Aivan Vo

Sisältöpohjainen suosittelija matka-suunnitelmasovellukselle

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

1.5.2024

Tiivistelmä

Tekijä:	Aivan Vo
Otsikko:	Sisältöpohjainen suosittelija matkasuunnitelmasovellukselle
Sivumäärä:	33 sivua
Aika:	1.5.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Vesa Ollikainen

Tämän opinnäytetyön tavoitteena on toteuttaa sisältöpohjainen suosittelualgoritmi, joka hyödyntää kosinin samankaltaisuutta matkasuunnitelmasovellusta varten. Työn tarkoituksena on parantaa käyttäjäkokemusta tarjoamalla personoituja suosituksia eri nähtävyyksistä, jotka perustuvat aikaisempiin vuorovaikutuksiin sovelluksen kanssa. Hyödyntämällä kosinin samankaltaisuutta algoritmi arvioi eri nähtävyyksien ja käyttäjän mieltymysten välisiä samankaltaisuuksia luodakseen kymmenen kärjen suosituslistan, jota käyttäjä voi dynaamisesti muokata.

Kehitysprosessi sisältää Google Places API:n integroinnin reaaliaikaisen tietojen hakemiseen mahdollisista nähtävyyksistä. Tämän jälkeen seuraa datan suodattaminen ja järjestely nähtävyyksien suositummuuden perusteella. Keskeinen analyysitekniikka perustuu malliin, joka soveltaa kosinin samankaltaisuutta kohteiden ominaisuuksien ja käyttäjämieltymysten suhteen laskemiseksi. Tätä mallia hienosäädetään jatkuvasti käyttäjän antaman palautteen kautta, jossa käyttäjien vuorovaikutukset analysoidaan jatkuvasti suositusmoottorin päivittämiseksi.

Tämä työ edistää koneoppimistekniikoiden käytännön soveltamista matkailualalla, erityisesti kehittämällä käyttäjäkeskeisemmän suositusjärjestelmän. Suositusjärjestelmä parantaa nähtävyyskohteiden ehdotusten tarkkuutta ja personointia yhdistämällä käyttäjän vuorovaikutuksen sovelluksen kanssa dynaamisiin suosituspäivityksiin.

Avainsanat: kosinin samankaltaisuus, sisältöpohjainen suodatus, matkasuunnitelma, suosittelujärjestelmä, personalisointi, google places API

Abstract

Author:	Aivan Vo
Title:	Content-based recommender for a travel itinerary application
Number of Pages:	33 pages
Date:	1 May 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Vesa Ollikainen, Senior Lecturer

The goal of this study is to develop a content-based recommendation algorithm using cosine similarity for a travel itinerary application. The target is to enhance the user experience by providing personalized travel recommendations based on interactions with the application. By using the properties of cosine similarity, the algorithm finds the similarity between attractions and user preferences to generate a top 10 list of attractions to visit, which the user can dynamically edit.

The development process includes integrating the Google Places API for real-time data retrieval on potential attractions. This is followed by filtering and organizing the data based on popularity. The analytical technique is based on a model that applies cosine similarity to calculate the relationship between the features of destinations and user preferences. This model is continuously fine-tuned through user feedback, where user interactions are constantly analyzed to update the recommendation engine.

This study enhances the practical application of machine learning techniques in the tourism industry, particularly by developing a more user centered recommendation system. By integrating user interaction with the application into dynamic recommendation updates, the algorithm improves the accuracy and personalization of suggested attractions.

Keywords: cosine similarity, content-based filtering, travel itinerary, recommendation system, personalization, google places API

Sisällys

1 Johdanto

Suositusjärjestelmistä on tullut olennainen osa erilaisia digitaalisia alustoja, jotka auttavat käyttäjiä löytämään sisältöä, joka on räätälöity heidän mieltymystensä mukaan.

Tämä opinnäytetyö keskittyy hyödyntämään sisältöpohjaista suodatustekniikkaa yksilöllisen suosittelualgoritmin kehittämiseksi matkailusovellukselle. Sisältöpohjainen suodatus hyödyntää kohteiden ominaisuuksia ja käyttäjän aiempia vuorovaikutuksia suositusten luomiseksi.

