CalculateSimilarities():
    document_list = GetDocumentList()
    i_vals = []
    for i in range(0,len(document_list)):
        i_vals.append(i)
        for j in range(0,len(document_list)):
             if i != j :
                  Calculate(i,j,document_list)

def CalculatePMI():
    document_list = GetDocumentList()
    for document in document_list:
        vectors = GetDocumentVector(document[0])
        i_vals = []
        for i in range(0,len(vectors)):
           i_vals.append(i)
           for j in range(0,len(vectors)):
              if i == j :
                   print("P(" + str(i) + "," + str(j) + ")")
                   CalculateTwoWordAssociation(i,j,len(document_list))
                   CalculatePMIFinalization(i,j,document)
              if not j in i_vals:
                  if vectors[i] == "1" and vectors[j] == "1":
                      print("P(" + str(i) + "," + str(j) + ")")
```

```
                        CalculateTwoWordAssociation(i,j,len(document_list))
                        CalculatePMIFinalization(i,j,document)
```

```
#cosine similarity
sortlist = sorted(os.listdir(DocumentSet))
documentlist = []
if len(sortlist) == 0:
    FillPathIfFolderPathIsEmpty()
    sortlist = sorted(os.listdir(path))
i = 0
while(i < len(sortlist)):
        dna = open(DocumentSet + "\\" +
sortlist[i],encoding='utf8',errors='ignore')
        soup = BeautifulSoup(dna,"html.parser")
        documentlist.append(soup)
        i+=1
stemmer = nltk.stem.porter.PorterStemmer()
remove_punctuation_map = dict((ord(char), None) for char in
string.punctuation)

def CalculateCosineSimilarity(train_set):
    print("Cosine Similarity Calculation Started...")
    nlp = spacy.load('en')
    for i in range(0,len(document_list)):
        for j in range(0,len(document_list)):
            if i != j :

InsertCosineSimilarityToDB(i,j,cosine_sim(str(train_set[i].contents),str(tra
in_set[j].contents)))
                print("D["   +     str(i)   +   "]"   +   "D["   +str(j)+"]",
cosine_sim(str(train_set[i].contents),str(train_set[j].contents)))
```

```
def CalculateJaccardSimilarity(train_set):
    print("Jaccard Similarity Calculation Started...")
    for i in range(0,len(document_list)):
        for j in range(0,len(document_list)):
            if i != j :

InsertJaccardSimilarityToDB(i,j,jaccard_similarity(str(train_set[i].contents
),str(train_set[j].contents)))
                print("D[" +  str(i) + "]" + "D[" +str(j)+"]",
jaccard_similarity(str(train_set[i].contents),str(train_set[j].contents)))

def InsertCosineSimilarityToDB(i,j,similarity):
    topic1 = document_list[i][2]
    topic2 = document_list[j][2]
    subtopic1 = document_list[i][3]
    subtopic2 = document_list[j][3]
    InsertSimilarity = ("INSERT INTO
CosineSimilarity(DocumentOneId,DocumentOneTopic,DocumentOneSubTopic,
DocumentTwoId,DocumentTwoTopic,DocumentTwoSubTopic,Similarity)
VALUES(?,?,?,?,?,?,?) ")
    InsertionSimilarityValues =
[i,topic1,subtopic1,j,topic2,subtopic2,float(similarity)]
    cursor.execute(InsertSimilarity,InsertionSimilarityValues)

def InsertJaccardSimilarityToDB(i,j,similarity):

    topic1 = document_list[i][2]
    topic2 = document_list[j][2]
    subtopic1 = document_list[i][3]
    subtopic2 = document_list[j][3]
```

```
    InsertSimilarity = ("INSERT INTO
JaccardSimilarity(DocumentOneId,DocumentOneTopic,DocumentOneSubTopic,
DocumentTwoId,DocumentTwoTopic,DocumentTwoSubTopic,JaccardSimilarity)
VALUES(?,?,?,?,?,?,?) ")
    InsertionSimilarityValues =
[i,topic1,subtopic1,j,topic2,subtopic2,float(similarity)]
    cursor.execute(InsertSimilarity,InsertionSimilarityValues)

def stem_tokens(tokens):
    return [stemmer.stem(item) for item in tokens]

def normalize(text):
    return
stem_tokens(nltk.word_tokenize(text.lower().translate(remove_punctuation_map
)))
vectorizer = TfidfVectorizer(tokenizer=normalize, stop_words='english')
def cosine_sim(text1, text2):
    tfidf = vectorizer.fit_transform([text1, text2])
    return ((tfidf * tfidf.T).A)[0,1]

CalculateCosineSimilarity(documentlist)

# Jaccard Index
def jaccard_similarity(query, document):
    intersection = set(query).intersection(set(document))
    union = set(query).union(set(document))
    return len(intersection)/len(union)

CalculateJaccardSimilarity(documentlist)
```

```
#Dokümanların vektör olarak ifade edilmesi
VectorizeDocument(document_list)
#Kelimelerin herbir doküman içerisinden tek olarak geçme olasılıkları
for document in document_list:
    CalculateSingleWordProbabilityForDocuments(document)
#Birlikteliklerin hesaplanması
CalculatePMI()
##Doküman benzerliklerinin hesaplanması
CalculateSimilarities()
connection.commit()
cursor.close()
end = datetime.datetime.now()
print("Elapsed Time: " + str(end - start))
```

```
# K-Means Clustering

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv(Document_Similarity100.csv')

X = dataset.iloc[:,:3].values

X = pd.DataFrame(X)
X = X.convert_objects(convert_numeric=True)
X.columns = [' DocumentOne',' DocumentTwo',' Similarity']
```

```
# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(1,2):
    kmeans = KMeans(n_clusters=i,init='k-
means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit_transform(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,2),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()


kmeans = KMeans(n_clusters=2,init='k-
means++',max_iter=300,n_init=10,random_state=0)
y_kmeans = kmeans.fit_predict(X)

