DESTEK SİSTEMLERİNE CHATBOT ENTEGRESİ

-

MERVE GÜLAÇTI

ERDEM ŞAHİN

FURKAN BERK AKDAŞ

KIRKLARELİ ÜNİVERSİTESİ

2020

Onaylayan _____

Denetim Kurulu Başkanı

_____

_____

_____

Derece Vermeye Yetkili

Program_____

Tarih_____

KIRKLARELİ ÜNİVERSİTESİ

Önsöz

Bu tez çalışmasında, chatbot'un ne olduğuna, kullanım alanlarıyla birlikte ne şekilde faaliyet gösterdiğine, literatürel araştırmasına ve chatbot - insan etkileşimi incelenecektir. Lisans projesi olarak hazırlanan bu çalışmanın konusunu Kırklareli Üniversitesi öğrenicilerinin sorularının daha hızlı ve doğru bir şekilde cevaplana bilmesi adına yapılmıştır. Bu çalışma Kırklareli Üniversitesi Mühendislik Fakültesi Yazılım Mühendisliği bünyesinde gerçekleştirilmiştir.

Çalışma, üç bölümden oluşmaktadır. Birinci bölümde chatbot nedir gibi sorular açıklanarak chatbot hakkında bilgilendirmeler bulunmaktadır. İkinci bölümde KLUBOT fikrinin ortaya çıkışı, uygulamanın gelişim adımları, uygulama tasarımı için gerekli algoritmalar ve bu algoritmaların nasıl çalıştığı ele alınmıştır. Üçüncü ve son bölümümüzde son kullanıcı testleri sonucu elde ettiğimiz verilerin incelenerek, kullanıcıların chatbot uygulamamız hakkındaki görüşlerine yer verilerek "kullanıcı- proje etkileşimi" incelecektir.

Çalışmalarda bilgi ve tecrübesiyle bizi aydınlatan proje danışmanımız **Dr.Öğr.Üyesi Fatih AYDIN** 'a, Python programlama dilini kullanmamızı sağlayan **Dr.Öğr.Üyesi Murat Olcay ÖZCAN** 'a Doğal Dil İşlemeyi öğreten ve her konuda sorularımızı yanıtsız bırakmayan Yazılım Mühendisliği Bölüm Başkan Yardımcımız Sayın **Dr. Öğr. Üyesi Edip Serdar GÜNER** 'e teşekkürü borç biliriz.

ÖZET

DESTEK SİSTEMLERİNE CHATBOT ENTEGRESİNİN İNCELENMESİ -

Merve GÜLAÇTI, Erdem ŞAHİN, Furkan Berk AKDAŞ

Denetim Kurulu Başkanı: DANIŞMAN : Dr. Öğr. Üyesi Fatih AYDIN

Bilgi ve iletişim teknolojilerinin gelişimi ile birlikte her alanda olduğu gibi, iletişim alanındaki kavramlar ve uygulamalar da çok hızlı bir şekilde değişmekte ve ilerlemektedir. Dijitalleşen dünyada yaşanan söz konusu kapsamlı ve hızlı dönüşüm yapay zeka, sanal gerçeklik gibi kavramların ortaya çıkmasına neden olmuştur. Dijital dünyadaki bu dönüşüm halkla ilişkiler uygulamalarında ve eğitim alanlarında da yeni iletişim ortamlarının doğmasına vesile olmuştur. Bunlardan biri de yapay zekânın en önemli kullanım şekilleri arasında yer alan, insan ve teknolojinin birlikteliğinden doğan "chatbot"lardır. Yapay zekânın gelişmesiyle birlikte ve kurumlar açısından bireyin öneminin artmasıyla, bu yeniçağa ayak uydurmak, onları anlamak ve sorularını etkili bir şekilde yanıtlamak için önemli olacaktır. Bu değişimlerle birlikte toplumun yapısı değişmekte ve her toplumun bu değişime uyumunu zorunlu kılmaktadır.

Bu çalışmamızdaki amaç, odak noktası olarak Kırklareli Üniversitesi bünyesindeki öğrencilerin veya üniversitemiz ile ilgili herhangi bir etkileşimi olan/olacak kişilerin aklındaki soru işaretlerini “Chatbot” yardımı ile yok etmektir.

Anahtar Kelimeler: Chatobot, String Benzerlik, Morfolojik Analiz

ABSTRACT

INVESTIGATION OF CHATBOT INTEGRATION ON SUPPORT SYSTEMS

With the development of information and communication technologies, as in every field, concepts and practices in the field of communication are changing and advancing very quickly. The comprehensive and rapid transformation in the digital world caused the emergence of concepts such as artificial intelligence and virtual reality. This transformation in the digital world has led to the emergence of new communication environments in public relations practices and education. One of them is the "chatbots", which are among the most important uses of artificial intelligence and born out of the unity of people and technology. With the development of artificial intelligence and the importance of the individual for institutions, it will be important to keep up with this new age, to understand them and to answer their questions effectively. With these changes, the structure of the society is changing and it obliges each society to adapt to this change.
The aim of this study is to eliminate the question marks in the minds of students of Kırklareli University or students who will interact with our university with the help of "Chatbot".

Keywords: Chatobot, String Similarity, Morphological Analysis

İÇİNDEKİLER TABLOSU

İçindekiler

ŞEKİL LİSTESİ

Numara *Sayfa*

RESİM LİSTESİ SAYFA

TABLO LİSTESİ *SAYFA*

TEŞEKKÜR

Çalışmamızın her aşamasında bize destek olan, bilgi ve deneyimleri ile bize yol gösteren danışman hocamız Sayın Dr.Öğr.Üyesi Fatih AYDIN ’a, yoğun çalışma programına rağmen bizim sorularımızı yanıtsız bırakmayarak elinden geldiğince bize yardımcı olan Yazılım Mühendisliği Bölüm Başkan Yardımcımız Sayın Dr. Öğr. Üyesi Edip Serdar GÜNER ‘e , ayrıca bu süreçte bizi yalnız bırakmayıp desteğini hissettiren Öğr. Gör. Fatih Bal ‘a çok teşekkür ederiz.

Çalışmamız boyunca her koşulda maddi ve manevi olarak yanımızda olan ailelerimize teşekkür ederiz.

Erdem ŞAHİN

Merve GÜLAÇTI

Furkan Berk AKDAŞ

KIRKLARELİ, 2020

SÖZLÜK

CHATBOT. Chatbot, internet üzerinden insan kullanıcılarla konuşmayı simüle etmek üzere tasarlanmış bir bilgisayar programıdır.

ALGORİTMA. Algoritma, belli bir problemi çözmek veya belirli bir amaca ulaşmak için tasarlanan yol.[1][2] Matematikte ve bilgisayar biliminde bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler kümesidir. Genellikle bilgisayar programlamada kullanılır ve tüm programlama dillerinin temeli algoritmaya dayanır.

GOOGLE ASİSTAN. Google Asistan, Siri ve Cortana'sı gibi bir sesli dijital asistan uygulamalarından bir tanesidir. Android ve iOS platformlarında kullanılan Google Asistan'a bir şey sorduğunuz zaman o da size yanıt vermektedir.

GİRİŞ

Bilgi ve iletişim teknolojilerinde her geçen gün yarattığı değişim eğitim alanındaki iletişimde ve iletişim uygulamalarının tümünde yeni iletişim ortamları ortaya çıkarmıştır. Dijitalleşen çağda, dijitalleşen iletişim her alanda önemli dönüşümlere yol açmaktadır. Söz konusu alanlar arasında yer alan eğitimde iletişim süreci de gelişmekte, dijitalleşmektedir.

İçinde bulunduğumuz çağda dijital bir devrim yaşanmaktadır. İnsana benzer robotların, akıllı ve kendi kararlarını alabilen makinelerin, dijital asistanların var olduğu devrimden söz edilmektedir. Teknoloji ile birlikte yapay zeka tüm dünyayı etkileyerek değişimlere sebep olmaktadır. Bu devrim devam ederken karşımıza yeni bir kavram "Süper Akıllı Toplum" kavramı çıkmaktadır. Süper Akıllı Toplum kavramı insanların, robotlar ve makinelerle iletişiminin daha etkili olmasını sağlamak ve insanların çıkarlarını ön planda tutarak birtakım çözüm önerileri sunmaktadır.

Günümüzde yapay zekâ teknolojilerinin kullanım alanlarından olan eğitim, sağlık, turizm, bankacılık gibi sektörler dışında tüm bilim dalları da yapay zekâ ile ilgilenmektedir. Günümüzde birçok farklı kullanım alanlarındaki değişimler ve gelişmeler sonucunda kullanıcılara dönütler daha farklı bir davranış sergilenmektedir.
Günümüzde anlık mesajlaşmalar daha çok yerini "Chatbot"lara bırakmaktadır. Kullanıcıların talep ve sorunları her geçen gün artmakta, değişmekte ve hatta şirketlerin bu beklentileri karşılayabilmesi için pazarlama stratejilerinde, kanallarında değişikliklere gidip yenilikleri takip etmesi gerekmektedir.

Bu tez çalışmasının birinci bölümünde; chatbot'un ne olduğuna, kullanım alanlarıyla birlikte ne şekilde faaliyet gösterdiğine, ayrıca literatürel araştırmasına yer verilecektir. Ardından chatbot - insan etkileşimi incelenecektir. KLUBOT uygulamamızın amacına, hangi ihtiyaç/ihtiyaçlar doğrultusunda ortaya çıktığına, ardından Kırklareli Üniversitesi için neden böyle bir projenin geliştirildiğinden bahsedilecektir.

İkinci bölümümüzde ise, uygulamamızın gelişim aşamalarında karşılaştığımız zorluklardan, etkilendiğimiz diğer uygulamalardan bahsedilecektir. Projenin teknik kısmından yani kod yapısı ve algoritmalarından söz edilecektir. Ardından araştırmalarımızda karşılaştığımız kullanıma elverişli algoritmalara yer verilecektir. Daha sonra üniversite verileri ile uygulamanın nasıl entegre edildiğine yer verilecektir.

Üçüncü ve son bölümümüzde; son kullanıcı testleri sonucu elde ettiğimiz verilerin incelenerek, kullanıcıların chatbot uygulamamız hakkındaki görüşlerine yer verilerek "kullanıcı- proje etkileşimi" incelecektir.

1.BÖLÜM: GELİŞEN TEKNOLOJİDEKİ YENİ DOSTLARIMIZ:

CHATBOTLAR

Tarih boyunca insanlar birbirleriyle bir şekilde iletişim halinde olmuş. Belirli yöntemler denenerek insanlar aralarındaki iletişimi kolaylaştırmak için çeşitli yollar denemiştir. Bu tarihsel sürece bakıldığında seslerle, ateş ve dumanla başlayan haberleşme, yerini sırasıyla güvercin, telgraf, matbaa, telefon ve internete bırakmıştır. Ve dolayısıyla internet çağının insanoğlunun hayatına girişiyle bazı iletişim standartları değişti. İnsanlar artık internet üzerinden görüş ve bilgi paylaşımlarını rahatlıkla yapabilir hale geldi. Ayrıca internet ile birlikte hayatımızda pek çok alanda değişimler yaşandı. Mesela insanoğlu artık bir noktadan bir yere giderken çevredeki insanlarla iletişim kurmak yerine, internet üzerinden bir navigasyon uygulamasıyla rahatlıkla amaçladığı yere varabiliyor.

Bu bölümde, chatbot'un ne olduğuna, kullanım alanlarıyla birlikte ne şekilde faaliyet gösterdiğine, ardından Chatbot - insan etkileşimine yer verilmiştir. Ayrıca literatürel araştırmasından söz edilmiştir.

1.1 Chatbot Nedir ?

Chatbotun Türkçe kelime karşılığına baktığımızda karşılık olarak "Sohbet Botu" olduğunu görürüz.
Chatbotlar, kullanıcıların dijital ortamda mesajlaşma yoluyla bir insan ile yazışır gibi iletişim kurdukları ve bir konuda bilgi alma, işlem yapma gibi çeşitli amaçlarla kullandıkları sohbet robotlarıdır.
Chatbotlar günlük hayatımızda en yaygın olarak müşteri hizmetlerinde kullanılmaktadır. Bununla birlikte çeşitli sektörde çalışmakta olan destek operatörlerinin ve müşteri temsilcilerinin görevini üstlenmektedir. Artık insanoğlu herhangi bir işlemi gerçekleştiremediğinde yakınındaki bir kişiden yardım almak yerine, bir "bot"tan yardım istiyor.

1.2 Chatbot Tasarım

Chatbotların sistem üzerindeki yeteneklerine her geçen gün bir yenisi eklenerek güncel ve hızlı kullanımı amaçlıyorlar. Tasarımsal olarak, kullanıcı için sade, okunabilir ya da duyulup anlaşılabilir bir kullanıcı arayüzü oluşturulmalıdır.

Birçok banka kendi müşteri hizmeti için bir bot tasarlamış, geliştirmiş ve hizmete sunmuş durumda. Bunun beraberinde botların kullanım kolaylığı sayesinde günümüzün içerisinde fark edilmeden yer edinmiş oldu.