Tämän työn tavoitteena on suunnitella matkailusovellus, joka tarjoaa käyttäjälle henkilökohtaisia suosituksia vierailukohteista, jotka perustuvat heidän vuorovaikutukseen sovelluksen kanssa. Sovellukseen on tarkoitus kehittää sisältöpohjainen suosittelualgoritmi kosinin samankaltaisuutta hyödyntäen. Sovelluksen tavoitteena on luoda top 10 -lista vierailukohteista käyttäjän antamat matkakohteen perusteella. Työssä hyödynnetään Google Places API:sta saatavaa dataa erilaisista nähtävyyksistä, ravintoloista ja muista vierailukohteista.

Työssä käydään ensin sisältöpohjainen suosittelualgoritmi yleisesti läpi sekä perehdytään tarkemmin kosinin samankaltaisuuteen, jota hyödynnetään myöhemmin kehitettävässä suosittelualgoritmissa. Seuraavaksi käydään matkailusovelluksen konsepti yleisesti läpi.

Tämän jälkeen haetaan ja käsitellään Google Places API:sta saadut kohteet. API:sta saatava data muodostaa perustan myöhemmin kehitettävälle suosittelualgoritmille, kun käyttäjä on ollut vuorovaikutuksessa sovelluksen kanssa.

Seuraavaksi luodaan suosittelujärjestelmä sovellukselle ja toteutetaan prototyyppi.

Lopuksi arvioidaan luodun suosittelualgoritmin tehokkuutta henkilökohtaisten suositusten antamisen näkökulmasta käyttäjille.

2 Sisältöpohjainen suosittelualgoritmi

Algoritmi on ohjeiden tai sääntöjen sarja, jonka tarkoituksena on suorittaa annettu tehtävä tai ratkaista tietty ongelma [1]. Algoritmit toimivat täsmällisenä luettelona ohjeista, jotka suorittavat tiettyjä toimia askel askeleelta joko laitteisto- tai ohjelmistopohjaisissa rutiineissa. Niitä käytetään laajasti IT-alalla, kuten ohjelmoinnissa ja tietojenkäsittelytieteessä. Ne ratkaisevat toistuvia ongelmia ja toimivat ohjeina tietojenkäsittelyssä.

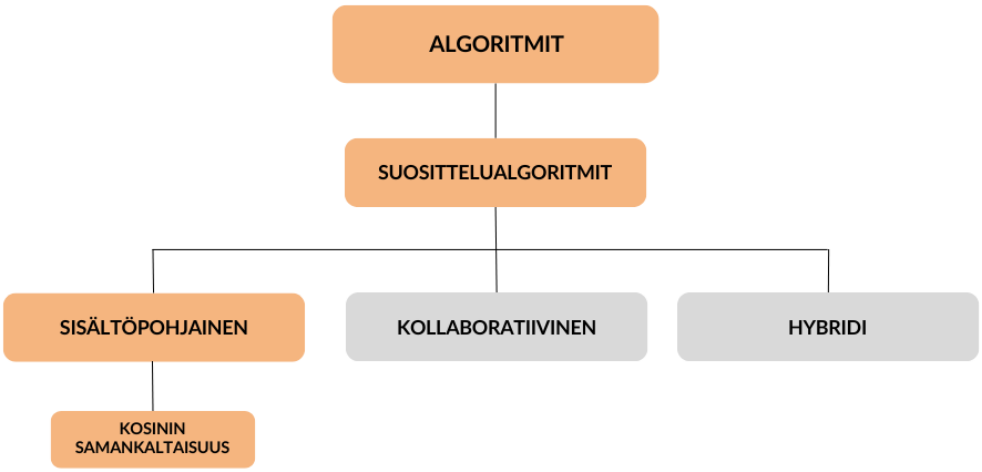
Algoritmit voivat käsitellä erilaisia tehtäviä, kuten lukujen järjestelemistä tai käyttäjän sisällön suosittelemista esimerkiksi verkkokaupoissa tai sosiaalisessa mediassa [2]. Nämä algoritmit määrittävät muun muassa sen, miten sisältö suodatetaan, arvostellaan, valitaan ja suositellaan käyttäjille [3].

Suosittelualgoritmeissa painopiste siirtyy kohti sisällön optimointia käyttäjien kiinnostusten ja käyttäytymisen perusteella.