X = X.as_matrix(columns=None)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans ==
0,1],s=100,c='red',label='Eğitim')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans ==
1,1],s=100,c='blue',label='Spor')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=300,
c='yellow',label='Centroids')
plt.title('Konulara göre dokümanların kümelenmesi')
plt.legend()
plt.show(block=True)
```

```
def getSimilarityResults(document):
    documentId = document
    query = "SELECT SimilarityResult FROM
[SmallWordsEducation].[dbo].[DocumentSimilarity] WHERE DocumentOne =" +
str(documentId)
    cursor.execute(query)
    results = cursor.fetchall()
    return results


def KMeansManual():
    document_list = GetDocumentList()
    for document in document_list:
        print("****************Doküman NO:", document[0])
        pmi_values = getSimilarityResults(document[0])
        centroids = centroid_noktası_olustur(pmi_values)
        total_iteration = 100
        [cluster_label, new_centroids] = k_means(pmi_values, centroids,
total_iteration,document)

        print_data([cluster_label, new_centroids])
        print()


def InsertKmeansResults(documentId,pmi_values,centroid,cluster_label):
    InsertKmeansResult = ("INSERT INTO
KMeansResult(DocumentId,PMIResult,OklidDistance, Cluster) VALUES(?,?,?,?) ")
    InsertionKmeansValues =
[documentId,float(pmi_values[0]),centroid,cluster_label]
    cursor.execute(InsertKmeansResult,InsertionKmeansValues)
```

```
def centroid_noktası_olustur(pmi_values):
    centroids = []

    centroids.append([random.choice(pmi_values)])
    centroids.append([random.choice(pmi_values)])
    return np.array(centroids)

def oklid_uzaklik_hesapla(point, centroid):
    return np.sqrt(np.sum((float(point[0]) - float(centroid))**2))

def kümelere_etiket_ata(distance, data_point, centroids):
    index_of_minimum = min(distance, key=distance.get)
    return [index_of_minimum, data_point, centroids[index_of_minimum]]

def yeni_centroidleri_hesapla(cluster_label, centroids):
    return np.array(float(cluster_label[0]) +float(centroids))/2

def k_means(data_points, centroids, total_iteration,document):
    label = []
    cluster_label = []
    total_points = len(data_points)
    k = len(centroids)

    for iteration in range(0, total_iteration):
        for index_point in range(0, total_points):
            distance = {}
            for index_centroid in range(0, k):
                distance[index_centroid] =
oklid_uzaklik_hesapla(data_points[index_point], centroids[index_centroid])
            label = kümelere_etiket_ata(distance, data_points[index_point],
centroids)
            centroids[label[0]] = yeni_centroidleri_hesapla(label[1],
centroids[label[0]])

            if iteration == (total_iteration - 1):
                cluster_label.append(label)
            InsertKmeansResults(document[0],
data_points[index_point],distance[index_centroid],label[0])
    return [cluster_label, centroids]

def getPMIValuesForDocument(document):
    documentId =document
    query = "SELECT PMI FROM [SmallWordsEducation].[dbo].[DocumentsPMI]
WHERE DocumentId =" + str(documentId)
    cursor.execute(query)
    results = cursor.fetchall()
    return results

def print_data(result):
    print("K-Means Kümeleme Sonuçları: \n")
    for data in result[0]:
        print("PMI Sonucu: {}".format(data[1]))
        print("Küme: {} \n".format(data[0]))
    print("Centroid'lerin son pozisyonları: \n {}".format(result[1]))

#Kmeans ile Kümeleme
print("****************K MEANS WITH OKLID ****************************:",
document[0])
KMeansManual()

#Kmeans ve PMI ile Kümeleme
```

```
def pmi_uzaklik_hesapla(pmiscore, centroid):
    distance = float(pmiscore[0]) - float(centroid)
    return distance

def getSimilarityResults(document):
    documentId = document
    query = "SELECT SimilarityResult FROM
[SmallWordsEducation].[dbo].[DocumentSimilarity] WHERE DocumentOne =" +
str(documentId)
    cursor.execute(query)
    results = cursor.fetchall()
    return results


def InsertKmeansPMIResults(documentId,pmi_values,centroid,cluster_label):
    InsertKmeansResult = ("INSERT INTO
KMeansPMIResult(DocumentId,PMIResult,PMIDistance,Cluster) VALUES(?,?,?,?) ")
    InsertionKmeansValues =
[documentId,float(pmi_values[0]),centroid,cluster_label]
    cursor.execute(InsertKmeansResult,InsertionKmeansValues)

def k_meansPMIAlgorithm(data_points, centroids, total_iteration,document):
    label = []
    cluster_label = []
    total_points = len(data_points)
    k = len(centroids)

    for iteration in range(0, total_iteration):
        for index_point in range(0, total_points):
            distance = {}
            for index_centroid in range(0, k):
                distance[index_centroid] =
pmi_uzaklik_hesapla(data_points[index_point], centroids[index_centroid])
            label = kümelere_etiket_ata(distance, data_points[index_point],
centroids)
            centroids[label[0]] = yeni_centroidleri_hesapla(label[1],
centroids[label[0]])

            if iteration == (total_iteration - 1):
                cluster_label.append(label)
            InsertKmeansPMIResults(document[0],
data_points[index_point],float(distance[index_centroid]),label[0])
    return [cluster_label, centroids]

def KMeansPMI():
    document_list = GetDocumentList()
    for document in document_list:
        print("****************Doküman NO:", document[0])
        similarityResults = getSimilarityResults(document[0])
        centroids = centroid_noktası_olustur(similarityResults)
        total_iteration = 100
        [cluster_label, new_centroids] =
k_meansPMIAlgorithm(similarityResults, centroids, total_iteration,document)
        print_data([cluster_label, new_centroids])
        print()
print("****************K MEANS WITH PMI ******************************:",
document[0])
KMeansPMI()
connection.commit()
def Visualation(algType):
    if(algType == "Euclidian"):
```