Şekil 1: Garanti BBVA/Sesli Asistan Ugi

Şekil 2: Mitsuku Chatbot

Yurt dışından güzel bir örnek: Mitsuku. 2013 ve 2016 yıllarında chatbot dünyasının en prestijli ödülüne sahip bir bottur. Steve Worswick tarafından geliştirilen bot, bilmediği ya da anlamadığı şeyleri çok düzgün bir şekilde ifade edip ne olduklarını soruyor. Dezavantaj diyebileceğimiz tek özelliği ise sadece düzgün cümleleri anlıyor oluşu. Yani "Naber?" yerine "nbr" yazarsanız, Mitsuku sizi anlamıyor. Sanırım Cleverbot'taki gibi bir kirliliğe sebep vermemek için böyle bir kural koyulmuş. Kullanıcıya farklı arayüz seçeneğini de sunuyor.

Bir diğer örneğimiz; Ugi. Ugi, tasarım olarak ve yaratılan tepki sesiyle kullanıcıların sempatisini hemen kazanabiliyor. Ugi, kullanıcının sesli olarak bir gönderi oluşturmasının ardından kullanıcıya en uygun şekilde direkt dönüş sağlıyor. Tasarım ve kullanım olarak gayet kolay ve sempatik olan bu müşteri hizmet bottur.

1.3 Chatbotlar Nasıl Çalışır?

Chatbot teknolojisinin temeli "natural language processing" NLP teknolojisine dayanıyor, bununla birlikte, aynı teknoloji ses tanıma tabanlı Google Now, Apple Siri ya da Microsoft Cortana gibi sanal asistanların yazılımlarında da bulunuyor.

Chatbotlar, kullanıcı tarafından söylenenleri yorumlayarak, ne istediğini anlayan bir dizi karmaşık algoritma ile çalışmaktadır. Böylece kullanıcıdan gelen girdiyi işleyerek ona uygun dönüşü sağlamaktadır.

Hatta günümüzdeki bazı Chatbotlar o kadar gerçekçi ve düzgün cevaplar verebiliyor ki karşınızdakinin bir yazılım programından ziyade insan olduğunu düşünebiliyorsunuz.

KLUBOT'umuzda kullandığımız algoritma yapılarından bazıları şu şekildedir:

- Sequence matcher,
- Cosine similarity,
- Jaccard algoritması

Programımızda kullandığımız algoritmalarımız sayesinde veritabanımızda bulunan sorular ile benzerlikleri karşılaştırılarak uygun cevap yanıtını kullanıcıya döndürmektedir.

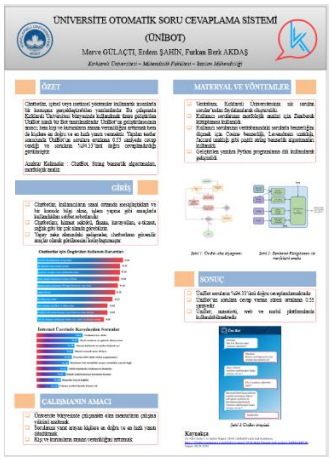
1.4 KLUBOT Nedir?

Projemiz bir "Mühendislik Bitirme Projesi"dir.

Geliştirilme amacı; Üniversitemizin bünyesine destek amaçlı olarak geliştirilmektedir. Üniversitemizin bünyesinde öğrenim görmekte olan öğrencilerin sorularına veyahut üniversitemize geçiş yapmayı düşünen kişilerin/ailelerin sorularına Chatbot programımız üzerinden hızlı bir şekilde yanıt vermektir.Amacımız tamamen üniversitemize katkı sağlamaktır.

Projemizde isimlendirme yaparken KLU Üniversitemizin isim kısaltmasıdır.

BOT sözcüğü ise "Chatbot" uygulaması olduğu için sadece bot sözcüğünü seçtik.



Resim 1: KLU Proje Pazarı KLUBOT Posteri

1.5 Chatbot İnsanlar Arası Etkileşimi Nasıl Etkiliyor?

Chatbotlar kullanım alanları olarak her geçen gün artmaktadır. Ve günlük hayatımızın birer parçası haline gelmiştir. Peki bu durum insanlar arası iletişimi ne şekilde etkilemektedir?

İnsanlar gerek gördükleri durumlarda uygulamalar üzerinden botlardan yardım almaktadır. Çözemedikleri problemleri çözmek için, ya da uygulama üzerinde gerçekleştiremediği bir işlem için bota başvurmaktadır. Müşteri hizmetlerine danışması gereken bir durum olduğunda karşılaştığı bir kişi değil bottur. Bunun gibi daha birçok gerekli durumlarda kullanıcılar "Bot"lara başvurmaktadır.

Peki hayatımıza Chatbotlar girmeden önce, günlük hayatımızda bu kadar yer edinmeden önce insanlar yukarıdaki gibi problemlerle karşılaştıklarında neler yapıyordu ?

Bankacılık işlemleri için mecburi olarak banka şubesine giderek, bankadaki çalışana problemini anlatarak çözüm yolu buluyordu. Yada bir müşteri hizmetleri işlemini gerçekleştirebilmek için telesekreter ile görüşme sağlaması gerekmekteydi.

Fakat Chatbotların hayatımıza girmesiyle çoğu alışkanlığımızla birlikte insanlara olan ihtiyaçlarda birtakım azalmalar meydana geldi. Chatbotlar da günlük kullanımdaki dili anlayıp, karşılık olarak bir yanıt döndürebildiği için, kullanıcılar durumu yadırgamadı.

Böylece hizmet sektörü gibi sektörlerde insan yükü azaltılmış oldu. Ve kullanıcılara dönüt hızlı gerçekleştiği için de sevildi.

2.BÖLÜM: HIZLI YARDIMCILAR

CHATBOTLAR NASIL YAPILIYOR?

Önceki bölümümüzde "Chatbot nedir?, insanlar için Chatbot nedir?, KLUBOT nedir ve ne için kullanılmaktadır?" gibi sorulara yanıt olması için anlatım yapıldı.

Bu bölümümüzde uygulamamızın gelişim aşamalarından, Kırklareli Üniversitesi için neden böyle bir projenin geliştirildiğinden bahsedilmiştir. Proje için algoritma araştırmasına, en verimli 3 algoritma hakkında bilgi verilmiştir. Proje için verilerin nasıl toplandığından ve son olarak; ilham aldığımız uygulamalara değinilmiştir.

1. GELİŞİM ADIMLARI
   - Projemiz bir bitirme projesi olduğu için projemizin fikri belirlendi. (Birinci bölümde bahsedildi.)
   - Proje üyeleri ile birlikte proje fikrinin analizi yapıldı. Chatbotu en kaliteli şekilde geliştirmek için nasıl bir gelişimin kaydedilmesi hakkında beyin fırtınası yapıldı.
   - Daha sonra projemiz için gerekli veriyi nasıl toplamamız gerektiği hakkında araştırma gerçekleştirildi.
     - Gerekli verilerin elde edilmesinin ardından, soruların "eş anlamlıları" türetilerek kaydedildi.
   - Verilerin kaynağı belirlendikten sonra, kodlama kısmı için belirli araştırmalar gerçekleştirildi.
     - Java ve Python programlama dilleri ile makine öğrenmesi(NLP) için kodlama yapmak daha rahattır.
     - Bizim tercihimiz ilgi alanımız olduğu için Python'dan yana oldu.
   - Kodlama adımında belirli zaman aralıkları ile bazı testler gerçekleştirildi. Programın çalışması ve gelişiminin durumu değerlendirildi.
     - Kod adımında bizim verimize uygun optimum algoritmalar belirlendi.
     - Verinin programa entegresi gerçekleştirildi.
     - Üniversitemizin bize sunduğu imkanlar doğrultusunda "Kırklareli Üniversitesi Web Server'ından" minimum özelliklere sahip, kullanıcı testlerinin gerçekleştirilmesi amacıyla alan talep edildi.

- Program geliştiricileri, programın testini yaptılar.
- Değerlendirme sonrası gerekli algoritma değişiklikleri yapıldı.

- Kodlama üzerinde optimum seviyeye ulaşıldıktan sonra, programımıza uygun bir "kullanıcı arayüzü" tasarlandı. Ve program ile entegre edildi. Test edildi.
- Belirlenen 10 kişinin program için testerlık yapması istendi. Elde edilen doğru yanıt verme oranları kaydedildi.
  - Oranlar bir Excel Dosyası üzerinde listelendi. Liste, doğru yanıt verme oranlarıyla renklendirme yapıldı.
  - Algoritmaların çalışma oranları renkler sayesinde, rahatlıkla belirlendi.

2. NEDEN BU FİKİR?

Kırklareli Üniversitesinin öğrencilerine, öğrenci adaylarına, ailelerine yol gösterici ve kişilerin sorularına yanıt verebilecek bir sistemi bulunmamaktaydı. Soru sormak isteyen bir kişi kurumun ilgili alanını telefon ile arayarak, sorusuna yanıt bulmaya çalışıyordu. Bu süreç anlatıldığı gibi kısa ve kolay olmuyordu genellikle. Genellikle bir kurum ile telefon bağlantısı kurma süreci zahmetli ve zordur.

Bu nedenle üniversitenin hızlı yanıt sevisi gibi bir uygulamaya ihtiyacı vardı. Bu eksiklik dolayısıyla üniversite için kullanışlı ve hızlı bir "Chatbot" projesi geliştirme fikri ortaya çıktı.

İsim olarak da kısa ve sade bir isimlendirme seçerek, projemize "KLUBOT" ismini verdik.

3. İLHAM ALDIĞIMIZ PROGRAMLAR

Bilinen üzere Chatbot kullanımı gündelik hayatımızda teknolojinin gelişmesiyle çokça yer almaktadır. Kullanıcılar Chatbotlar ile rahatlıkla iletişim kurup, kısa sürede sorunlarına çözüm bulmaya başladı. Bu nedenle akıllı telefon ve bilgisayar kullanıcıları Chatbot kullanmayı sevdi.

- En İyi 6 Chatbot Uygulaması
  1. Mitsuku
  2. Cleverbot
  3. ALICE
  4. Insomnobot
  5. Woebot
  6. Getir – Messenger(Yerli Chatbot)

Kullanıcılar için en önemli noktalardan birisi, kullanılan programın sade ve anlaşılır bir kullanıma sahip olmasıdır. Bizim de amacımız bu yönde bir program geliştirip, kullanıcıları memnun etmek.

4. HANGİ ALGORİTMA NE YAPIYOR?

1. Levenshtein Mesafe Algoritması (Levenshtein Distance)

Bu yöntem 1965 yılında Rus Matematikçi Vladimir Levenshtein (1935-2017) tarafından icat edildi. Mesafe değeri, bir dizeyi (kaynak) diğerine (hedef) dönüştürmek için gereken minimum silme, ekleme veya değiştirme sayısını açıklar. Levenshtein mesafesi ne kadar büyük olursa, harfler arasındaki fark o kadar büyük olur. Örneğin, "test" ten "test" e kadar Levenshtein mesafesi 0'dır, çünkü hem kaynak hem de hedef dizeler aynıdır. Dönüşüm gerekmez. Aksine, "test" ten "text" Levenshtein mesafesi 2'dir - "test" i "text" çevirmek için iki değişiklik yapılmalıdır.

Oyun- Ayan kelimeleri arasındaki mesafe 2 ‘dir çünkü ilk ve ikinci o harfleri a harfi ile değişmiştir.
Arap – araba kelimeleri arasındaki mesafe 2’dir çünkü p harfi b ile değişmiş ve a harfi eklenmiştir

Algoritma değişim değerini bulmak için iki boyutlu bir dizi (matris) üzerinde kelimelerin farklı olan harfleri için değer artırımına gider.

Örneğin karşılaştıracağımız dizgiler (string) “Cuma” ve “Pazar” kelimeleri olsun.

		C	U	M	A
	0	1	2	3	4
P	1	1	2	3	4

A	2	2	2	3	3
Z	3	3	3	3	4
A	4	4	4	4	3
R	5	5	5	5	4

Şekil 3: Levenshtein Mesafe Algoritması Çalışma Örneği

Yukarıdaki tabloda, yeşil renkle gösterilen satır ve sütun, boş bir dizgi ile kelimenin karşılaştırılmasıdır. Örneğin boş bir kelime ile CUMA kelimesi karşılaştırılınca her har için ekleme yapılacak ve neticede CUMA kelimesinin uzunluğu olan 4 değerine ulaşılacaktır (satırın sonu 4 sayısı ile bitmiştir).

Yeşil satır ve sütunun altında ve sağında kalan alan ise teker teker harflerin karşılaştırıldığı bölümdür.

Örneğin C harfi ile başlayan sütundaki değerler, sadece C harfinden oluşan bir kelimenin PAZAR kelimesi ile karşılaştırılmasını adım adım gösterir. Buna göre M harfiyle başlayan sütundaki karşılaştırma CUM kelimesi ile PAZAR kelimesinin her harfinin karşılaştırılmasıdır. Bu sütundaki değerler (tabloda sarı renktedir) aşağıdaki şekilde okunabilir:

P-CUM : 3 , P harfini C ile değiştirip UM eklendiği için toplam 3 işlem gerekir

PA-CUM: 3, P harfi C ile, A harfi U ile değiştikten sonra M harfi eklenmiştir

PAZ-CUM : 3 , P harfi C ile, A harfi U ile ve Z harfi M ile değişmiştir, toplam maliyet 3'tür.

PAZA-CUM: 4, P harfi C ile, A harfi U ile ve Z harfi M ile değişmiştir, ilave olarak A harfi eklenmiştir.