Suosittelualgoritmit tai suosittelujärjestelmä (recommendation systems) on koneoppimisjärjestelmän tyyppi, joka tarjoaa personoituja suosituksia käyttäjille heidän aiempien toimintojensa, mieltymystensä ja käyttäytymismallien perusteella [4]. Nämä suositukset voivat perustua erilaisiin kriteereihin, kuten aiempiin ostoksiin, hakuhistoriaan, demografisiin tietoihin ja muihin tekijöihin [5].

Niitä käytetään laajasti sähköisessä kaupankäynnissä, sosiaalisessa mediassa ja muilla verkkopalvelualustoilla käyttäjien asiakastyytyväisyyden parantamiseksi sekä myynnin ja tuloksen kasvattamiseksi [4]. Ne mahdollistavat yrityksille henkilökohtaisen ja relevantin sisällön tarjoamisen käyttäjilleen [6].

Erilaisia suosittelualgoritmeja on valtava määrä. Useimmat niistä voidaan kuitenkin jakaa näihin luokkiin (kuva 1): yhteistoiminnallinen suodatus (Collaborative Filtering), sisältöpohjainen suodatus (Content-based Filtering) ja hybridisuodatus (Hybrid Filtering) [5]. Tässä työssä keskitytään sisältösuodatukseen.



Kuva 1. Hahmotuskartta algoritmeista kosinin samankaltaisuuteen.

Sisältöpohjainen suodatus eli Content-Based Filtering (CBF) hyödyntää koneoppimisalgoritmeja ennustaakseen ja suositellakseen käyttäjälle uusia, mutta samankaltaisia kohteita [7].

Algoritmi analysoi tietokannassa olevien kohteiden ominaisuuksia ja sovittaa ne käyttäjän profiiliin. Kohteiden ominaisuuksia voivat olla esimerkiksi elokuvan genre, näyttelijät ja tuottaja, verkkokauppatuotteen tiedot tai Youtube-videon tiedot kuten pituus, kieli ja aihe.

Sisältöpohjainen suodatus määrittää tietyt attribuutit tietokannassa oleviin tuotteisiin, jolloin algoritmit voivat luokitella jokaisen tuotteen ja analysoida käyttäjien käyttäytymistä. Esimerkiksi Amazon analysoi attribuutteja, kuten tuotteen nimiä, kuvauksia ja ominaisuuksia. [8.]

Kun käyttäjä osoittaa kiinnostusta älypuhelimeen, jossa on korkea kameran resoluutio tai pitkä akun kesto, järjestelmä suosittelee samankaltaisia puhelimia vastaavilla ominaisuuksilla. Se saattaa suositella lisäksi puhelinkoteloita tai muita tarvikkeita, jotka vastaavat käyttäjän mieltymyksiä. Näin Amazon käyttää

sisältöpohjaista suodatusta räätälöimään suosituksia käyttäjän mieltymysten mukaan. [8.]

Näin ollen tuotteiden suositteleminen niiden ominaisuuksien perusteella on mahdollista vain, jos tuotteelle on selkeästi määritelty joukko ominaisuuksia ja käyttäjän valintoja [7].

Suosittelujärjestelmä tallentaa käyttäjätiedot, kuten käyttäjän klikkaukset, arvostelut ja tykkäykset, luodakseen käyttäjäprofiilin. Mitä enemmän asiakas on vuorovaikutuksessa käytettävän alustan kanssa, sitä tarkempia ovat tulevat suositukset [7].

Käyttäjäprofiili muodostuu siis käyttäjän vuorovaikutuksista - ostoksista, tykkäyksistä, ei-tykkäyksistä, hauista ja klikkauksista. Attribuutit, jotka esiintyvät useissa vuorovaikutuksissa, saavat korkeamman painoarvon, mikä osoittaa niiden tärkeyden käyttäjän mieltymysten kannalta. Painoarvo tarkentuu jatkuvasti esimerkiksi käyttäjien antaman palautteen kuten tuotearvioiden perusteella.