```
        DrawGraphEuclidian()
    else:
        DrawGraphPMI()


def GetClusterData():
    Clusters = ("SELECT OklidDistance,PMIResult,Cluster FROM
[SmallWordsEducation].[dbo].[KMeansResult]  GROUP BY
Cluster,OklidDistance,PMIResult")
    cursor.execute(Clusters)
    return cursor.fetchall()

def GetClusterDataPMI():
    Clusters = ("SELECT PMIDistance,PMIResult,Cluster FROM
[SmallWordsEducation].[dbo].[KMeansPMIResult]  GROUP BY
Cluster,PMIDistance,PMIResult")
    cursor.execute(Clusters)
    return cursor.fetchall()

def DrawGraphEuclidian():
    cluster = []
    cluster = GetClusterData()
    series_numeric = np.array(cluster)

plt.scatter(series_numeric[:,0],series_numeric[:,1],c=series_numeric[:,2])
    plt.title("Kmeans ve Öklid Uzaklığı kullanarak Küme Tahmini")
    plt.xlabel("Öklid uzaklıkları")
    plt.ylabel("Hesaplanan PMI değerleri")
    plt.show(block=True)
Visualation('Euclidian')

def DrawGraphPMI():
    cluster = []
    cluster = GetClusterDataPMI()
    series_numeric = np.array(cluster)

plt.scatter(series_numeric[:,0],series_numeric[:,1],c=series_numeric[:,2])
    plt.title("Kmeans ve PMI Uzaklığı kullanarak Küme Tahmini")
    plt.xlabel("PMI uzaklıkları")
    plt.ylabel("Hesaplanan PMI değerleri")
    plt.show(block=True)
Visualation()
##   K-means uzatma ##
import pypyodbc
import string
import random
import math

from Color_Console import *

connection = pypyodbc.connect('Driver={SQL Server};'
                                                'Server=DESKTOP-
TMO951D\GORKEMPC;'

'Database=SmallWordsEducation;'
                                                    'uid=sa;pwd=1')
cursor = connection.cursor()

## Veri tabanı methodları

def GetDocumentList():
    AllDocumentCommand = ("SELECT * FROM Documents")
```

```
    cursor.execute(AllDocumentCommand)
    return cursor.fetchall()

def GetDocumentVector(document):
    query = "SELECT Vector FROM [SmallWordsEducation].[dbo].[DocumentVector]
WHERE DocumentId=" + str(document)
    cursor.execute(query)
    results = cursor.fetchall()
    vector = str(results[0])
    vector = vector[2:-3]
    vector_elements = vector.split(',')
    return vector_elements

def GetDocumentVectors():
    query = "SELECT Vector FROM [SmallWordsEducation].[dbo].[DocumentVector]
"
    cursor.execute(query)
    results = cursor.fetchall()
    return results

def GetPMIValue(wordone,wordtwo):
    query = "SELECT Top 1 PMI FROM [SmallWordsEducation].[dbo].[DocumentsPMI]
WHERE WordOneId =" + str(wordone) + "and WordTwoId=" + str(wordtwo)
    cursor.execute(query)
    results = cursor.fetchone()
    return results

def PickRandomDocument():
    new_document = random.choice(alldocuments)
    return new_document[0]

def CalculateNewCentroid(centroid,currentDocument):
    #Merkez noktası ile karşılaştırılacak olan dokümanın vektörü
    newDocumentVector = GetDocumentVector(newDocument)
    newCentroidVectorCalc =  []
    #Rastgele seçilen merkez noktasının vektörü
    centroidVector = GetDocumentVector(centroid)
    for i,val in enumerate(centroidVector):
            if centroidVector[i] == '1' and newDocumentVector[i] == '1':
                newCentroidVectorCalc.append('1')
            else:
                newCentroidVectorCalc.append('0')
    return newCentroidVectorCalc

#def FirstCalculation(centroid1,centroid2,newDocument):
#    global newCentroidVectorClass1
#    global newCentroidVectorClass2
#    classoneSimilarity = CalculateSimilarityClass(centroid1,newDocument)
#    classtwoSimilarity = CalculateSimilarityClass(centroid2,newDocument)

#    if(classoneSimilarity>classtwoSimilarity):
#        for doc in alldocuments:
#            if(doc[0] == newDocument):
#                class1.append(doc)
#                alldocuments.remove(doc)
#                                                newCentroidVectorClass1   =
CalculateNewCentroid(centroid1,newDocument)
#    else:
#        for doc in alldocuments:
#            if(doc[0] == newDocument):
#                class2.append(doc)
#                alldocuments.remove(doc)
```

```
#                                                   newCentroidVectorClass2   =
CalculateNewCentroid(centroid2,newDocument)

def
CalculatePMIFinalization(i,j,singlePropList,currentDocumentVector,centroid):

    found = 0
    associatedProb = 0
    if(centroid[i] == "1" and centroid[j] == "1"):
        found = found +1
    if(currentDocumentVector[i] == "1" and currentDocumentVector[j] == "1"):
        found = found +1

    associatedProb = found /  2

    word1SingleProb = singlePropList[i]
    word2SingleProb = singlePropList[j]
    PMI = math.log2(((float(associatedProb)) + 1 / 1 +((float(word1SingleProb)
* float(word2SingleProb)))))

    return PMI

def CalculatePMI(centroid,currentDocument):
    totalPMI = 0
    singlePropList =[]
    ascPropList =[]
    currentDocumentVector = GetDocumentVector(currentDocument)
    for index in range(len(currentDocumentVector)-1):
        singlePropCount = 0
        if centroid[index] == "1":
            singlePropCount = singlePropCount +1
        if currentDocumentVector[index] == "1":
            singlePropCount = singlePropCount +1

        singlePropForWord = singlePropCount/ len(currentDocumentVector)
        singlePropList.append(singlePropForWord)

    i_vals = []
    for i in range(0,len(currentDocumentVector)-1):
           i_vals.append(i)
           for j in range(0,len(currentDocumentVector)):
              if i == j :
                   totalPMI             =             totalPMI             +
CalculatePMIFinalization(i,j,singlePropList,currentDocumentVector,centroid)