PAZAR – CUM: 5, P harfi C ile, A harfi U ile ve Z harfi M ile değişmiştir ilave olarak A ve R harfi eklenmiştir.

Levenshtein mesafesi hesaplanırken, dizideki hesaplanmakta olan hücrenin üstündeki ve solundaki değerlerden faydalanılır. Herhangi bir hücre için aşağıdaki durum geçerlidir:

		Harf1 …
	A	B
…	C	D

Tablo 1: Levenshtein mesafesi hesaplanması

D hücresinin değeri, satırdaki ve sütundaki kelimelerin karşılaştırmasını yaparken, Harf1 ve Harf2 harflerini karşılaştırır. Bu karşılaştırmada 3 ihtimal bulunur.

Harf1 ile Harf2 eşittir. Bu durumda D değerine A değeri konulur. Bunun sebebi, A değerinin daha önceki (Harf1 ve Harf2 karşılaştırmasından önceki kelime parçası) mesafeye yeni bir mesafe eklemiyor olmasıdır.Örneğin PARA ve YARA kelimelerini karşılaştırırken PA ve YA durumunda, Harf1 = Harf2 olur. Bu durumda PA-YA maliyeti 1'dir ve bu maliyet P ile Y nin değişmesinden gelen maliyettir, karşılaştırılan A değerleri için ilave bir maliyet olmaz.

		P	A
Y		1	
A			1

Tablo 2: Levenshtein mesafesi hesaplanması 2

Yukarıdaki tabloda görüldüğü üzere A-A karşılaştırmasının değeri üst çaprazdan okunmuştur.

Şayet Harf1-Harf2 karşılaştırma sırasında, harfler birbirinden farklıysa, bu durumda yine çaprazdaki değer alınır (yukarıdaki tabloda, D'nin değeri hesaplanırken A'nın değeri alınır) ve bu değere 1 eklenir.

Kısacası kelime boyları aynı olduğu sürece D değerinin hesabında A değeri kullanılır.

Şayet kelime boyları farklıysa, bu durumda Harf1 karşılığı olarak Harf2 bulunamayacak veya Harf2 karşılığı olarak Harf1 bulunamayacaktır. Her iki durum için de daha önceden bulunan ve kelimelerin son harflerinin karşılaştırmasına 1 eklenir.

Şayet Harf2 yok ve Harf1 varsa, bu durumda D değeri B+1 olarak hesaplanır.

Şayet Harf1 yok ve Harf2 varsa, bu durumda D değeri C+1 olarak hesaplanır.

Daha genel anlamda, bir hücrenin değeri hesaplanırken iki ihtimal bulunur.

Karşılaştırılan harfler eşittir (bu durumda sol çaprazdaki değer kopyalanır)

Harfler farklıdır (bu durumda, hücrenin solu, üstü ve sol çaprazındaki değerlere 1 eklenerek en küçük olanı alınır.
NumPy paketini ve hesaplamaları yapmak için tek bir matrisi kullanan yinelemeli bir sürümdür. Örnek olarak, "test" ve "text" arasındaki düzenleme mesafesini öğrenmek istiyoruz.

Dizelerin uzunluğuna sahip boş bir matris ile başlar. Sıfırdan başlayarak hem ilk satır hem de sütun giderek dizine eklenir:

```
    t e s t
```

```
 [[ 0. 1. 2. 3. 4.]
```

```
t [ 1. 0. 0. 0. 0.]
```

```
e [ 2.  0.  0.  0.  0.]

x [ 3.  0.  0.  0.  0.]

t [ 4.  0.  0.  0.  0.]]
```

Ardından, dizeleri harf-satır ve sütun-bilge ile karşılaştırmak için iki döngü izler. İki harf eşitse, konumdaki yeni değer `[x, y]`konum `[x-1, y] + 1`, konum `[x-1, y-1]`ve konum değeri arasındaki minimum değerdir `[x, y-1] + 1`.

```
[+0.] [+1.]

[+1.] [  ]
```

Aksi takdirde, bu konum değeri arasındaki minimum `[x-1, y] + 1`, konum `[x-1, y-1] + 1`ve pozisyonunda `[x, y-1] + 1`. Yine, bu, sağ alt konumda eksik değeri aşağıdaki gibi hesapladığınız ikiye iki alt matris olarak görüntülenebilir:

```
[+1.] [+1.]

[+1.] [  ]
```

İki karakter farklıysa üç olası değişiklik türü olduğunu unutmayın - ekleme, silme ve değiştirme. Son olarak, matris aşağıdaki gibi görünür:

```
     t  e  s  t

 [[ 0.  1.  2.  3.  4.]

t [ 1.  0.  1.  2.  3.]

e [ 2.  1.  0.  1.  2.]

x [ 3.  2.  1.  1.  2.]
```

```
t [ 4.  3.  2.  1.  1.]]
```

Düzenleme mesafesi, aslında sağ alt köşedeki [4, 4] konumundaki değerdir, ki bu aslında 1'dir. Bu uygulama olduğunu unutmayın O(N*M)(Complexity Classes) için zaman N ve M iki dizeleri uzunlukları. Algoritmanın karmaşıklığı, n ve m uzunluğunda iki kelime için, (n+1)x(m+1) işlem gerektirmesidir (+1 değerleri boş kelime ihtimalinden gelmektedir). Bu durumda algoritmanın hem yer hem de zaman karmaşıklığı O(nxm) olarak bulunur.

2. Jaccard İndeksi(Jaccard Index)

Jaccard benzerlik endeksi (bazen Jaccard benzerlik *katsayısı olarak* da adlandırılır), hangi üyelerin paylaşıldığını ve hangilerinin farklı olduğunu görmek için üyeleri iki kümeyle karşılaştırır. Bu, iki veri kümesi için% 0 ila% 100 (0-1) aralığında bir benzerlik ölçüsüdür. Yüzde arttıkça, iki popülasyon daha benzerdir. Jaccard Endeksi, örnek kümeler arasındaki benzerliklerin anlaşılmasında kullanılan bir istatistiktir. Ölçüm, sonlu örnek kümeleri arasındaki benzerliği vurgular ve resmi olarak kesişim boyutunun, örnek kümelerin birleşiminin boyutuna bölünmesiyle tanımlanır. Dizinin matematiksel gösterimi şu şekilde yazılır:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Resim 2: Jaccard Benzerliği Matematiksel gösterimi

Örnekler

Küme notasyonunu kullanarak basit bir örnek: Bu iki küme ne kadar benzer?

- A = {0,1,2,5,6}
- B = {0,2,3,4,5,7,9}

Çözüm : J (A, B) = | A∩B | / | A∪B | = | {0,2,5} | / | {0,1,2,3,4,5,6,7,9} | = 3/9 = 0.33.

Notlar :

1. A'nın kardinalitesi, | A | A kümesindeki elemanların sayısıdır.

2. Ayarlanmış gösterim kullanıyorsanız yanıtı ondalık biçimde bırakmak geleneksel olsa da,% 33,33'lük benzerlik elde etmek için 100 ile çarpabilirsiniz.

Belirsiz gösterimsiz örnek problem: Araştırmacılar iki yağmur ormanında biyoçeşitlilik üzerinde çalışıyorlar. A, B, C, D, E, F olmak üzere altı farklı türden örnekler kataloglar. İki yağmur ormanı arasında iki tür paylaşılmaktadır. Jaccard katsayısı nedir?

Çözüm :

1. Her iki popülasyon arasında iki tür (3 ve 5) paylaşılır.
2. İki popülasyonda 6 benzersiz tür vardır.
3. 2/6 = 1/3
4. 1/3 * 100 =% 33.33.

Yağmur ormanları A ve B% 33 benzerdir.

Jaccard Mesafesi
Benzer bir istatistik olan Jaccard mesafesi, iki setin ne kadar *farklı* olduğunun bir ölçüsüdür . Jaccard endeksinin tamamlayıcısıdır ve Jaccard Endeksinin% 100'den çıkarılmasıyla bulunabilir. Yukarıdaki örnek için, Jaccard mesafesi% 1 - 33.33 =% 66.67'dir.
Ayarlanmış gösterimde, Jaccard Mesafesi için 1'den çıkarın:
D (X, Y) = 1 - J (X, Y)
Ancak, ondalık sayıların yorumlanması daha kolay olduğu için genellikle yüzde olarak dönüştürüldüğüne dikkat edin.

Jaccard indeksi metinler arasındaki ilişkinin ölçülmesi amacıyla geliştirilen ölçülerde (metrics) birisi olan Jaccard indeksini açıklamaktır (jaccard index). Indeks basitçe iki metin üzerinden özellik çıkarımı (feature extraction) yapıldıktan sonra ortak olan özelliklerin sayısının, iki metindeki toplam özellik sayısına bölünmesi ile elde edilir. Bu durum aşağıdaki şekilde formüle edilebilir.

[latex]J(A,B)=frac {left| A cap B right|} {left| A cup B right|} [/latex]

Yukarıdaki formülün çalışmasını iki örnek dizgi (string) üzerinden gösterelim.

Dizgi 1 = “bilgi”

Dizgi 2 = “bilim”

Bu iki dizgi üzerinde, öncelikle özellik çıkarımı (feature extraction) yapıyoruz. Örneğin her harf bir özellik olabilir veya bi-gram kullanabiliriz. Diyelim ki bi-gram kullanmak istedik bu durumda iki dizginin bi-gram değerleri aşağıdaki şekilde olacaktır:

Bi-Gram(Dizgi 1)= {bi,il,lg,gi}

Bi-Gram(Dzigi 2)= {bi,il,li,im}

İki kümenin kesişimi = {bi,il}

İki kümenin birleşimi = {bi, il , lg, li , im , gi}

Formülde yerine koyacak olursak
[latex] J(“bilgi”,”bilim”) = frac 2 6 = frac 1 3 [/latex]
Jaccard indeksinin değerinin yüksek çıkması iki dizginin birbirine daha çok benzediği anlamına gelir. İki dizgi birbirine tamamen eşitse Jaccard indeksi 1 olarak bulunur. İki dizginin hiç ortak özelliği yoksa da değer 0 olarak bulunacaktır. Jaccard indeksi 0 ile 1 arasında değişen bir değer alabilir.

Jaccard Mesafesi (Jaccard Distance)

Jaccard indeksi kullanılarak hesaplanabilen diğer bir değer de Jaccard mesafesidir. Literatürde mesafe (distance) kavramı, yunan alfabesindeki delta sembolü ile gösterildiği için indis olarak delta almış J harfi ile ifade edilir. Bu değer basitçe Jaccard Indeks’inin 1’den çıkarılması ile elde edilir. Buradaki amaç, mesafe değeri olarak benzerlik değerinin tersi değer elde etmektir. Yani birbirine yakın olan dizgilerde, benzerliğin artması ancak mesafenin azalması gerekir. Bu anlamda mesafe, benzerliğin

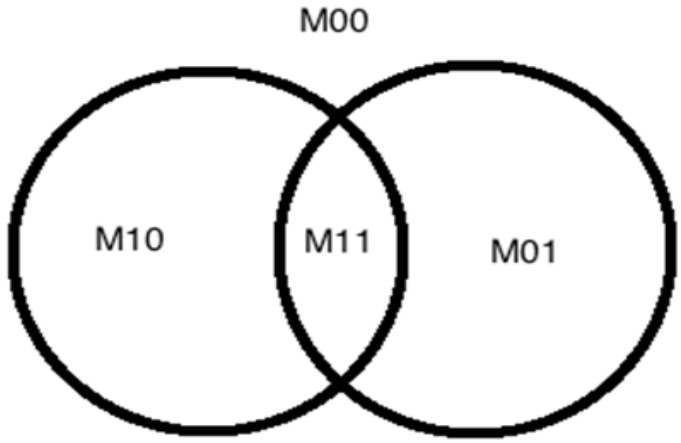
tam tersidir ve Jaccard mesafesi de bu yaklaşıma göre, Jaccard indeksinin tersini alır. Aşağıdaki şekilde hesaplanabilir:

[latex]J_delta(A,B)=1-J(A,B) = 1 – frac {left| A cap B right|} {left| A cup B right|} = frac {left| A cup B right| – left| A cap B right|} {left| A cup B right|}[/latex]

Buna göre, yukarıda verilen örnek dizgiler arasındaki mesafe olarak bulunacaktır. Jaccard indeksinde olduğu gibi 0 ile 1 arasında bir değer olup, Jaccard Indeksinin tam tersi şekilde, birbirinin aynı olan dizgiler için mesafe 0 ve birbiri ile hiç ortak özelliği bulunmayan dizgiler için mesafe değeri 1 çıkacaktır.

Jaccard Benzerlik Katsayısı (Jaccard Similartiy Coefficient)

Sembol olarak J harfi ile gösterilen değerdir. Jaccard indeksi ve mesafesinin metin madenciliği (text mining) gibi ortamlarda kullanılması sırasında, çok sayıda metinin bulunacağı gerçeğinden yola çıkarak geliştirilmiştir. Bu değerin hesaplanması için 4 farklı değerin hesaplanması gerekir.



Yukarıdaki şekilde gösterildiği gibi 2 farklı küme olduğunu kabul edelim. Bu kümelerin ikisinde de olan elemanların sayısı M11, birisinde olup diğerinde olmayan elemanların sayısı M01 veya M10 ve ikisinde de olmayan değerlerin sayısı M00 olarak gösterilecektir.