Yleisesti ottaen sisältöpohjaisessa suodatuksessa käytetään usein kosinin samankaltaisuuteen (Cosine Similarity) tai luokitteluun (Classification) perustuvia menetelmiä. Näiden lisäksi on muita tekniikoita, kuten esimerkiksi TF-IDF (Term Frequency-Inverse Document Frequency), vektoriavaruusmalli (Vector space Model), ominaisuustekniikat (Feature Engineering), neuroverkot (Neural Networks), semanttinen analyysi (Semantic Analysis) ja ensemble-menetelmät (Ensemble methods). Näitä tekniikoita voidaan käyttää joko erikseen tai yhdistelmänä sisältöpohjaisen suosittelualgoritmin kehittämiseksi. Keskitymme kuitenkin tässä työssä kosinin samankaltaisuuteen, jota hyödynnetään myöhemmin kehitettävässä suosittelualgoritmissa matkasuunnitelmasovellukselle.

2.1 Kosinin samankaltaisuus

Kosinin samankaltaisuus (Cosine similarity) mittaa samankaltaisuutta kahden vektorin välillä vektoriavaruudessa [9]. Se mittaa vektorien välisen kulman kosinin arvon ja määrittää, osoittavatko vektorit suunnilleen samaan suuntaan.

Kosinin samankaltaisuutta käytetään laajasti datatieteessä kuten dokumenttien samankaltaisuuden mittaamiseen tekstianalyysissä [9] sekä koneoppimisen sovelluksissa kuten suositusjärjestelmissä [10].

Kosinin samankaltaisuus määritellään jakamalla vektoreiden pistetulo niiden magnitudilla. Esimerkiksi vektorin A ja vektorin B samankaltaisuus lasketaan seuraavasti:

$$samankaltaisuus(A,B) \;=\; cos(\theta) \;=\; \frac{A \cdot B}{\big||A|\big| \cdot \big||B|\big|}$$

jossa,

- θ, on vektoreiden välinen kulma
- $A \cdot B$, on vektoreiden A ja B pistetulo. Lasketaan seuraavasti

  $$A \cdot B = \sum_{i=1}^{n} A_i B_i \; = A_1 B_1 + A_2 B_2 + .. + A_n B_n$$

- $\big||A|\big|$, on vektorin pituus, joka lasketaan seuraavasti

  $$\big||A|\big| \cdot \big||B|\big| = \sqrt{\textstyle\sum_{i=1}^{n} A_i^2}\ \sqrt{\textstyle\sum_{i=1}^{n} B_i^2} = \sqrt{A_1^2 + A_2^2 + .. + A_n^2} \cdot \sqrt{B_1^2 + B_2^2 + .. + B_n^2}$$

Samankaltaisuuden mahdolliset arvot ovat välillä -1 ja 1. Mitä pienempi kulma on vektorien välillä, sitä suurempi on kosinin arvo, mikä tarkoittaa suurempaa kosinin samankaltaisuutta. [10.]

Tämä perustelu voidaan esittää seuraavasti:

Kun kahdella vektorilla on sama suunta, kulma on 0 ja samankaltaisuus on 1. Vektorit ovat siis samanlaisia.

$$cos(0) = 1$$

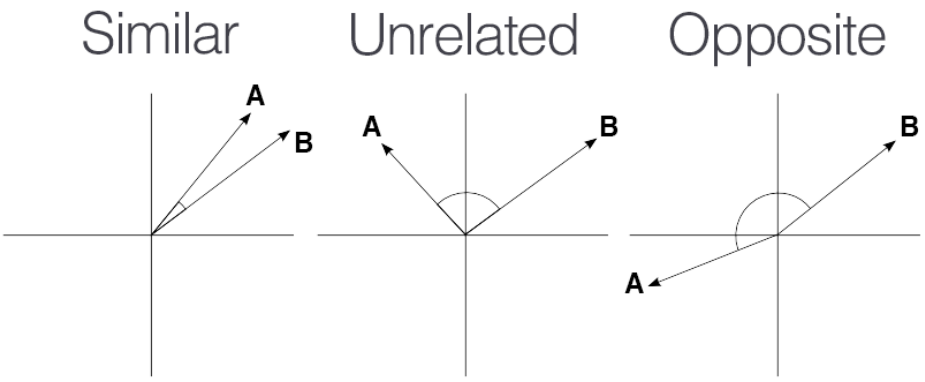
Kohtisuorat vektorit ovat 90 asteen kulmassa toisiinsa nähden, ja niiden kosinin samankaltaisuus on 0.

$$cos(90) = 0$$

Vastakkaisilla vektoreilla on 180 asteen kulma toisiinsa nähden. Näin ollen niiden kosinin samankaltaisuus on -1. Vektorit ovat näin ollen toisensa vastakohtia

$$cos(180) = -1$$

Kuvassa 2 on esitetty kosinin samankaltaisuuden eri vektorisuunnat.