              if not j in i_vals:
                  if currentDocumentVector[i] == "1" and centroid[j] == "1":
                      #CalculateTwoWordAssociation(i,j,2)
                      totalPMI            =            totalPMI            +
CalculatePMIFinalization(i,j,singlePropList,currentDocumentVector,centroid)

    return totalPMI

def
EsikDegeriKontrol(index,classDocumentVectorList,totalCount,matchedCount):
    for cd in classDocumentVectorList:
        if cd[index] == "1":
            matchedCount = matchedCount +1

    percentage = ((matchedCount / totalCount) * 100)
    return percentage
```

```
def UpdateCentroid(newDocument,newCentroidVectorClass,classList):
    newDocumentVector =GetDocumentVector(newDocument)

    classDocumentVectorList =[]

    for document in classList:
        classDocumentVectorList.append(GetDocumentVector(document[0]))

    matchedCount = 0
    totalCount = len(classDocumentVectorList)

    for index in range(0,len(newDocumentVector)-1):
        if newDocumentVector[index] == "1":
            percentage=
EsikDegeriKontrol(index,classDocumentVectorList,totalCount,matchedCount)

            if(percentage >= float(esikdegeri)):
                newCentroidVectorClass[index] = "1"
            else:
                newCentroidVectorClass[index] = "0"

    return newCentroidVectorClass

def CalculateWithCentroidFirstTime(centroid1,centroid2,newDocument):
    global newCentroidVectorClass1
    global newCentroidVectorClass2

    newCentroidVectorClass1 = GetDocumentVector(centroid1)
    newCentroidVectorClass2 = GetDocumentVector(centroid2)

    classonePMI = CalculatePMI(newCentroidVectorClass1,newDocument)
    classtwoPMI = CalculatePMI(newCentroidVectorClass2,newDocument)

    if(classonePMI>classtwoPMI):
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "green")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "red")
                ctext(str(newDocument) + " class1'e eklenecektir", "blue")
                print("\n")
                class1.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass1                                    =
UpdateCentroid(newDocument,newCentroidVectorClass1,class1)
    elif classonePMI<classtwoPMI:
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "red")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "green")
                ctext(str(newDocument) + " Class2'e eklenecektir", "blue")
                print("\n")
                class2.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass2                                    =
UpdateCentroid(newDocument,newCentroidVectorClass2,class2)


def
CalculateWithCentroid(newCentroidVectorClass1,newCentroidVectorClass2,newDoc
ument):
    classonePMI = CalculatePMI(newCentroidVectorClass1,newDocument)
    classtwoPMI = CalculatePMI(newCentroidVectorClass2,newDocument)
```

```
    if(classonePMI>classtwoPMI):
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "green")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "red")
                ctext(str(newDocument) + " class1'e eklenecektir", "blue")
                print("\n")
                class1.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass1                                          =
UpdateCentroid(newDocument,newCentroidVectorClass1,class1)
    elif classonePMI<classtwoPMI:
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "red")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "green")
                ctext(str(newDocument) + " Class2'e eklenecektir", "blue")
                print("\n")
                class2.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass2                                          =
UpdateCentroid(newDocument,newCentroidVectorClass2,class2)

def CalculateSimilarityClass(centroid,newDocument):
    totalSimilarity = 0
    #Merkez noktası ile karşılaştırılacak olan dokümanın vektörü
    newDocumentVector = GetDocumentVector(newDocument)
    #Rastgele seçilen merkez noktasının vektörü
    centroidVector = GetDocumentVector(centroid)
    for i,val in enumerate(centroidVector):
        if centroidVector[i] == "1" and newDocumentVector[i] == "1":
            pmi_value = GetPMIValue(i,i)
            if(pmi_value != None):
                totalSimilarity += pmi_value[0]
    return totalSimilarity

#Algoritma Adımları

class1 = []
class2 = []
newCentroidVectorClass1 = []
newCentroidVectorClass2 = []


alldocuments = GetDocumentList()

secim = input("Manuel seçim : o  Rastgele Seçim : r \n")
esikdegeri = input("Öbek Eşik % Değeri :")
if secim == "r" :
    rastgeleClass1 = random.choice(alldocuments)
    alldocuments.remove(rastgeleClass1)
    rastgeleClass2 = random.choice(alldocuments)
    alldocuments.remove(rastgeleClass2)
elif secim == "o":
    rastgeleClass1 = input("Class 1 için merkez noktası olarak seçilecek
doküman")
    alldocuments.remove(rastgeleClass1)
    rastgeleClass2 = input("Class 2 için merkez noktası olarak seçilecek
doküman")
    alldocuments.remove(rastgeleClass2)
```

```
class1.append(rastgeleClass1)
class2.append(rastgeleClass2)

ctext("Rastgele seçilen ve Class1'e eklenen ilk doküman :" +
str(rastgeleClass1),"magenta")
ctext("Rastgele seçilen ve Class2'ye eklenen ilk doküman :" +
str(rastgeleClass2), "yellow")

centroid1 = class1[0][0]
centroid2 = class2[0][0]

ctext("Öbekleme başlıyor...",'magenta')
while len(alldocuments) > 0:
    newDocument = PickRandomDocument()
    ctext("Öbeklenecek Doküman:" + str(newDocument),"yellow")
    print("\n")
    if len(newCentroidVectorClass1) == 0 and len(newCentroidVectorClass2) ==
0:
        ctext("İlk kez hesaplama yapılıyor...CalculateWithCentroidFirstTime
methodu çağrıldı","white")
        print("\n")
        CalculateWithCentroidFirstTime(centroid1, centroid2,newDocument)
    else:
        CalculateWithCentroid(newCentroidVectorClass1,
newCentroidVectorClass2,newDocument)