İkisinde de olmayan eleman, şayet sadece iki metin ile çalışıyorsak, karşılaşılacak bir durum değildir, çünkü bir özelliğin var olması için iki kümeden birisinde olması gerekir. Ancak derlemimizde

(corpus) ikiden fazla metin varsa, örneğin 3. Metinde olup mevcut iki metinde bulunmayan özellikler olacaktır. İşte bu özelliklerin sayısı M00 değerini oluşturur.
Buna göre Jaccard benzerlik katsayısı ve Jaccard mesafe katsayısı aşağıdaki şekilde hesaplanabilir:
[latex] J = frac { M11 } { M00 + M10 + M01 }[/latex]

Jaccard mesafe katsayısı (jaccard distance coefficient) ise aşağıdaki şekilde hesaplanabilir:
[latex] J' = frac { M10 + M01 } { M11 + M10 + M01 }[/latex]
Daha önce anlatıldığı gibi J + J' = 1 olacaktır.

3. TF-IDF ile Cosine Benzerliği

TF-IDF kullanarak cümleyi kelimelerine ayrıştırıyoruz ve cosine similarity ile hesaplıyoruz. Kullanıcı tarafından girilen giriş sorgusu ilgili belgeleri eşleştirmek ve puanlamak için kullanılmalıdır. Bunun nasıl çalıştığını göstermek için 3 belge oluşturdum. Bunların üzerinden geçmek için biraz zaman ayırın.

Document 1: The game of life is a game of everlasting learning

Document 2: The unexamined life is not worth living

Document 3: Never stop learning

life learning sorgusu ile yukarıdaki belgelerde bir arama yapalım. Her şeyin nasıl çalıştığını görmek için her adımı ayrıntılı olarak inceleyelim.

1.Adım:Term Frequency(TF)

TF olarak da bilinen Terim Frekansı, bir belgede bir terimin (kelime) kaç kez oluştuğunu ölçer.
Aşağıda, belgenin her birindeki terimler ve sıklıkları verilmiştir.

Document 1 için TF

Document1	the	game	of	life	is	a	everlasting	learning
Term Frequency	1	2	2	1	1	1	1	1

Document 2 için TF

Document2	the	unexamined	life	is	not	worth	living
Term Frequency	1	1	1	1	1	1	1

Document 3 için TF

Document3	never	stop	learning
Term Frequency	1	1	1

Şekil 4: 1.TF-IDF ile Cosine Benzerliği Çalışma Prensibi

Gerçekte her belge farklı boyutta olacaktır. Büyük bir belgede, terimlerin sıklığı küçük olanlardan çok daha yüksek olacaktır. Bu nedenle belgeyi boyutuna göre normalleştirmemiz gerekiyor . Basit bir hile, terim frekansını toplam terim sayısına bölmektir. Belge 1 Örneğin terim game oluşur two defa. Belgedeki toplam terim sayısı 10'dur . Bu nedenle normalleştirilmiş terim frekansı 2/10 = 0.2'dir . Aşağıda, tüm belgeler için normalleştirilmiş terim sıklığı verilmiştir.

Document 1 için Normalize Edilmiş TF

Document1	The	game	of	life	is	a	everlasting	learning
Normalized TF	0.1	0.2	0.2	0.1	0.1	0.1	0.1	0.1

Document 2 için Normalize Edilmiş TF

Document2	the	unexamined	life	is	not	worth	living
Normalized TF	0.142857	0.142857	0.142857	0.142857	0.142857	0.142857	0.142857

Document 3 için Normalize Edilmiş TF

Document3	never	stop	learning
Normalized TF	0.333333	0.333333	0.333333

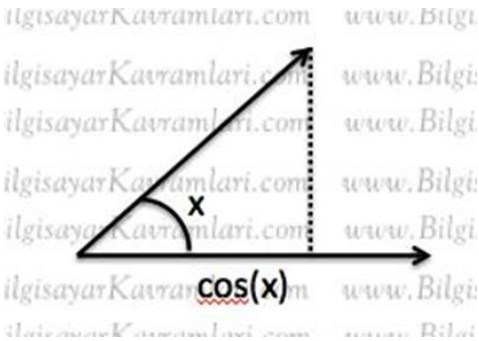
Şekil 5: Şekil 4: 1.TF-IDF ile Cosine Benzerliği Çalışma Prensibi 2

4. Kosinüs Benzerliği (Cosine Similarity)

Kosinüs Benzerliği, bilgisayar bilimlerinin de bir alt çalışma alanı olan, metin madenciliği konularında (text mining) sıkça geçen kosinüs benzerliği (cosine similarity) konusunu açıklamaktır.

Basitçe iki farklı doküman (text) arasındaki benzerliği, trigonometrideki kosinüs (cosine) fonksiyonu üzerinden formülize etmek amaçlanmaktadır.

Metinlerin birer vektör (yöney, vector) olarak düşünüldüğü bu yaklaşımda, iki vektörün birbirine göre olan ilişkisi bir açı ile ifade edilmektedir.



Şekil 6: Kosinüs Benzerliği Çalışma Prensibi

Kosinüs Benzerliği formülü

$$cos(\theta) = \frac{A.B}{||A||.||B||} = \frac{(\sum_{i=1}^{n} Ai.Bi)}{\sqrt[2]{\sum_{i=1}^{n}(Ai)^2}\ \sqrt[2]{\sum_{i=1}^{n}(Bi)\text{^}2}}$$

Burada iki vektör arasındaki kosinüs bağlantısı için iki vektörün çarpımının (dot product) iki vektörün boylarının çarpımına oranı alınmıştır.

Şimdi çarpımı hatırlayalım. Boyutların ayrı ayrı çarpılmasından elde edilen değerlerin toplamıdır. Yani örnek vektörlerimiz için

dot(d1,d2) = [1,1,1,0,0,1].[0,1,1,1,1,1] = (1)*(0) + (1)*(1) + (1)*(1) + (0)*(1) + (0)*(1) + (1)*(1) = 3

sonucu elde edilir. Bu sonuç iki vektörün noktasal çarpımıdır (dot product).

Şimdi vektörlerin boyutlarını hesaplayabiliriz. Bunun için öklit uzayından (euclidean space) faydalanıyoruz ve her boyuttaki değerlerinin karelerini toplayarak toplamın karekökünü alıyoruz.

||d1|| = sqrt((1)^2 + (1)^2 + (1)^2 + (0)^2 + (0)^2 + (1)^2) = 2

||d2|| = sqrt((0)^2 + (1)^2 + (1)^2 + (1)^2 + (1)^2 + (1)^2) = 2.2360679775

olarak boyutları da hesaplanmış oluyor. Son olarak formülde değerleri yerine koyduğumuz zaman:

cos(d1,d2) = dot(d1,d2) / (||d1|| ||d2||) = 3 / (2 x 2.2360679775) = 3 / 4.472135955 = 0.67082039325

olarak iki doküman arasındaki kosinüs benzerliğini bulmuş oluyoruz. Yani bu iki dokümanın kosinüs benzerliğine göre %67'si benzerdir demek yanlış olmaz.

5. Sequence Matcher

İlk olarak sırasıyla iki giriş dizesi str1 ve str2 ile SequenceMatcher nesnesini başlatırız, find_loggest_match (aLow, aHigh, bLow, bHigh) sırasıyla 4 parametre alır aLow, bLow sırasıyla birinci ve ikinci dizenin başlangıç indeksidir ve aHigh, bHigh birinci ve ikinci dizenin uzunluğudur.
find_longest_match (), tuple (i, j, k) adlı bir döndürür, böylece bir [i: i + k] b [j: j + k] 'ya eşit olur, hiçbir blok eşleşmezse bu döndürür (aLow, bLow, 0).
SequenceMatcher, dizi çiftlerini karşılaştırmak için esnek bir sınıftır.sekans elemanları yıkanabilir olduğu sürece herhangi bir tip. Basit algoritma bir algoritmadan önce gelir ve biraz daha meraklıdır 1980'lerin sonunda Ratcliff ve Obershelp tarafından hiperbolik ad "gestalt pattern matching ". Temel fikir bulmak "junk" içermeyen en uzun bitişik eşleştirme dizisi öğeleri.

[Alo: ahi] ve b [blo: bhi] 'de en uzun eşleşen bloğu bulun. İsjunk atlanmışsa veya None ise, find_longest_match () işlevi, [i: i + k] b [j: j + k] değerine eşit olacak şekilde döndürür (i, j, k), burada alo <= i <= i + k <= ahi ve blo <= j <= j + k <= bhi. Bu koşulları karşılayan tüm (i ', j', k ') için k> = k', i <= i 'ek koşulları ve i == i' ise j <= j 'de karşılanır. Diğer bir deyişle, tüm en yüksek eşleşen bloklardan a'da en erken başlayan blokları döndürün ve a ile en erken başlayan en fazla eşleşen blokların tümünün b'de en erken başlayan blokları döndürün.

```
>>> s = SequenceMatcher(None, " abcd", "abcd abcd")
>>> s.find_longest_match(0, 5, 0, 9)
Match(a=0, b=4, size=5)
```

İsjunk sağlanırsa, ilk olarak en uzun eşleşen blok yukarıdaki gibi belirlenir, ancak ek kısıtlama ile blokta önemsiz eleman görünmez. Daha sonra bu blok, her iki taraftaki önemsiz öğeleri eşleştirerek (yalnızca) mümkün olduğunca genişletilir. Böylece, sonuçta ortaya çıkan blok hiçbir zaman önemsiz eşleşme ile eşleşmez, ancak aynı önemsiz, ilginç bir eşleşmenin bitişiğindedir.

İşte öncekiyle aynı örnek, ancak boşlukların önemsiz olduğunu düşünüyoruz. Bu, 'abcd'nin doğrudan ikinci dizinin kuyruk ucundaki' abcd 'ile eşleşmesini önler. Bunun yerine yalnızca 'abcd' eşleşebilir ve ikinci sırada en soldaki 'abcd' ile eşleşebilir:

```
>>> s = SequenceMatcher(lambda x: x==" ", " abcd", "abcd abcd")
```

>>> s.find_longest_match(0, 5, 0, 9)

Match(a=1, b=0, size=4)

Hiçbir blok eşleşmezse, bu geri döner (alo, blo, 0).

6. JARO WİNKLER

Jaro-Winkler işlevleri iki dizeyi karşılaştırır ve dizelerin ne kadar yakın eşleştiği hesaplar.Eşleşme aralığı 0 ile 1 arasında değişiyor. İki dize (‘ ’) eşit olarak karşılaştırılır. Dizeler Unicode olmalıdır diziler ve verilen şekilde karşılaştırılacak; arayan sorumludur büyük / küçük harf kullanımı ve önde gelen / sondaki boşlukları kırpmak.

Jaro Benzerliği aşağıdaki formül kullanılarak hesaplanır.

The Jaro distance d_j of two given strings s_1 and s_2 is

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

where:

- m is the number of *matching characters* (see below);
- t is half the number of *transpositions* (see below).

nerede:

m , eşleşen karakterlerin sayısıdır

t aktarım sayısının yarısıdır

nerede | s1 | ve | s2 | s1 ve s2 dizelerinin uzunluğu

Karakterler aynı ise ve karakterlerden daha uzun değilse eşleştiği söylenir $\left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1$

Yer değiştirme her iki dizgideki eşleşen karakter sayısının yarısıdır, ancak farklı bir sıradadır.

Hesaplama:

s1=”arnab”, s2=”raanb”, olsun, böylece her karakterin eşleştiği maksimum mesafe 1 olur.

Her iki dizenin de eşleşen 5 karaktere sahip olduğu açıktır, ancak sıra aynı değildir, bu nedenle sırayla olmayan karakter sayısı 4'tür, bu nedenle aktarma sayısı 2'dir.

Bu nedenle Jaro benzerliği aşağıdaki gibi hesaplanabilir:

Jaro Similariy = (1/3) * {(5/5) + (5/5) + (5-2) / 5} = **0.86667**

Jaro Winkler Benzerlik(Similarity)

Jaro-Winkler benzerlik iki dizeleri arasında bir dize metrik ölçüm düzenlemek mesafedir. Jaro - Winkler Benzerlik Jaro Benzerlik'e çok benzer. Her ikisi de iki dize öneki eşleştiğinde farklılık gösterir. Jaro - Winkler Benzerlik, dizeler tanımlı bir maksimum uzunluk l'e kadar ortak bir önek olduğunda daha doğru bir cevap veren 'p' önek ölçeğini kullanır.

Örnek:

Input: s1 = “DwAyNE”, s2 = “DuANE”;

Output: Jaro-Winkler Similarity =0.84

Input: s1=”TRATE”, s2=”TRACE”;

Output: Jaro-Winkler similarity = 0.906667

Hesaplama:

Jaro Winkler benzerliği aşağıdaki gibi tanımlanır:

Sw = Sj + P * L * (1 - Sj),

Sj, jaro benzerliği

Sw, jaro-winkler benzerliği

P ölçeklendirme faktörüdür (varsayılan olarak 0.1)

L, eşleşen örneğin maksimum 4 karakter uzunluğudur

s1=”arnab”, s2=”aranb” olsun. İki dizenin Jaro benzerliği 0.933333'tür

Eşleşen örnek uzunluğu 2'dir ve ölçeklendirme faktörünü 0,1 olarak alırız

Formülde ikame;

Jaro-Winkler Benzerlik = 0.9333333 + 0.1 * 2 * (1-0.9333333) = 0.946667

7. Longest Common Subsequence(En uzun Ortak Küme-LCS)

İki küme arasındaki ortak elamanların (sıralı olmak şartıyla) en uzun ortaklığını arar. Örnek:

A-> {X,M,J,Y,A,U}

B-> {M,Z,J,A,W,X,U}

olarak verilmiş olsun. Bu iki kümenin, sırası bozulmadan ortak olan en uzun alt kümesi:

LCS -> {M,J,A,U} olarak bulunur.