Kuva 2. Kosinin samankaltaisuuden eri vektorisuunnat [11].

Kun kaikki vektoreissa olevat painokertoimet ovat positiivisia, liikutaan siinä osassa koordinaatistoa, jossa vektorien välinen kulma vaihtelee 0-90 asteen välillä. Kosini voi tämän seurauksena saada vain nollan ja yhden välillä olevia arvoja.

Kosinin samankaltaisuuden avulla voidaan arvioida esimerkiksi eri elokuvien tai tuotteiden välisiä suhteita suosittelualgoritmeissa. Tarkastellaan esimerkiksi ti-

lannetta, jossa x-akseli edustaa elokuvan toimintaominaisuutta ja y-akseli elokuvan komediaominaisuutta. Vertaillaan elokuvia Batman, Mr. Bean ja Kung Fu Panda ja sitä, mitä algoritmi suosittelisi käyttäjälle.

Oletetaan, että Batman-elokuvan vektori on (8, 0), sillä se sisältää paljon toimintaa, mutta ei huumoria ja Mr. Bean-elokuvan vektori on (1,8), joka puolestaan sisältää paljon huumoria mutta erittäin vähän toimintaa. Tässä tapauksessa näiden kahden elokuvan välisen kosinin samankaltaisuuden arvo on:

$$\frac{8 \cdot 0 \ + \ 1 \cdot 8}{\sqrt{8^2 + 0^2} \cdot \sqrt{1^2 + 8^2}} \approx \ 0{,}12$$

Samankaltaisuus näiden elokuvien välillä on melko alhainen, mikä tarkoittaa, että ne eroavat paljon toisistaan. Jos käyttäjä on katsonut yhtä näistä elokuvaa, ei ole suositeltavaa suositella toista vaihtoehtoa tällä perusteella. Jos taas verrataan Batman-elokuvan huumoria ja toimintaa sisältävään Kung Fu Panda -elokuvaan, jonka vektori olisi (7,7), näiden kosinin samankaltaisuuden arvo on:

$$\frac{8 \cdot 0 \ + \ 7 \cdot 7}{\sqrt{8^2 + 0^2} \cdot \sqrt{7^2 + 7^2}} \approx \ 0{,}70$$

Samankaltaisuus on siis suurempi, mikä osoittaa, että nämä elokuvat ovat ominaisuuksiltaan lähempänä toisiaan. Tämä tekee Kung Fu Panda -elokuvan suosittelemisesta todennäköisempää käyttäjälle, joka on katsonut Batman-elokuvan.

Yksi kosinin samankaltaisuuden eduista on sen riippumattomuus vektorien pituuksista, sillä ainoastaan vektorien välisellä kulmalla on merkitystä. Näin ollen, esimerkiksi elokuvien ominaisuudet on ehkä mitattu asteikolla 0-10, mutta käyttäjäprofiilin arvot voivat kasvaa rajattomasti. Tämä tekee kosinin samankaltaisuudesta erittäin hyödyllisen suosittelujärjestelmissä, joissa käyttäjien mieltymysten ja kohteiden ominaisuuksien välinen suhde voi vaihdella suuresti.

2.2 Vahvuudet ja heikkoudet

Sisältöpohjainen suosittelujärjestelmä tarjoaa useita etuja käyttäjilleen. Ensinnäkin se skaalautuu helposti suurelle määrälle käyttäjiä, koska suositusten tekemiseen ei tarvita muiden käyttäjien tietoja [12]. Tämä tekee järjestelmästä tehokkaan ja skaalautuvan, mikä on erityisen tärkeää suurissa käyttäjäkannoissa. Lisäksi koska suositukset perustuvat käyttäjän päivittäisiin toimintoihin, ovat kaikki suositukset ja parametrit tarkasti säädettyjä käyttäjän valintaan. Tämän ansiosta malli pystyy suosittelemaan myös erikoistuneita tuotteita, joista muut käyttäjät eivät ehkä ole kiinnostuneita [7].