    #if len(newCentroidVectorClass1) == 0:
    #   FirstCalculation(centroid1,centroid2,newDocument)
    #elif len(newCentroidVectorClass2)  == 0 :
    #   FirstCalculation(centroid1,centroid2,newDocument)
    #else:
    #                        CalculateWithCentroid(newCentroidVectorClass1,
newCentroidVectorClass2,newDocument)




ctext("Class1 e eklenen doküman sayısı:" + str(len(class1)),"cyan")
ctext("Class2 e eklenen doküman sayısı:" + str(len(class2)),"cyan")
```

```
##   K-means uzatma ##
import pypyodbc
import string
import random
import math

from Color_Console import *

connection = pypyodbc.connect('Driver={SQL Server};'
                                              'Server=DESKTOP-
TMO951D\GORKEMPC;'

'Database=SmallWordsEducation;'
                                                 'uid=sa;pwd=1')
cursor = connection.cursor()

## Veri tabanı methodları

def GetDocumentList():
    AllDocumentCommand = ("SELECT * FROM Documents")
```

```
    cursor.execute(AllDocumentCommand)
    return cursor.fetchall()

def GetDocumentListCount():
    AllDocumentCommand = ("SELECT Count(*) FROM Documents")
    cursor.execute(AllDocumentCommand)
    return cursor.fetchone()

def GetDocumentVector(document):
    query = "SELECT Vector FROM [SmallWordsEducation].[dbo].[DocumentVector]
WHERE DocumentId=" + str(document)
    cursor.execute(query)
    results = cursor.fetchall()
    vector = str(results[0])
    vector = vector[2:-3]
    vector_elements = vector.split(',')
    return vector_elements

def GetDocumentVectors():
    query = "SELECT Vector FROM [SmallWordsEducation].[dbo].[DocumentVector]
"
    cursor.execute(query)
    results = cursor.fetchall()
    return results

def GetPMIValue(wordone,wordtwo):
    query = "SELECT Top 1 PMI FROM [SmallWordsEducation].[dbo].[DocumentsPMI]
WHERE WordOneId =" + str(wordone) + "and WordTwoId=" + str(wordtwo)
    cursor.execute(query)
    results = cursor.fetchone()
    return results

def PickRandomDocument():
    new_document = random.choice(alldocuments)
    return new_document[0]

def CalculateNewCentroid(centroid,currentDocument):
    #Merkez noktası ile karşılaştırılacak olan dokümanın vektörü
    newDocumentVector = GetDocumentVector(newDocument)
    newCentroidVectorCalc =  []
    #Rastgele seçilen merkez noktasının vektörü
    centroidVector = GetDocumentVector(centroid)
    for i,val in enumerate(centroidVector):
            if centroidVector[i] == '1' and newDocumentVector[i] == '1':
                newCentroidVectorCalc.append('1')
            else:
                newCentroidVectorCalc.append('0')
    return newCentroidVectorCalc

def
CalculatePMIFinalization(i,j,singlePropList,currentDocumentVector,centroid):

    found = 0
    associatedProb = 0
    if(centroid[i] == "1" and centroid[j] == "1"):
        found = found +1
    if(currentDocumentVector[i] == "1" and currentDocumentVector[j] == "1"):
        found = found +1

    associatedProb = found /  2

    word1SingleProb = singlePropList[i]
```

```
    word2SingleProb = singlePropList[j]
    PMI = math.log2(((float(associatedProb)) + 1 / 1 +((float(word1SingleProb)
* float(word2SingleProb)))))

    return PMI

def CalculatePMI(centroid,currentDocument):
    totalPMI = 0
    singlePropList =[]
    ascPropList =[]
    currentDocumentVector = GetDocumentVector(currentDocument)
    for index in range(len(currentDocumentVector)-1):
        singlePropCount = 0
        if centroid[index] == "1":
            singlePropCount = singlePropCount +1
        if currentDocumentVector[index] == "1":
            singlePropCount = singlePropCount +1

        singlePropForWord = singlePropCount/ len(currentDocumentVector)
        singlePropList.append(singlePropForWord)

    i_vals = []
    for i in range(0,len(currentDocumentVector)-1):
           i_vals.append(i)
           for j in range(0,len(currentDocumentVector)):
              if i == j :
                   totalPMI            =            totalPMI             +
CalculatePMIFinalization(i,j,singlePropList,currentDocumentVector,centroid)

              if not j in i_vals:
                  if currentDocumentVector[i] == "1" and centroid[j] == "1":
                      #CalculateTwoWordAssociation(i,j,2)
                      totalPMI           =            totalPMI           +
CalculatePMIFinalization(i,j,singlePropList,currentDocumentVector,centroid)

    return totalPMI

def
EsikDegeriKontrol(index,classDocumentVectorList,totalCount,matchedCount):
    for cd in classDocumentVectorList:
        if cd[index] == "1":
            matchedCount = matchedCount +1

    percentage = ((matchedCount / totalCount) * 100)
    return percentage

def UpdateCentroid(newDocument,newCentroidVectorClass,classList):
    newDocumentVector =GetDocumentVector(newDocument)

    classDocumentVectorList =[]

    for document in classList:
        classDocumentVectorList.append(GetDocumentVector(document[0]))

    matchedCount = 0
    totalCount = len(classDocumentVectorList)

    for index in range(0,len(newDocumentVector)-1):
        if newDocumentVector[index] == "1":
            percentage=
EsikDegeriKontrol(index,classDocumentVectorList,totalCount,matchedCount)
```