Örnek bir çözüm yöntemini inceleyelim:

Problemin ilk akla gelen ve en basit çözümü kümelerden birisini ana küme seçerek, bu kümedeki elemanları teker teker diğer küme ile sınamak olabilir.

8. N-GRAM

Verilen bir dizilimdeki (sequence) tekrar oranını bulmaya yarayan yöntemdir. İsmi n ve gram kelimelerinin birleşiminden oluşmaktadır. Buradaki n, tekrarın kontrol edildiği değerdir. Gram ise bu tekrarın dizilim içerisindeki ağrılığını ifade etmek için kullanılmıştır.
Örneğin bir dizgi (string) içerisindeki n-gram değerini bulmak isteyelim ve buradaki n değeri 2 olsun (n =2)
Yukarıdaki yazının 2-gram değerleri aşağıda verilmiştir:

ww	2	ka	1	ad	1
wb	1	av	1	di	1
bi	1	vr	2	ie	1
il	1	ra	1	ev	1
lg	1	am	1	re	1
gi	1	ml	1	en	1
is	1	la	1	ns	1

sa	2	ri	1	se	1
ay	1	ic	1	ek	1
ya	1	co	1	ke	1
ar	2	om	1	er	1
rk	1	ms	1		

Şekil 7: N-GRAM Algoritması Çalışma Prensibi

Yukarıdaki değerler üzerinde çalıştığımız dizginin (string) 2'şer komşuluktaki alt kümesidir.

Benzer şekilde aynı dizginin 3-gram değerleri aşağıda verilmiştir:

www	1	kav	1	sad	1
wwb	1	avr	1	adi	1
wbi	1	vra	1	die	1
bil	1	ram	1	iev	1
ilg	1	aml	1	evr	1
lgi	1	mla	1	vre	1

gis	1	lar	1	ren	1
isa	1	ari	1	ens	1
say	1	ric	1	nse	1
aya	1	ico	1	sek	1
yar	1	com	1	eke	1
ark	1	oms	1	ker	1
rka	1	msa	1		

Şekil 8: N-GRAM Algoritması Çalışma Prensibi 2

Benzer şekillerde sayı arttırılarak istenilen bir n-gram değeri listelenebilir. Bu değerlerin bazıları literatürde özel olarak isimlendirilmektedir.

Örneğin bu isimlendirme listesi aşağıda verilmiştir:

- 1-gram > unigram
- 2-gram > bigram
- 3-gram > trigram

3'ten büyük değerler için sadece n-gram terimi kullanılır.

n değerine bağlı olarak dizilimin alacağı ihtimal uzayı artmaktadır. Örneğin 29 harfli Türkçe için unigram bir işleme sonucunda 29 farklı alternatife göre dizilinim dağılımı ortaya çıkar. Yani listemizin sonucunda 29 farklı ihtimal ve bu ihtimallerin her birisinin kaçar kere tekrarlandığı bulunur.

n değerini arttırdıkça bu ihtimal uzayı büyür ve yine 29 harfli Türkçe için (Sadece alfabedeki harflerin olduğunu kabul ediyoruz) 29^n olarak bulunabilir.

String(Dizgi)
Bir dilde bulunan ve o dilin tanımlı olan alfabesi içerisindeki sembollerin çeşitli sayılarda ve çeşitli sırada dizilmesi ile elde edilen yazılardır.

Örneğin bir dildeki alfabe aşağıdaki şekilde tanımlı olsun:
$\sum_1 = \{0,1\}$

Buna göre dilimizde sadece "0" ve "1" sembolleri tanımlı demektir. Bu dilde örneğin w_1=0 veya w_2=10101011010 gibi bir dizgi elde etmek mümkündür.

Bir dizginin belirli bir kımını içeren dizgiye ise alt dizgi adı verilir. Örneğin w_3=1011 dizgisi w_2 dizgisinin bir altdizgisidir.

Ayrıca iki dizginin arka arkaya eklenmesine de üleştirme(concatenation) denilir.Örneğin w_1 ile w_3 dizgilerinin üleştirilmiş hali w_4=01011 olur.

9. Smith Waterman Algoritması

Algoritma, dizgi yaslama (string alignment) işlemlerinde kullanılmaktadır.
Dizgi yaslama işlemi (string alignment), basitçe iki dizgiyi alıp bu iki dizgideki ortak alanları bulmayı amaçlar. Bu tip algoritmalar, özellikle gelişen biyobilişim (bioinformatics) çalışmaları ile, genler arasında benzerlik bulunmasının önem kazanmasıyla günümüzde hatırı sayılır bir öneme haizdir.
Algoritma Hakkında
Smith-Waterman algoritması, Needleman-Wunsch algoritmasına oldukça benzemektedir ve bir çeşidi olarak görülebilir. Algoritma 1981 yılında, algoritmaya ismini veren iki bilimadamı tarafından ortaya atılmıştır. Algoritmanın en önemli özelliği, dinamik programlama (dynamic programming) kullanan yapısıdır. Bu tip dinamik programlama kullanan algoritmaların ortak özelliği, ikame masfufu (substitution

matrix, yer değiştirme matrisi) kullanmaları ve boşluk değerleri için bir boşluk fonksiyonu kullanmalarıdır (gap scoring function).

Needleman-Wunsch algoritmasından tek farkı, eksi değerli skorlar yerine 0 kullanıyor olmasıdır.

Algoritmanın Çalışması

Algoritma, bir masfuf (matrix) inşa ederek çalışmaya başları. Bu masfuf aşağıdaki özelliklerdedir:

H(i,0) = 0 , 0 <= i <= m

H(0,j) = 0 , 0 <= j <= n

if(ai == bj)

w(ai,bj) = w (uyar)

else

w(ai,bj) = w (uymaz)

Yukarıdaki matris tanımından anlaşılacağı üzere yaslama işlemi (alignment) yapılan iki dizgi a ve b olarak gösterilmiştir. Ayrıca m, a'nın boyutu ve n de b'nin boyutunu gösteren değerlerdir. i ve j değişkenleri ise, bu matristeki ve dolayısıyla dizgilerdeki kaçıncı elemanın işlendiğini tutan birer değişken sayıdır. H(i,j) değeri, matris üzerindeki bir sayıya işaret etmekte olup bu değer, a[1…i] ve b[1…j] şeklinde gösterilebilecek, a'nin i. ve b'nin j. elemanına kadar olan en yüksek benzerlik skorunu tutmaktadır.

w(x,y) şeklinde gösterilen fonksiyon ise, parametre olarak aldığı iki değer için boşluk skorunu (gap score) tutmaktadır.

Örnek

Örnek olarak, aşağıdaki iki dizgiyi (string) ele alalım:

a= G CACGCTG

b= GA CGCGCG

Bu iki dizgi için n = 8 ve m = 8 olarak hesaplanabilir. Şimdi matrisin oluşturulmasına geçelim:

Bütün ceza durumlarını -1 olarak verelim. Bu ceza durumları şunlardır:

•w(boşluk) = -1

•w(a,-) = -1

•w(-,b) = -1

•w(uymaz) -1

Bütün iyi durumları da +2 olarak alalım ki tek durum uyma durumudur:

•w(uyar) = 2

Şimdi matrisimizi oluşturabiliriz:

	-	G	C	A	C	G	C	T	G
-									
G									
A									
C									
G									
C									
G									
C									
G									

Şekil 9: Smith Waterman Algoritması Çalışma Prensibi

Sonuç olarak herhangi bir satır ve sütun için o ana kadarki en çok benzerlik durumunu veren bir ölçü geliştirilmiş olur.

Örneğin, GCACG ile GACG dizgilerinin (string) mesafesi tablodan 4 olarak okunabilir. Bunun anlamı iki dizgi arasındaki mesafeye en büyük değerlerin seçimi ile ulaşılması halinde, iki dizginin birbirine yaslanması mümkündür.

Tablodaki en yüksek değerler ile sol üst köşeye kadar dönecek olursak:

	-	G	C	A	C	G	C	T	G
-	0	0	0	0	0	0	0	0	0
G	0	2	1	0	0	2	1	0	2
A	0	1	0	2	1	0	0	0	0
C	0	0	2	1	3	2	4	3	2
G	0	2	1	0	2	4	3	2	4
C	0	1	3	2	4	3	5	4	3
G	0	3	2	1	3	5	4	3	5
C	0	2	4	3	5	4	6	5	4
G	0	4	3	2	1	6	5	4	6

Şekil 10: Smith Waterman Algoritması Çalışma Prensibi 2

Yukarıdaki şekilde, iki dizginin birleştirildiği noktalarda tartışmasız tek seçenek olan durumlar kırmızı, aynı yere ulaşmayı sağlayan birden fazla alternatifin olduğu durumlar ise mavi olarak gösterilmiştir. Bu durumda iki dizginin alacağı değerlerin en yüksekleri seçilerek sol üst köşeye ulaşacak birden fazla alternatif bulunmuştur. Bu yolların hepsi aynı sonuca varmaktadır ancak örnek olarak aşağıdaki seçimi yapacak olursak (konunun anlaşılması için bir tanesini seçtiğimizi düşünelim)

	-	G	C	A	C	G	C	T	G
-	0	0	0	0	0	0	0	0	0
G	0	2	1	0	0	2	1	0	2
A	0	1	0	2	1	0	0	0	0
C	0	0	2	1	3	2	4	3	2
G	0	2	1	0	2	4	3	2	4
C	0	1	3	2	4	3	5	4	3
G	0	3	2	1	3	5	4	3	5
C	0	2	4	3	5	4	6	5	4
G	0	4	3	2	1	6	5	4	6

Şekil 11: Smith Waterman Algoritması Çalışma Prensibi 3

Şekilde görüldüğü üzere, iki dizgi arasındaki bağlantı aşağıdaki şekilde kurulabilir:

GCACG

G-ACG

bağlantısı kurulabilir. Bu bağlantıda iki dizgi arasında kopan tek nokta bulunmuş olur.

String Alignment(Dizgi hizalama)

Bu yazının amacı, dizgi yaslaması (string alignment) kavramını açıklamaktır. Literatürde dizilim yaslaması (sequence alignment) olarak da geçen kavram basitçe iki dizgiyi alarak bu dizgilerin birbiri ile olan ilişkilerini bulmayı hedefler.

Günümüzde özellikle biyobilişim (bioinformatics) çalışmalarının artması ile, gen dizilimlerinin karşılşatırılması çalışmalarında önemli bir role sahip olmuştur.

Örneğin iki adet dizgi (string) aşağıdaki şekilde verilmiş olsun:

GCACGCTG

GACGCGCG

Amacımız bu dizgilerin birbiri ile olan yaslamasını (hizalamasını) bulmak. Buradaki kasıt, dizgilerden birisinin üzerine harf ekleme veya çıkarma işlemi yapılarak diğerine ulaşan en kısa yolu bulmaktır.

Örneğin bir dizgi hizalama algoritması olan Hunt-Macllory algoritmasına göre, yukarıdaki iki dizginin hizalanmış hali aşğaıdaki şekildedir:

GCACGCTG–

G-ACGC-GCG

Görüldüğü üzere ikinci ve yedincei harflerde ikinci dizgiden çıkarma yapılıdığı bulunmuş, buna karşılık dokuzuncu ve onunca harflerde de ilk dizgiden çıkarılma yapıldığı kabul edilmiştir.

Bu sayede iki dizgi birbirine göre hizalı hale getirilmiş olur. Aradaki eksik genler bulunarak hizalanır.

Ayrıca dizgi hizalama algoritmaları, dinamik programlama (dynamic programming) kullanımı açısından oldukça elverişlidir.

Dizgi hizalama algoritmaları, iki dizgiyi hizalamak ve eksik ve fazla harfleri bulmak için kullanılır ancak bu amaç, aşağıdaki bazı diğer amaçlara yardım eder:

- En uzun ortak küme problemi (longest common subsequence problem) çözümü
- İki dosya arasında intihal bulunması (pligiarism detection)
- Bir dizgideki hataların bulunması ve düzeltilmesi (error detection , error correction)
- Biyobilişim çalışmaları

- Doğal dil işleme çalışmaları (NLP)

10. NeedLeman Wunsch Algoritması

Bu yazının amacı, bir dizgi (string) işleme algoritması olan Needleman-Wunsch algoritmasını açıklamaktır. Algoritma, basitçe iki dizgi (string) arasındaki yaslama durumunu bulmayı amaçlar. Buna göre iki dizgiden oluşturulan bir ölçüm değeri ile (metric) dizgiler karşılaştırılır ve farklılıklar bulunur. Algoritma yapısal olarak, Smith-Waterman algoritmasına benzemektedir. İki algoritma arasındaki temel farklılık Smith-Waterman algoritmasında tutulan masfuf (matrix) üzerinde eksi değerlerin olamaması ancak Needleman-Wunsch algoritmasında, eksi değerlere izin verilmesidir. Algoritmanın en belirgin özelliklerinden birisi de dinamik programlama (dynamic programming) kullanıyor olmasıdır. Yani iki dizginin karşılaştırıldığı noktaya kadar olan bilgileri bir masufuf (matrix) üzerinde tutularak bu değerler her seferinde yeniden hesaplanmaz.