Toinen merkittävä etu on se, että uusimpia tuotteita voidaan ehdottaa käyttäjälle heti niiden julkaisun jälkeen, koska tuotteiden ominaisuudet ovat heti saatavilla. Tämä tekee järjestelmästä dynaamisen ja ajantasaisen, ja se pystyy tarjoamaan käyttäjilleen tuoreita suosituksia ilman viivettä.

Vaikka sisältöpohjainen suosittelujärjestelmä tarjoaa lukuisia etuja, sillä on myös joitakin heikkouksia. Ensinnäkin sen rakentaminen vaatii paljon tietämystä ja ymmärrystä aihealueesta, johon suositukset liittyvät, koska tuotteiden ominaisuudet on pääosin koodattu järjestelmään. Tämän vuoksi algoritmin tehokkuus ja tarkkuus riippuvat suuresti siitä, kuinka hyvin kehittäjä tuntee ja ymmärtää käsiteltävän aihealueen. [7.] Toiseksi, uusien alueiden löytäminen ja laajentuminen, jotka voisivat kiinnostaa käyttäjää, ei ole mahdollista, koska malli suosittelee uusia tuotteita käyttäjän nykyisen kiinnostuksen perusteella [12].

Lisäksi, kylmäkäynnistysongelma on merkittävä haitta, koska järjestelmällä ei ole riittävästi tietoa uuden käyttäjän mieltymyksistä suositusten tekemiseen. Tämä voi rajoittaa järjestelmän tehokkuutta uusien käyttäjien kohdalla.

Kylmäkäynnistysongelma (cold start problem) viittaa haasteeseen, jossa järjestelmä yrittää tuottaa suosituksia uusille käyttäjille tai uusilla kohteilla, joista sillä ei ole riittävästi dataa. Tämä ongelma on tyypillinen suosittelujärjestelmissä, jotka

perustuvat käyttäjien aikaisempiin toimintoihin suositusten generoimiseksi. Sisältöpohjaisissa suosittelualgoritmeissa ongelma ilmenee tyypillisesti, kun uusille käyttäjille ei pystytä tarjoamaan kohdennettuja suosituksia heidän mieltymystietojen puuttuessa ja uusia kohteita ei osata suositella tehokkaasti, koska käyttäjäpalautetta ei vielä ole saatavilla. Näin ollen algoritmit joutuvat turvautumaan yleisiin suosituksiin tai alustaviin tietoihin.

Lisäksi, on vaikea tehdä uusia suosituksia käyttäjille, jotka eivät ole aktiivisia käyttäjiä, mikä voi johtaa suositusten puutteeseen tai epätarkkuuteen. Lopuksi sisältöpohjaisen suosittelujärjestelmän käyttö on mahdollista vain, jos tuotteelle on selkeä joukko ominaisuuksia ja käyttäjän valintojen lista. [7.]

3 Matkasuunnitelmasovellus

Matkasuunnitelmasovellus on sovellus, jonka tarkoituksena on auttaa käyttäjiä löytämään personoituja vierailukohteita haluttuun matkakohteeseen. Sovellus pyrkii tarjoamaan räätälöityjä vierailukohteita hyödyntämällä Google Places API:a suodattaakseen ja näyttääkseen käyttäjälle suosittuja vierailukohteita. Käyttäjä syöttää sovellukseen matkakohteen, jonka jälkeen sovellus luo listan potentiaalisista vierailukohteista.

Erityisenä ominaisuutena sovelluksessa on mahdollisuus interaktiivisesti muokata luotua matkasuunnitelmaa. Käyttäjät voivat poistaa listalta kohteita, jotka eivät ole heidän mieltymystensä mukaisia, ja näin päivittää kohteita vastaamaan paremmin heidän toiveitaan. Tämä mahdollistaa yksilöllisen matkasuunnitelman luomisen ja varmistaa, että käyttäjät voivat löytää ja valita heitä eniten kiinnostavat kohteet.

Sovellukseen syötetään ainoastaan matkakohde, joten aluksi järjestelmällä ei ole vielä mitään tietoa käyttäjän preferensseistä. Tämän vuoksi sovellukseen kehitetään sisältöpohjainen suosittelujärjestelmä, joka hyödyntää kosinin samankaltaisuutta. Tämä algoritmi auttaa suosittelemaan uusia kohteita käyttäjille heidän

aloitettuaan interaktiivisen toiminnan sovelluksen kanssa, poistamalla ei-kiinnostavia vierailukohteita listalta. Tavoitteena on tarjota käyttäjille entistä personoidumpia ja relevantimpia suosituksia matkakohteisiin.