```
            if(percentage >= float(esikdegeri)):
                newCentroidVectorClass[index] = "1"
            else:
                newCentroidVectorClass[index] = "0"

    return newCentroidVectorClass



def            CalculateWithCentroidFirstTime(centroid1,centroid2,centroid3,
newDocument):
    global newCentroidVectorClass1
    global newCentroidVectorClass2
    global newCentroidVectorClass3

    global correctClass1
    global correctClass2
    global correctClass3

    correctClass1 = 0
    correctClass2 = 0
    correctClass3 = 0

    newCentroidVectorClass1 = GetDocumentVector(centroid1)
    newCentroidVectorClass2 = GetDocumentVector(centroid2)
    newCentroidVectorClass3 = GetDocumentVector(centroid3)

    classonePMI = CalculatePMI(newCentroidVectorClass1,newDocument)
    classtwoPMI = CalculatePMI(newCentroidVectorClass2,newDocument)
    classthreePMI = CalculatePMI(newCentroidVectorClass3,newDocument)

    if(classonePMI>classtwoPMI and classonePMI>classthreePMI):
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "green")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "red")
                ctext("Class3   için   PMI   Değeri:"   +   str(classthreePMI),
"magenta")
                ctext(str(newDocument) + " class1'e eklenecektir", "blue")
                print("\n")
                class1.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass1                                          =
UpdateCentroid(newDocument,newCentroidVectorClass1,class1)
                if(doc[2] == class1Type):
                    correctClass1 = correctClass1 +1
    elif (classtwoPMI>classonePMI and classtwoPMI>classthreePMI):
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "red")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "green")
                ctext("Class3   için   PMI   Değeri:"   +   str(classthreePMI),
"magenta")
                ctext(str(newDocument) + " Class2'e eklenecektir", "blue")
                print("\n")
                class2.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass2                                          =
UpdateCentroid(newDocument,newCentroidVectorClass2,class2)
                if(doc[2] == class2Type):
                    correctClass2 = correctClass2 +1
```

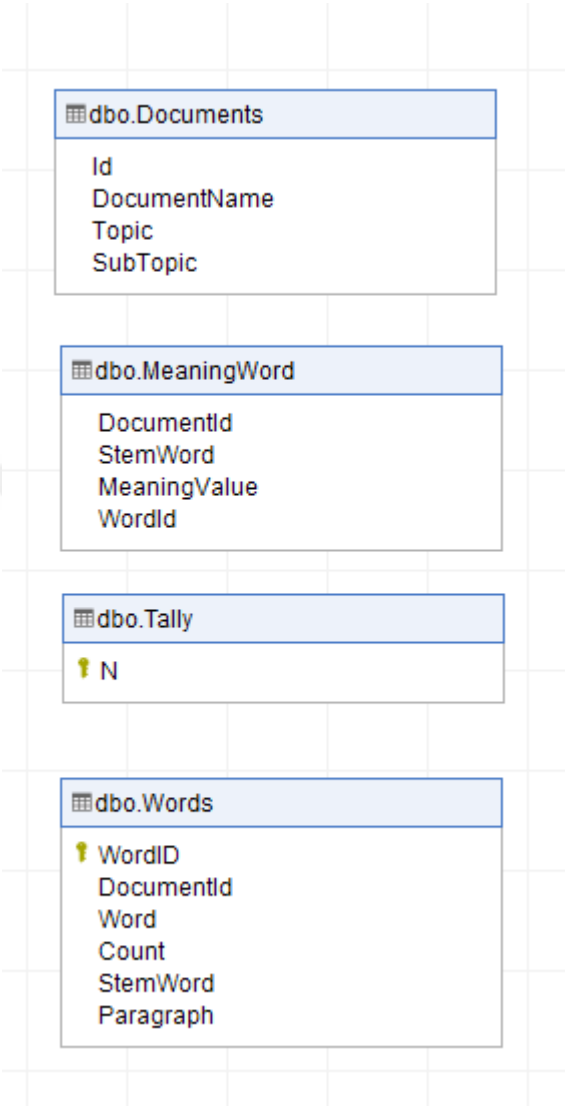
```
    elif (classthreePMI>classonePMI and classthreePMI>classtwoPMI):
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "red")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "green")
                ctext("Class3  için  PMI  Değeri:"  +  str(classthreePMI),
"magenta")
                ctext(str(newDocument) + " Class3'e eklenecektir", "blue")
                print("\n")
                class3.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass3                                    =
UpdateCentroid(newDocument,newCentroidVectorClass3,class3)
                if(doc[2] == class3Type):
                    correctClass3 = correctClass3 +1

def
CalculateWithCentroid(newCentroidVectorClass1,newCentroidVectorClass2,newCen
troidVectorClass3,newDocument):

    classonePMI = CalculatePMI(newCentroidVectorClass1,newDocument)
    classtwoPMI = CalculatePMI(newCentroidVectorClass2,newDocument)
    classthreePMI = CalculatePMI(newCentroidVectorClass3,newDocument)