Algoritmanın çalışmasını örnek olarak kabul edeceğimiz iki dizgi üzerinden gösterelim.

Örnek olarak karşılaştırmak istediğimiz iki dizgi aşağıdaki şekilde verilmiş olsun :

a= G CACGCTG

b= GA CGCGCG

İki dizginin karşılaştırılacağı algoritmada öncelikle algoritmanın sayısal değerlerini belirtmemiz gerekiyor. Bu değerler parametrik olarak değişebilmekle beraber, örneğimizde kullanacağımız ceza değerlerini -1 alalım ve ödül değerlerini de +2 olarak kabul edelim. Bu durumda ceza değeri olan aşağıdaki durumlarda -1 alınacaktır:

- w(boşluk) = -1
- w(a,-) = -1
- w(-,b) = -1
- w(uymaz) -1

Bütün iyi durumları da +2 olarak alalım ki tek durum uyma durumudur:

- w(uyar) = 2

Bu iki dizgi (string) için aşağıdaki şekilde bir tablo oluşturmak mümkündür:

	-	G	C	A	C	G	C	T	G
-									
G									
A									
C									
G									
C									
G									
C									
G									

Şekil 12: NeedLeman Wunsch Algoritması Çalışma Prensibi

Öncelikle, yukarıda görüldüğü üzere, boş bir tablonun satır ve kolon başlıklarına birer harf gelecek şekilde iki dizgi de (string) yerleştiriliyor.

	-	G	C	A	C	G	C	T	G
-	-1	-1	-1	-1	-1	-1	-1	-1	-1
G	-1								
A	-1								
C	-1								
G	-1								
C	-1								
G	-1								
C	-1								
G	-1								

Şekil 13: NeedLeman Wunsch Algoritması Çalışma Prensibi 2

Ardından, w(a,-) ve w(-,b) şartlarının -1 olmasından dolayı bu durumu tablomuza yerleştiriyoruz. Görüldüğü üzere ceza değeri bütün durumlar için uygulanmıştır.

	-	G	C	A	C	G	C	T	G
-	-1	-1	-1	-1	-1	-1	-1	-1	-1
G	-1	1	0	-1	-2	1	0	-1	1
A	-1								
C	-1								
G	-1								
C	-1								
G	-1								
C	-1								
G	-1								

Şekil 14: NeedLeman Wunsch Algoritması Çalışma Prensibi 3

İlk satırın çıkarılışını adım adım görelim. Öncelikle satır başlığımız G olduğu için sütunlarda bu harf ile eşleşmeler kontrol edilecektir. İlk kontrolümüz G-G kontrolü ve bu yüzden +2 değeri gelecek. Ancak hücrenin komşularından en büyük değer -1 olduğu için -1 + 2 = 1 sonucunu bularak hücreye yazıyoruz. Devamında ikinci kolonda G-C uyuşmazılığı söz konusu. Bu durumda -1 ceza uygulanacak ve en büyük komşumuz biraz önce bulduğumuz 1 olduğu için 1 – 1 = 0 değerini bu hücreye yazıyoruz. Ardından G-A uyuşmazlığından dolayı en büyük komşumuz olan 0 -1 = -1 sonucunu hücreye yazıyoruz. Hemen ardından G-C uyuşmazlığından dolayı bütün komşuları -1 olan hücremize -1 -1 = -2 sonucu yazılıyor. Bir sonraki kolonda ise G-G uyuşması var ve en büyük komşumuz olan -1 + 2 = 1 değeri hücreye yazılıyor. Çalışma bu şekilde devam ediyor yani o hücreye geldiğimizde satır ve sütun karşılaştırması yapılarak uyuşma halinde +2 diğer durumlarda ise -1 yazılıyor değeri en yüksek komşu değerine ekleniyor.

	-	G	C	A	C	G	C	T	G
-	-1	-1	-1	-1	-1	-1	-1	-1	-1
G	-1	1	0	-1	-2	1	0	-1	1
A	-1	0	-1	1	0	0	-1	-2	0
C	-1	-1	1	0	2	1	3	2	1
G	-1	1	0	-1	1	3	2	1	3
C	-1	0	2	1	3	2	4	3	2
G	-1	2	1	0	2	4	3	2	4
C	-1	1	3	2	4	3	5	4	3
G	-1	3	2	1	3	5	4	3	5

Şekil 15: NeedLeman Wunsch Algoritması Çalışma Prensibi 4

Yukarıda, tablonun tamamının doldurulmuş hali verilmiştir. Bu tabloda netice olarak iki dizginin birbirine yaslanması durumunda skoru gösterilmektedir. Ayrıca tablodaki herhangi bir hücre seçildiğinde, iki dizginin bu noktaya kadar olan halleri de bulunabilir. Örneğin GACG ile GCACG dizgilerini karşılaştırmak isteyelim, bu iki dizginin karşılaştırılması yukarıdaki tabloda 3 olarak değerlendirilmiştir ve en yüksek değerdeki komşuları, yukarı ve sola doğru izlenirse aşağıdaki gibi bir yol ortaya çıkar:

	-	G	C	A	C	G	C	T	G
-	-1	-1	-1	-1	-1	-1	-1	-1	-1
G	-1	1	0	-1	-2	1	0	-1	1
A	-1	0	-1	1	0	0	-1	-2	0
C	-1	-1	1	0	2	1	3	2	1
G	-1	1	0	-1	1	3	2	1	3
C	-1	0	2	1	3	2	4	3	2
G	-1	2	1	0	2	4	3	2	4
C	-1	1	3	2	4	3	5	4	3
G	-1	3	2	1	3	5	4	3	5

Şekil 16: NeedLeman Wunsch Algoritması Çalışma Prensibi 5

Yukarıdaki yol, sol üst köşeye kadar giden en yüksek değerlere sahip yoldur. Bu yolu okuyacak olursak:
GCACG
G-ACG
şeklinde arada bir boşluk bırakarak iki dizginin birleştirildiği görülebilir.

11. FuzzyWuzzy Algoritması

Eşleme dizelerini nasıl bulanıklaştırabileceğimizi göstermek faydalı olacaktır. Normalde, Python'daki dizeleri karşılaştırdığınızda şunları yapabilirsiniz:

```
Str1 = "Apple Inc."
Str2 = "Apple Inc."
Result = Str1 == Str2
print(Result)
```

```
True
```

Bu durumda, dizeler tam eşleşme (% 100 benzerlik) olduğu için Sonuç değişkeni True yazacaktır, ancak Str2 durumu değişirse ne olacağını görün:

```
Str1 = "Apple Inc."
Str2 = "apple Inc."
Result = Str1 == Str2
print(Result)
False
```

Bu kez ipler insan gözüyle aynı görünmesine rağmen bir Yanlış yaptık. Ancak, her iki dizeyi de küçük harfe zorlayabileceğimiz için bu sorunu çözmek çok basittir:

```
Str1 = "Apple Inc."
Str2 = "apple Inc."
Result = Str1.lower() == Str2.lower()
print(Result)
True
```

Bununla birlikte, bu mutluluk bir karakter biter bitmez sona erer. Örneğin:

```
Str1 = "Apple Inc."
Str2 = "apple Inc"
Result = Str1.lower() == Str2.lower()
print(Result)
False
```

Yukarıdaki gibi durumlar, bazen, insan veri girişine dayanarak oluşturulmuş veritabanlarında görünebilir ve bu durumlarda dizeleri karşılaştırmak için daha güçlü araçlara ihtiyacımız vardır.

Şimdiye kadar "Apple Inc." ile kullandığım örnek ve "apple Inc" nispeten basitti. Sonuçta, her iki dizeyi de küçük harfe çevirirseniz yalnızca bir tam duraklama / fark süresi vardır. Ancak, bir şey düzensiz olduğunda ne olur? Bir şeyin hatırı sayılır bir yazım varyasyonu olduğunda ne olur, ama yine de aynı şeyi

ifade eder? FuzzyWuzzy paketi, bulanık eşleme komut dosyalarımızın bu tür vakaları işlemesine izin veren işlevlere sahip olduğu için devreye giriyor.

Basit başlayalım. FuzzyWuzzy, tıpkı Levenshtein paketi gibi, iki sekans arasındaki standart Levenshtein mesafe benzerlik oranını hesaplayan bir oran fonksiyonuna sahiptir. Aşağıda bir örnek görebilirsiniz:

```
from fuzzywuzzy import fuzz
Str1 = "Apple Inc."
Str2 = "apple Inc"
Ratio = fuzz.ratio(Str1.lower(),Str2.lower())
print(Ratio)
95
```

Bu benzerlik oranı, yukarıdaki diğer örnekler göz önüne alındığında beklediğimizle aynıdır. Bununla birlikte, fuzzywuzzy, alt dize eşleştirme gibi daha karmaşık durumlarla başa çıkmamızı sağlayan daha güçlü işlevlere sahiptir. İşte bir örnek:

```
Str1 = "Los Angeles Lakers"
Str2 = "Lakers"
Ratio = fuzz.ratio(Str1.lower(),Str2.lower())
Partial_Ratio = fuzz.partial_ratio(Str1.lower(),Str2.lower())
print(Ratio)
print(Partial_Ratio)
50
100
```

fuzz.partial_ratio (), her iki dizginin de Lakers'a atıfta bulunduğunu tespit edebilir. Böylece% 100 benzerlik sağlar. Bunun çalışma şekli, "optimal kısmi" mantık kullanmaktır. Başka bir deyişle, kısa dizginin uzunluğu k isekve uzun dize m uzunluğuna sahiptirm, ardından algoritma en uygun uzunluğun skorunu arar- kk alt dize döndürür.

Bununla birlikte, bu yaklaşım kusursuz değildir. Dizeler aynı şekilde karşılaştırıldığında, ancak farklı bir sırada olduklarında ne olur? Neyse ki bizim için fuzzywuzzy'nin bir çözümü var. Aşağıdaki örneği görebilirsiniz:

```
Str1 = "united states v. nixon"
Str2 = "Nixon v. United States"
Ratio = fuzz.ratio(Str1.lower(),Str2.lower())
Partial_Ratio = fuzz.partial_ratio(Str1.lower(),Str2.lower())
Token_Sort_Ratio = fuzz.token_sort_ratio(Str1,Str2)
print(Ratio)
print(Partial_Ratio)
print(Token_Sort_Ratio)
59
74
100
```

Fuzz.token işlevlerinin oran ve kısmi_ratio'ya göre önemli bir avantajı vardır. Dizeleri tokenleştiriyor ve küçük harfe çevirip noktalama işaretlerinden kurtararak ön işlem yapıyorlar. Fuzz.token_sort_ratio () durumunda, dize belirteçleri alfabetik olarak sıralanır ve sonra birleştirilir. Bundan sonra, benzerlik yüzdesini elde etmek için basit bir fuzz.ratio () uygulanır. Bu, bu örnekteki mahkeme davaları gibi davaların aynı olarak işaretlenmesine izin verir.

Yine de, bu iki dize çok farklı uzunluklarda olursa ne olur? İşte fuzz.token_set_ratio () devreye girer. İşte bir örnek:

```
Str1 = "The supreme court case of Nixon vs The United States"
Str2 = "Nixon v. United States"
Ratio = fuzz.ratio(Str1.lower(),Str2.lower())
Partial_Ratio = fuzz.partial_ratio(Str1.lower(),Str2.lower())
Token_Sort_Ratio = fuzz.token_sort_ratio(Str1,Str2)
Token_Set_Ratio = fuzz.token_set_ratio(Str1,Str2)
print(Ratio)
print(Partial_Ratio)
print(Token_Sort_Ratio)
print(Token_Set_Ratio)
57
77
```

```
58
95
```

% 95 benzerlik bu sihir mi? Hayır, sadece kaputun altında ip ön işleme. Özellikle, fuzz.token_set_ratio (), fuzz.token_sort_ratio () 'dan daha esnek bir yaklaşım benimser. Yalnızca dizeleri belirtmek, sıralamaları birleştirmek ve sonra birleştirmek yerine, token_set_ratio, ortak belirteçleri (kesişim) alan ve ardından aşağıdaki yeni dizeler arasında fuzz.ratio () çift karşılaştırmaları yapan bir küme işlemi gerçekleştirir:

- s1 = Sorted_tokens_in_intersection
- s2 = Sorted_tokens_in_intersection + sort_rest_of_str1_tokens
- s3 = Sorted_tokens_in_intersection + sort_rest_of_str2_tokens

Bu karşılaştırmaların ardındaki mantık, Sorted_tokens_in_intersection her zaman aynı olduğundan, bu kelimeler orijinal dizelerden daha büyük bir yığın oluşturduğundan veya kalan belirteçler birbirine daha yakın olduğundan puan yükselme eğilimindedir.