    global correctClass1
    global correctClass2
    global correctClass3


    if(classonePMI>classtwoPMI and classonePMI>classthreePMI):
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "green")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "red")
                ctext("Class3  için  PMI  Değeri:"  +  str(classthreePMI),
"yellow")
                ctext(str(newDocument) + " class1'e eklenecektir", "blue")
                print("\n")
                class1.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass1                                    =
UpdateCentroid(newDocument,newCentroidVectorClass1,class1)
                if(doc[2] == class1Type):
                    correctClass1 = correctClass1 +1
    elif (classtwoPMI>classonePMI and classtwoPMI>classthreePMI):
        for doc in alldocuments:
            if(doc[0] == newDocument):
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "red")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "green")
                ctext("Class3  için  PMI  Değeri:"  +  str(classthreePMI),
"yellow")
                ctext(str(newDocument) + " Class2'e eklenecektir", "blue")
                print("\n")
                class2.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass2                                    =
UpdateCentroid(newDocument,newCentroidVectorClass2,class2)
                if(doc[2] == class2Type):
                    correctClass2 = correctClass2 +1
    elif (classthreePMI>classonePMI and classthreePMI>classtwoPMI):
        for doc in alldocuments:
            if(doc[0] == newDocument):
```

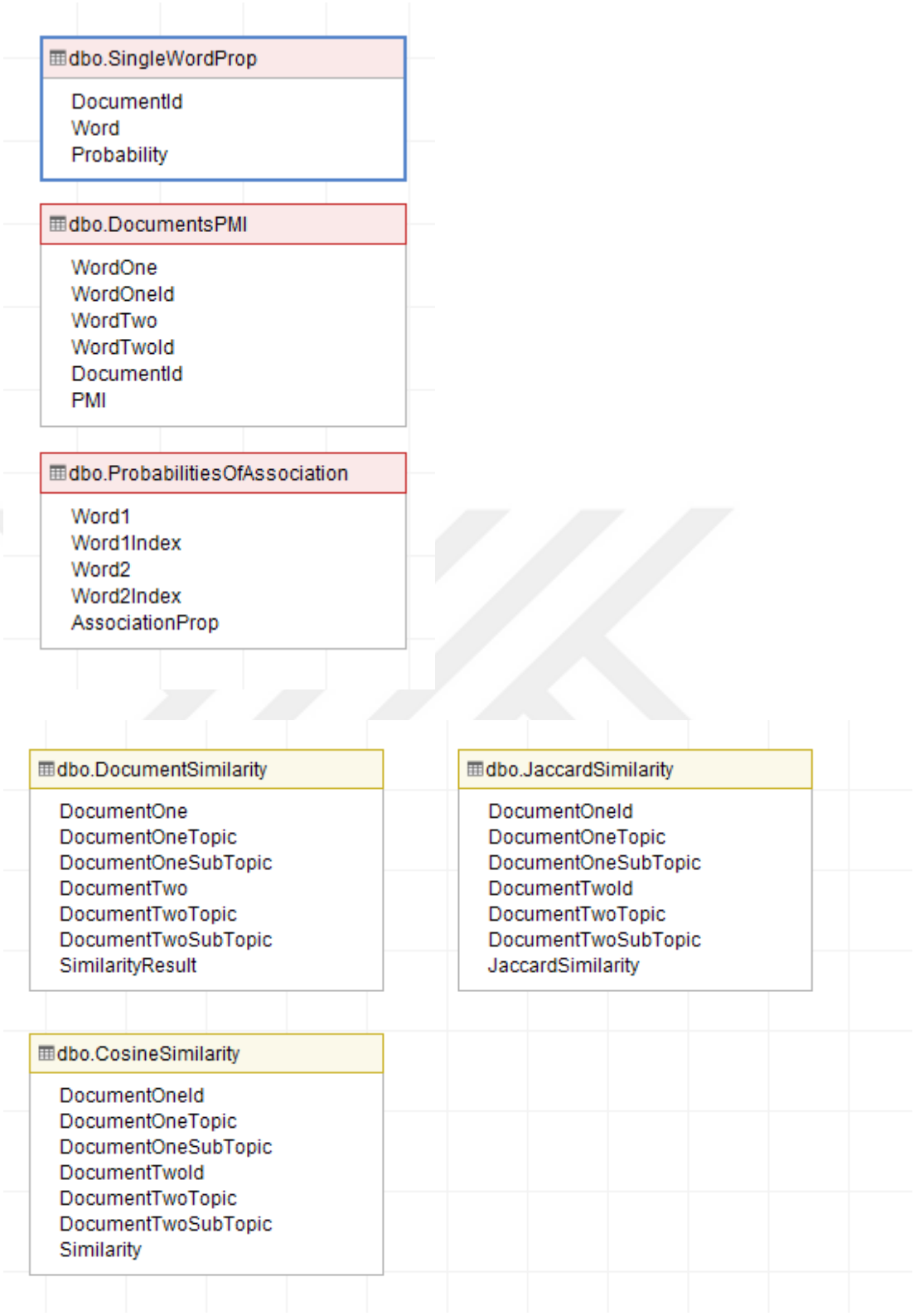
```
                ctext("Class1 için PMI Değeri:" + str(classonePMI), "red")
                ctext("Class2 için PMI Değeri:" + str(classtwoPMI), "green")
                ctext("Class3 için PMI Değeri:" + str(classthreePMI),
"yellow")
                ctext(str(newDocument) + " Class3'e eklenecektir", "blue")
                print("\n")
                class3.append(doc)
                alldocuments.remove(doc)
                newCentroidVectorClass3                                    =
UpdateCentroid(newDocument,newCentroidVectorClass3,class3)
                if(doc[2] == class3Type):
                    correctClass3 = correctClass3 +1

def CalculateSimilarityClass(centroid,newDocument):
    totalSimilarity = 0
    #Merkez noktası ile karşılaştırılacak olan dokümanın vektörü
    newDocumentVector = GetDocumentVector(newDocument)
    #Rastgele seçilen merkez noktasının vektörü
    centroidVector = GetDocumentVector(centroid)
    for i,val in enumerate(centroidVector):
        if centroidVector[i] == "1" and newDocumentVector[i] == "1":
            pmi_value = GetPMIValue(i,i)
            if(pmi_value != None):
                totalSimilarity += pmi_value[0]
    return totalSimilarity

#Algoritma Adımları

class1 = []
class2 = []
class3 = []
newCentroidVectorClass1 = []
newCentroidVectorClass2 = []
newCentroidVectorClass3 = []


alldocuments = GetDocumentList()

#secim = input("Manuel seçim : o  Rastgele Seçim : r \n")


secim ='r'
esikdegeri = input("Öbek Eşik % Değeri :")
if secim == "r" :

    rastgeleClass1 = random.choice(alldocuments)
    class1Type = rastgeleClass1[2]
    alldocuments.remove(rastgeleClass1)

    rastgeleClass2 = random.choice(alldocuments)
    class2Type = rastgeleClass2[2]
    alldocuments.remove(rastgeleClass2)

    rastgeleClass3 = random.choice(alldocuments)
    class3Type = rastgeleClass3[2]
    alldocuments.remove(rastgeleClass3)
elif secim == "o":
    rastgeleClass1 = input("Class 1 için merkez noktası olarak seçilecek
doküman")
    alldocuments.remove(rastgeleClass1)
    rastgeleClass2 = input("Class 2 için merkez noktası olarak seçilecek
doküman")
```