Son olarak, fuzzywuzzy paketinde, bir dize vektöründen en yüksek benzerliğe sahip dizeyi hesaplamanızı sağlayan süreç adı verilen bir modül bulunur. Bunun nasıl çalıştığını aşağıda görebilirsiniz:

```
from fuzzywuzzy import process
str2Match = "apple inc"
strOptions = ["Apple Inc.","apple park","apple incorporated","iphone"]
Ratios = process.extract(str2Match,strOptions)
print(Ratios)
# You can also select the string with the highest matching percentage
highest = process.extractOne(str2Match,strOptions)
print(highest)
[('Apple Inc.', 100), ('apple incorporated', 90), ('apple park', 67), ('iphone', 30)]
('Apple Inc.', 100)
```

Bu öğreticide, yaklaşık olarak dizeleri nasıl eşleştireceğinizi ve nasıl benzer olduklarını belirlemeyi öğrendiniz. Burada sunulan örnekler basit olabilir, ancak bir bilgisayarın uyumsuz dizeler olduğunu düşündüğü şeylerin çeşitli durumlarının nasıl ele alınacağını göstermek için yeterlidir. Bu eğiticiye,

manuel olarak girilen şirket adlarını işverenimin Salesforce CRM'sinde ("Apple Inc." için "apple inc" olarak mevcut hesap adlarıyla eşleştirmek için bulanık dize eşleşmesi kullanmak zorunda kaldığım bir duruma göre yaklaştım. dönüşümler).

5. VERİLER NASIL ELDE EDİLİYOR?

Programımızda kullanmış olduğumuz soru ve cevapları bizim program verilerimizdir. Kırklareli Üniversitesinin Chatbot projesini geliştirdiğimiz için, veritabanımız Kırklareli ve Kırklareli Üniversitesi ile ilgili her soruyu ve cevabını barındırmak zorundadır.

Fakat; elimizde o denli geniş bir veri olmadığı için, ve o kadar büyük veriyi elde edebileceğimiz herhangi bir kaynak olmadığı için temel olarak Kırklareli Üniversitesi SSS kullanmayı tercih ettik. Küçük bir veritabanı ile daha az maliyet, sorunsuz çalışma olasılığı yüksek bir program yapmamız kolaydır.

Bu ve bunun gibi etmenler göz önünde bulundurularak veri olarak KLU SSS kullanımı seçildi. Bu soruların programa entegre işlemi ise manuel olarak gerçekleşmiştir. Sorular öncelikle bir kaynak dosyaya aktarılmıştır. Daha sonra kodun içerisine gömülmüştür.

3.BÖLÜM: CHATBOT HAYAT BULUYOR:

TESTLER VE GÖRÜŞLER

1. Yazılım Testlerinin Yapılma Nedenleri

Geliştirilen yazılımlarda, kaynağı ve yazılımı değişmekle birlikte çeşitli hataların ortaya çıkması kaçınılmazdır. Çalışılan uygulama alanının kritikliğine göre bu hataların doğuracağı sonuçların biçimi değişebilmektedir. Bir finans uygulamasında yapılan küçük bir hata çok büyük miktarlarda para kaybına yol açabilecekken, bir askeri uygulamada yapılması muhtemel küçük bir hata mal kaybının yanı sıra can kaybına da neden olma riskini taşımaktadır. Bununla birlikte uygulamanın türüne bağlı olarak yazılım testleri,

- Müşteriye sunulmadan önce ürün kalitesinden emin olmak,
- Yeniden çalışma (düzeltme) ve geliştirme masraflarını azaltmak,
- Geliştirme işleminin erken aşamalarında yanlışları saptayarak ileri aşamalara yayılmasını önlemek, böylece zaman ve maliyetten tasarruf sağlamak,
- Müşteri memnuniyetini arttırmak ve izleyen siparişler için zemin hazırlamak,

amaçları doğrultusunda da yapılmaktadır. Ayrıca ürettikleri yazılımları yeterince test etmeden müşterilerinin hizmetine sunan firmalar, olası yazılım hataları sonucunda itibar kaybedecekleri gibi müşterilerine tazminat ödemek durumunda da kalabilirler.

Genel olarak yazılım projeleri analiz -> tasarım -> kodlama -> test -> ürün süreçleri izlenerek geliştirilir. Bütün süreçler birbirini bu şekilde izlese de test süreci hiçbir zaman kodlama sürecinin bitmesini beklemez. İdeal bir yazılım test süreci, analiz -> tasarım -> test hazırlık süreci-> kodlama -> dinamik test süreci -> testin sonlandırılması -> ürün şeklinde olmak durumundadır. Test süreçlerince yapılması gereken işlemler şu şekilde sıralanır:

a. Test Hazırlık Süreci

Bu süreç, yazılım test süreçleri içindeki ilk aşama olmakla beraber, testin efektif sonuçlar vermesi ve verimli olması açısından büyük öneme sahiptir. Dolayısıyla, bir yazılımı iyi test edebilmek için, test işlemlerinden önce, sağlıklı bir test hazırlık süreci kaçınılmazdır. Test hazırlık sürecinde yapılması gereken birtakım standart işlemler şu şekilde sıralanır:

- Bu süreç, yazılım test süreçleri içindeki ilk aşama olmakla beraber, testin efektif sonuçlar vermesi ve verimli olması açısından büyük öneme sahiptir. Dolayısıyla, bir yazılımı iyi test edebilmek için, test işlemlerinden önce, sağlıklı bir test hazırlık süreci kaçınılmazdır. Test hazırlık sürecinde yapılması gereken birtakım standart işlemler şu şekilde sıralanır:
- Öncelikle test edilecek yazılıma ait analiz ve teknik tasarım aşamaları ile ilgili dökümanlar, test ekibi tarafından incelenir.
- Yazılım içinde test edilecek ve edilmeyecek modüller belirlenir.
- Risk analizi yapılır ve yapılan değerlendirmeye göre dinamik test aşamasında uygulanacak olan test teknikleri ve metodları belirlenir.
- Dinamik testin uygulanacağı ortamlar ve bu ortamların ihtiyaçları belirlenip, uygun şartlar sağlanır.
- Test ekibi içinde görev paylaşımı ve zaman planlaması yapılır.
- Testin sonlandırma kriterleri belirlenir.
- Bir programa belirli girdiler (input) verildiğinde hangi çıkışların (output) ne şekilde alınması gerektiğini bildiren test case senoryaları belirlenir.
- Dinamik testin hangi adımlarla ve ne şekilde uygulanacağının belirtildiği test planı hazırlanır.

Yukarıda sıralanan test hazırlık sürecine ait aşamalar gerçeklendikten sonra dinamik yazılım testi aşamalarına geçilir.

b. Dinamik Test Süreci

Bu süreç kodlama çalışmalarının bitmesine yakın bir dönemde başlar. Bulunan tüm hatalar çözülmeden ve testin sonlandırma kriterleri sağlanmadan sona ermez. Test edilecek yazılımın türüne göre, dinamik olarak uygulanacak test teknikleri ve bu tekniklerin uygulanma metotları farklılık gösterebilir. Genel olarak dinamik test süreci içinde ve sonrasında uygulanabilecek olan testler ve test teknikleri şu şekilde sıralanır:

- Birim Testi (Unit Testing) : Dinamik test sürecinin ilk aşaması olmakla beraber, hataların erken bulunup düzeltilebilmesi açısından da bu sürecin en önemli aşamasını oluşturur. Mikro ölçekte yapılan bu testte, özel fonksiyonlar veya kod modülleri (fonksiyonlar, veri yapıları, nesneler vb.) test edilir. Bu test, test uzmanlarınca değil programcılar tarafından yapılır ve program kodunun ayrıntıları ile içsel tasarım biçiminin bilinmesi gerekir. Uygulama kodu çok iyi tasarlanmış bir mimaride değilse oldukça zor bir testtir.

- Tümleyim Testi (Integration Testing) : Bir uygulamanın farklı bileşenlerinin beraberce uyum içinde çalışıp çalışmadığını sınamak için yapılan bir testtir. Bileşenler, modüller, bağımsız uygulamalar, istemci/sunucu uygulamaları biçiminde olabilirler. Bu tür testlere, özellikle istemci/sunucu uygulamaları ve dağıtık sistemlerin testinde başvurulmaktadır. Bunun yanısıra uygulamaya yeni işlevsel elemanlar ya da program modülleri eklendikçe sürekli test edilmesi işlemine de “Artımsal Tümleyim Testi” adı verilir. Test uzmanları ve/veya programcılar tarafından gerçekleştirilen testlerdir.

- Regresyon Testi (Regression Testing) : Uygulamada ve uygulama ortamlarında gerekli değişiklikler ve sabitlemeler yapıldıktan sonra yeniden yapılan testlere çekilme (regresyon) testi denilir. Böylece, önceki testlerde belirlenen sorunların giderildiğinden ve yeni hatalar oluşmadığından emin olunur. Uygulamanın kaç kez yeniden test edilmesi gerektiğini belirlemek güçtür ve bu nedenle, özellikle uygulama geliştirme döneminin sonlarına doğru yapılır.

- Zorlanım – Performans Testi (Performance Testing) : Bu test, çoğu kez "yük testi" ile aynı anlamda kullanılmaktadır. Aynı zamanda, beklenmedik (normal olmayan) ağır yükler, belirli

eylemler ve taleplerin çok fazla artışı, çok yoğun sayısal işlemler, çok karmaşık sorgulamalar vb. ağır koşullar altında olan bir sistemin işlevsellik testi (iş yapabilme testi) olarak da tanımlanabilmektedir. Bir web sitesi için sistem tepkisinin hangi noktada azaldığı veya yanıt veremez olduğunu belirlemek için yapılan testler, performans testine örnek teşkil edebilir.

- Kullanıcı Kabul Testi (User Acceptance Testing) : Son kullanıcı veya müşteri siparişine (veya isteklerine) dayanan son test işlemidir. Kullanıcıların, uygulamayı "kabul" etmeden önce, söz konusu uygulamanın gereksinimlerini ne ölçüde karşılayıp karşılamadığını belirleyip, geri dönüş yapabileceği testlerdir.

- Beyaz Kutu Test Tekniği (White Box Testing Technic) : Beyaz kutu test tekniğinin en genel tabiri kod testidir. Projenin hem kaynak kodu, hem de derlenmiş kodu test edilir. Bu tür testler, uygulama kodunun iç mantığı üzerindeki bilgiye bağlıdır. Yazılım kodundaki deyimler, akış denetimleri, koşullar vb. elemanlar sınanır.

- Kara Kutu Test Tekniği (Black Box Testing Technic) : Test ekipleri tarafından en çok kullanılan teknik olan kara kutu test tekniği adından da anlaşılacağı gibi uygulamanın sadece derlenmiş kodu üzerinden test edilmesi olarak bilinir. Bu test tekniğinde, yazılımın program yapısı, tasarımı veya kodlama tekniği hakkında herhangi bir bilgi olması gerekli değildir. Yazılımın gereksinimine duyulan şeylere yanıt verip veremediği ve işlevselliği sınanmaktadır.

2. Uygulama Testimizi Nasıl Yaptık?

Geliştirdiğimiz her yazılıma adım adım test işlemi uygulamalıyız. Hata ve eksikliklerin giderilmesi için testlerin yapılması, sonuçlandırılıp gerekli işlemlerin yapılması gerekmektedir.

KLUBOT uygulamasına test işleminin yapılmasının temel sebebi "algoritmaların çalışma verimliliğini" değerlendirebilmektir. Projemizin test işlemini öncelikle ekip üyeleri gerçekleştirdi. Fark edilen hatalar giderildi, giderilemeyen hatalar için gerekli kişilerden yardım alındı.

Ardından program üzerinde "Tester" olarak işlem gerçekleştirecek kişiler belirlendi. 10 kişilik küçük bir tester grubu oluşturuldu. Ve arayüze entegre edilmiş KLUBOT uygulamamız testerlara teslim edildi.

Testerın uygulamaya sorduğu her soru veritabanımızca kaydediliyor. Testerın sorduğu soruyu veritabanındaki herhangi bir soru ile benzerliğini kuramazsa, yeni bir soru olarak veritabanına kaydını gerçekleştirmektedir.