```
    alldocuments.remove(rastgeleClass2)
    rastgeleClass3 = input("Class 3 için merkez noktası olarak seçilecek
doküman")
    alldocuments.remove(rastgeleClass3)


class1.append(rastgeleClass1)
class2.append(rastgeleClass2)
class3.append(rastgeleClass3)

ctext("Rastgele seçilen ve Class1'e eklenen ilk doküman :" +
str(rastgeleClass1),"magenta")
ctext("Rastgele seçilen ve Class2'ye eklenen ilk doküman :" +
str(rastgeleClass2), "yellow")
ctext("Rastgele seçilen ve Class3'e eklenen ilk doküman :" +
str(rastgeleClass3),"cyan")

centroid1 = class1[0][0]
centroid2 = class2[0][0]
centroid3 = class3[0][0]

ctext("Öbekleme başlıyor...",'magenta')
while len(alldocuments) > 0:
    newDocument = PickRandomDocument()
    ctext("Öbeklenecek Doküman:" + str(newDocument),"yellow")
    print("\n")
    if len(newCentroidVectorClass1) == 0 and len(newCentroidVectorClass2) ==
0 and len(newCentroidVectorClass3) == 0:
        ctext("İlk kez hesaplama yapılıyor...CalculateWithCentroidFirstTime
methodu çağrıldı","white")
        print("\n")
        CalculateWithCentroidFirstTime(centroid1,                centroid2,
centroid3,newDocument)
    else:
        CalculateWithCentroid(newCentroidVectorClass1,
newCentroidVectorClass2,newCentroidVectorClass3,newDocument)

totalDocumentCount = GetDocumentListCount()
ctext("Toplam Doküman Sayısı : " + str(int(totalDocumentCount[0])),"blue")
ctext("Class1 e eklenen doküman sayısı:" + str(len(class1)),"cyan")
ctext("Class2 e eklenen doküman sayısı:" + str(len(class2)),"cyan")
ctext("Class3 e eklenen doküman sayısı:" + str(len(class3)),"cyan")

ctext("Class1'e Eklenen ve Doğru Sınıfta Olan Doküman Sayısı : " +
str(int(correctClass1+1)),"green")
ctext("Class2'e Eklenen ve Doğru Sınıfta Olan Doküman Sayısı : " +
str(int(correctClass2+1)),"green")
ctext("Class3'e Eklenen ve Doğru Sınıfta Olan Doküman Sayısı : " +
str(int(correctClass3+1)),"green")


ctext("-----------------------------------------------------------------
","yellow")

ctext("Class1 için bilenen sınıf: " + class1Type)
ctext("Doğru Hesaplanan Class1 doküman yüzdesi: %" + str(int(correctClass1+1)/
int(len(class1)) *100),"green")

ctext("Class2 için bilenen sınıf: " + class2Type)
ctext("Doğru Hesaplanan Class2 doküman yüzdesi: %" + str(int(correctClass2+1)/
int(len(class2)) *100),"yellow")
```

```
ctext("Class3 için bilenen sınıf: " + class3Type)
ctext("Doğru Hesaplanan Class2 doküman yüzdesi: %" + str(int(correctClass3+1)/
int(len(class3)) *100),"blue")

ctext("-----------------------------------------------------------------
","blue")
```

EK B. Veritabanı Diyagramı




dbo.SingleWordProp
DocumentId
Word
Probability
dbo.DocumentsPMI
WordOne
WordOneId
WordTwo
WordTwoId
DocumentId
PMI
dbo.ProbabilitiesOfAssociation
Word1
Word1Index
Word2
Word2Index
AssociationProp
dbo.DocumentSimilarity
DocumentOne
DocumentOneTopic
DocumentOneSubTopic
DocumentTwo
DocumentTwoTopic
DocumentTwoSubTopic
SimilarityResult
dbo.JaccardSimilarity
DocumentOneId
DocumentOneTopic
DocumentOneSubTopic
DocumentTwoId
DocumentTwoTopic
DocumentTwoSubTopic
JaccardSimilarity
dbo.CosineSimilarity
DocumentOneId
DocumentOneTopic
DocumentOneSubTopic
DocumentTwoId
DocumentTwoTopic
DocumentTwoSubTopic
Similarity

ÖZGEÇMİŞ

Adı Soyadı : Ahmet Görkem Özdoğan

Doğum Yeri ve Yılı : Adana, 02/11/1982

Medeni Hali : Evli

Yabancı Dili : İngilizce

E-posta : agozdogan@gmail.com

Eğitim Durumu

Lise : Antalya Çağlayan Lisesi, 1999

Lisans : Kocaeli Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 2008

Yüksek Lisans : İstanbul Ticaret Üniversitesi,
Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı

Mesleki Deneyim

Tecnicas Reudinas	2007-2007
RfLogy	2008-2008
Axis-IT	2010-2011
Turks.Net A.Ş.	2011-2014
Turkiye Finans Katılım Bankası	2014-…

Yayınları

A. G. Özdoğan, K. O. Akgün and C. Kaya, "Service governance in service oriented architecture," 2017 International Conference on Computer Science and Engineering (UBMK), Antalya, 2017, pp. 195-200.
doi: 10.1109/UBMK.2017.8093374

A. G. Özdoğan and A. Toprak, "The Effect Of Gamification in Information Technologies Projects," 2018 3rd International Conference on Computer Science and Engineering (UBMK), Sarajevo, 2018, pp. 36-40.
doi: 10.1109/UBMK.2018.8566557

A. G. Özdoğan, A. Toprak, "Kurumsal Şirketlerde DevOps Süreçlerinin Zorlukları", 21. Akademik Bilişim Konferansı AB 2019, Ordu

A. G. Özdoğan and M. Turan, "Metin Madenciliği Kullanarak İngilizce Doküman Sınıflama", İstanbul Ticaret Üniversitesi Teknoloji ve Uygulamalı Bilimler Dergisi, 2019, İstanbul (Tez çalışması sonucu yapılan yayın)