Böylelikle her testerın uygulamadan yüzde kaç oranında doğru yanıt alabildiğini, verimliliğinin ne kadar olduğunu görüntüleyip değerlendirebildik.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Sorulan Soru	SequenceMatcher 1		SequenceMatcher 2		CosineSimilarityAlg 1		CosineSimilarityAlg 2		JaccardAlg 1		JaccardAlg 2	
2	yks sınavı ile yerleştim. ne yapmam gerekiyor kayıt için?	1	79%	2	53%	1	100%	2	60%	1	50%	2	28%
3	yks sınavı ile kazandım. kayıt için gelebilecek durumda değilim ne yapmalıyım? kaydı posta ile yollayarak yapabilir miyim?	2	76%	1	48%	2	67%	1	54%	2	43%	1	32%
4	kayıt için gereken belgelerimde eksikler var. belgeler tamamlanmadan kayıt yaptırabilir miyim?	3	69%	2	47%	3	52%	2	43%	3	25%	2	20%
5	yüksek öğretim kurumları sınavı sonucu klü'ye yerleştim. yerleşmeme rağmen kaydı gerçekleştirememe durumu olur mu?	4	61%	1	43%	4	23%	1	17%	4	14%	12	8%
6	danışman hoca kimdir? ne yapar?	5	87%	127	44%	5	80%	45	22%	5	67%	45	13%
7	kayıt dondurabilir miyim?	6	80%	94	45%	3	41%	2	39%	6	17%	2	13%
8	öğrenim ücreti hangi dönemde ve nasıl ödenir?	7	80%	30	51%	7	71%	27	31%	7	56%	27	21%
9	öğrenci harçlarını kim belirliyor?	8	56%	20	45%	8	45%	16	25%	8	29%	16	14%
10	belirtilen süre zarfında öğrenciler okula kaydını yaptıramazsa ne olur?	9	58%	122	46%	9	56%	20	25%	9	38%	20	14%
11	kesin kayıt için önceden arayıp bir işlem yapmam gerekiyor mu?	10	67%	1	59%	1	42%	12	37%	12	22%	10	20%
12	dönem ortalamam 1.80'in altında olduğunda kaç kredi ders alabiliyorum?	11	85%	107	56%	11	61%	86	17%	11	27%	28	7%
13	başka bir yükseköğretimde kaydım var. kayıt olmak için diplomamın kopyası yeterli olur mu?	12	66%	17	43%	12	64%	94	39%	12	47%	3	17%
14	üniversiteye yerleşmeye hak kazandım fakat liseyi henüz bitiremedim. nasıl kayıt yaptırabilirim?	13	67%	4	46%	13	39%	95	23%	13	18%	95	13%
15	üniversiteye yeni kayıtların ders alma işlemleri nasıl yapılır?	14	83%	21	63%	14	67%	21	47%	14	50%	21	30%
16	üniversitenin programlarına ösym yerleşiminden başka bir şekilde kayıt olunabilir mi?	15	61%	28	52%	6	32%	12	27%	6	17%	12	16%
17	öğrenci belgemi nereden almalıyım?	16	82%	111	55%	16	50%	131	25%	16	33%	131	14%
18	daha önce bir yükseköğretimde kayıtlı olduğumdan dolayı başarılı durumdaki derslerim muaf olabilir mi? nasıl?	17	75%	4	39%	17	43%	4	16%	17	23%	4	9%
19	gano ve yano ne anlama gelir?	159	48%	69	48%	18	65%	43	33%	18	45%	43	20%
20	öğrenci harcımı yatırdıktan sonra kaydımı dondurdum. kayıt dondurduğum dönem öğrenci harcımı yatırmalı mıyım?	19	59%	24	40%	19	28%	16	24%	19	20%	128	13%
21	talebe kaydını sildirmek istediğinde ne yapmalı?	20	80%	143	52%	20	46%	9	27%	20	30%	9	15%
22	askerlik erteleme işlemleri nasıl yapılır?	21	91%	14	53%	21	80%	14	42%	21	67%	14	25%
23	ilk sınıf öğrencileri üstten ders alabilir mi?	22	84%	133	65%	22	63%	133	32%	22	50%	133	20%
24	öğrenim süremden sayılır mı kayıt yaptırmadığım dönem?	128	49%	23	48%	23	100%	19	43%	23	56%	19	11%
25	lise diplomam olmadan kayıt yaptırabilir miyim?	24	57%	128	54%	3	38%	2	37%	24	18%	2	18%
26	üniversitenin yerleşkelerine nasıl ulaşabilirim?	25	69%	125	57%	78	41%	25	38%	78	25%	25	22%
27	kırklareli'nin barınma olanakları nelerdir?	26	77%	46	45%	26	51%	141	27%	26	22%	98	13%
28	ne zaman ve nasıl staj müracaatı yapılır? işletmelerde hangi staj yapabilirim?	27	73%	43	43%	27	82%	43	44%	27	67%	43	33%
29	bölümümle alakalı programları nereden öğrenebilirim?	28	73%	45	47%	28	37%	16	22%	28	22%	16	13%
30	okula girerken veya imtihanlarda öğrenci belgem yanımda yoksa ne yapabilirim?	29	60%	16	47%	29	57%	144	28%	29	40%	144	15%
31	öğrenci toplulukları nelerdir? nasıl katılabilirim?	30	76%	16	59%	30	55%	96	26%	30	38%	96	14%
32													
33			Doğru Cevap		100%		0%						
34													

Şekil 17: Testerın uygulamaya sorduğu ilk 30 soru ve algoritmaların doğru yanıt verme istatistikleri

Yukarıdaki excel veri tablosu bir testerın KLUBOT uygulamasına sorduğu ilk 30 soru ve doğru yanıt alma oranlarıdır. Görselden de göründüğü üzere "Sequence Matcher 1" , "Cosine Similarity Alg 1" , "Jaccard Alg 1" en verimli çalışan algoritmalardır. Ve bizim programımızda yer verdiğimiz 3 algoritma budur.

Veriler veritabanına "JSON" formatında kaydediliyor. Ardından "Sık sorulan Sorular Veritabanı" kontrol edilerek sorunun en benzer sorusu veritabanı içerisinde aranıyor. Algoritmaların çıkarttığı sonuçlar doğrultusunda "Çoğunluk Oylaması İşlemi" kullandığımız 3 algoritmanın üzerinde gerçekleşir.

Çoğunluk oylaması işleminin sonucuna bağlı olarak, soruya doğru yanıt en hızlı şekilde döndürülmüş olur.

Chatbotlar günlük hayatta fazlaca kullanıldığı için ve kullanıcılar vaktinin küçük bir kısmını bot ile iletişim halinde geçirmek istiyorlar. Kullanıcılar, botlardan hızlıca doğru cevap ve yönlendirmeler alarak konuşmayı sonlandırma yanlısıdırlar. KLUBOT'un da çalışma prensibi tam olarak bu yöndedir. "**HIZ"** ve "**verimlilik**".

3. KLUBOT ARAYÜZ TASARIMI VE LOGOSU

RESİM 3: KLUBOT ARAYÜZ ÖRNEĞİ

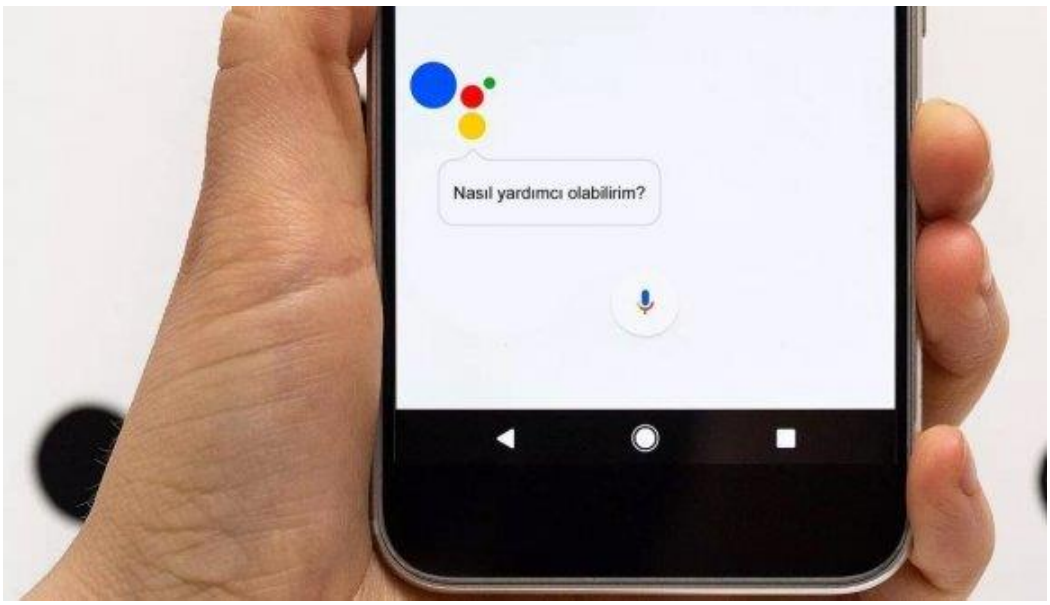
RESİM 4: KLUBOT LOGO

- Uygulamaya Google Asistan'ın Entegresi

Projemizde Google Asistana yer verme amacımız; KLUBOT üzerinde gerçekleşen görüşmelerin seslendirilmesidir. Kullanıcıların sesli bir şekilde soru sorup, sesli yanıt olabilmesine olanak sağlamak için projemize entegre ettik.

Projemize entegre işlemi için yapmış olduğumuz adımlar:

1. gtts ve playsound kütüphanesi proje edildi.
2. Kullanıcıya döndürülen veri aynı zamanda bu kütüphanelere iletildi.
3. Bu kütüphaneler yardımıyla alınan string değerini sese çevirerek mp3 formatında kayıt edildi.
4. Mp3 oynatıldı ve sistemde yer kaplamaması için işlem sonunda silindi.



Resim 5: Örnek Google Asistan kullanıcı arayüzü

4. Kullanıcıların KLUBOT Hakkındaki Görüşleri

“Doğru yanıt verebilme becerisi gayet iyi. Fakat geliştirilmesi gereken noktaları var.”

“Arayüz tasarımı olarak gayet kullanışlı. Tavsiye ediyorum.”

“Üniversite öğrencisi olarak birçok soruma yanıt bulabildim. Fakat veritabanlarını genişletmeliler.”

“Herkesin emeğine sağlık. Güzel bir iş..”

5. Projenin Verimi ve Doğru Yanıt Döndürme Süresi

Projemizin son halinde belirli algoritmik iyileştirmeler yapıldı. Belirli testler yapılarak, projenin halihazırda varmış olduğu noktadaki verimi ölçüldü. Projemizin verimliliğinden şu şekilde bahsetmek isterim, uygulamanın kullanıcıya cevap döndürme süresi 0.55 saniyedir. Ve bu süre içerisinde kullanıcıya döndürdüğü cevapların %94.33’ü doğru cevaptır.

SONUÇ

KLUBOT projemize başlarken üniversite bünyesinde çalışan memurların yükünü azaltmak, yeni öğrenci/ yeni kayıt/aday öğrenciler için hızlı bir geri dönüt sistemi elde etmek, kişilerin ve kurumların zaman verimliliğini arttırmak gibi amaçlarımız vardı. Projemizin sonunda görüyoruz ki, amaçlarımıza ulaşabildik. Üniversite bünyesi için elverişli olacak bir sistemi bünyesine katmış olduk. Üniversite web sayfasında projemize yer verilerek daha çok kişiye ulaşabileceğiz.

Kullandığımız karşılaştırma algoritmalarında detaylıca bahsettik. Bu algoritmaların çeşitli şekilde projeye dahil edilip test edilmesiyle hangi algoritmanın bizim için daha verimli sonuçlar ürettiğini görüntüledik.

Son olarak "Google Asistanın" projeye entegre edilip sorunsuz çalışması ile, projenin verimliliği, doğru yanıt döndürme olasılığı artmış oldu. Ayrıca uygulamanın kullanılabilirliğini de arttırmış olduk.

Resim 5: Örnek Google Asistan Logo

ELEKTRONİK KAYNAKLAR

https://blog.euromsg.com/yeni-baslayanlar-icin-chatbot-nedir-nasil-calisir/ (6 Mayıs 2020)

https://lechatbot.com/türkiyeden-ve-dünyadan-chatbot-örnekleri-5f99ff10a4c (6 Mayıs 2020)

https://tr.wikipedia.org/wiki/Algoritma (12 Mayıs 2020)

https://www.mediaclick.com.tr/blog/turkiye-ve-dunyadan-chatbot-ornekleri (14 Mayıs 2020)

https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/yazılım-testi-ve-test-süreçleri (14 Mayıs 2020)

http://bilgisayarkavramlari.sadievrenseker.com/2010/12/30/levenshtein-mesafe-algoritmasi-levenshtein-distance/

https://stackabuse.com/levenshtein-distance-and-text-similarity-in-python/

http://bilgisayarkavramlari.sadievrenseker.com/2013/07/01/jaccard-indeksi-jaccard-index/

https://www.statisticshowto.com/jaccard-index/

https://deepai.org/machine-learning-glossary-and-terms/jaccard-index

https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity/

http://bilgisayarkavramlari.sadievrenseker.com/2012/11/08/kosinus-benzerligi-cosine-similarity-2/

https://medium.com/@emrehavan/recommender-system-%C3%B6neri-sistemi-uygulamas%C4%B1-geli%C5%9Ftirme-b%C3%B6l%C3%BCm-1-4-cosine-similarity-1323e3b5b6ae

https://www.sciencedirect.com/topics/computer-science/cosine-similarity

https://www.geeksforgeeks.org/sequencematcher-in-python-for-longest-common-substring/

https://kite.com/python/docs/difflib.SequenceMatcher.find_longest_match

https://pypi.org/project/jaro-winkler/

https://www.geeksforgeeks.org/jaro-and-jaro-winkler-similarity/

http://www.bilgisayarkavramlari.com/2007/12/04/en-uzun-ortak-kume-longest-common-subsequence-lcs/

http://bilgisayarkavramlari.com/2007/12/03/dinamik-programlama-dynamic-programming/

http://bilgisayarkavramlari.sadievrenseker.com/2011/04/23/n-gram/

http://www.bilgisayarkavramlari.com/2008/08/02/dizgi-string/

http://www.bilgisayarkavramlari.com/2012/06/06/smith-waterman-dizgi-yaslama-string-alignment-algoritmasi/
http://www.bilgisayarkavramlari.com/2012/06/10/dizgi-hizalama-string-alignment/
http://www.bilgisayarkavramlari.com/2012/06/05/needleman-wunsch-algoritmasi/
https://www.datacamp.com/community/tutorials/fuzzy-string-python