4 Käsiteltävä data

Datasettinä käytetään Google Places API:sta saatua dataa. Google Places API on palvelu, joka vastaanottaa sijaintietojen HTTP-pyyntöjä. Se palauttaa formatoituja sijaintitietoja ja kuvia eri maantieteellisistä paikoista, nähtävyyksistä sekä yrityksistä. [13.]

Yksi Google Places API:n tarjoamista toiminnoista on Nearby Search -haku, joka mahdollistaa paikkojen etsimisen määritellyltä alueelta. Tähän hakuun on sisällytettävä sijainti, joka määritetään antamalla leveys- ja pituuspiiri sekä säde metreissä. Näiden lisäksi muita valinnaisia parametrejä, joilla voi tarkentaa hakupyyntöä, ovat avainsanat, kieli, minimi- ja maksimihintataso asteikolla 0–4, kohteen tyyppi, tulosten järjestys tai tieto siitä, onko kohde hakuhetkellä avoinna [14].

4.1 Datan hakeminen

Datan hakuun käytettävä Nearby Search -haku palauttaa oletuksena korkeintaan 20 hakukriteerejä vastaavaa paikkaobjektia. Mikäli hakuun ei sisällytetä mitään valinnaisia kriteerejä, tulokset järjestetään niiden näkyvyyden perusteella etäisyyden sijaan. Näkyvyyteen vaikuttavat tekijät, kuten paikan sijoitus Googlen hakemistossa ja maailmanlaajuinen suosio, joista Google ei tarjoa tarkempia tietoja. Tämä tarkoittaa sitä, että tulokset sisältävät ensimmäisenä paikat, jotka ovat tunnetumpia tai huomionarvoisempia. Tulokset voidaan saada joko JSON- tai XML-muodossa, mutta tässä käytämme JSON-muotoa sen yhteensopivuuden vuoksi JavaScriptin kanssa.

Nearby Search -haku tarjoaa erilaisia paikkatyyppejä, joista valitaan. Tähän tarkoitukseen on valittu 12 paikkatyyppiä (kuva 3) saatavilla olevista tyypeistä. Vali-

koidut tyypit ovat ravintolat, kahvilat, baarit, kirkot, ostoskeskukset, puistot, luontokohteet, taidegalleriat, museot, maamerkit, turistikohteet sekä huvipuistot. Tarkoituksena on keskittyä turistiystävällisiin kohteisiin. Nämä tyypit edustavat kategorioita, jotka ovat suosittuja kohteita uutta kaupunkia tai aluetta tutkittaessa.

```
const allPossibleTypes = [
            'restaurant', 'cafe', 'bar', 'church', 'shopping_mall','park',
      'natural_feature', 'art_gallery', 'museum', 'landmark','tourist_at-
      traction', 'amusement_park'
];
```

Kuva 3. Valikoidut paikkatyypit.

Uuden käyttäjän luomisen yhteydessä tehdään erillinen API-kutsu kullekin tyypille. Konfiguraatiotiedostoon ‘config.json’ on tallennettu arvoja, joita ei ole tarkoitus muuttaa, kuten API-avain ja säteen arvo, joka on määritetty 10 km:ksi. Sijainti määritellään käyttäjän antamana ja tyyppeinä käytetään edellä mainittuja. Kuvassa 4 on funktio, jolla tehdään Nearby Search -haku.

```
async function makeNearbySearchRequest(location, type) {
	const params = new URLSearchParams({
			key: config.apiKey,
			location: location,
			radius: config.radius,
			type: type,
	});

	try {
		const response = await fetch(`https://maps.google-
	leapis.com/maps/api/place/nearbysearch/json?${params}`);
	if (!response.ok) {
			throw new Error(`Failed to fetch data: ${response.sta-
	tus}`);
	}
			return await response.json();
	} catch (error) {
			console.error('Error:', error);
			throw error;
	}
}
```

Kuva 4. Funktio, jolla API-kutsu tehdään.

Esimerkkikoodissa 1 on esimerkki yhdestä API-kutsun palauttamasta objektista ja sen attribuuteista ennen esikäsittelyä. Objekti sisältää muun muassa kohteen nimen, osoitteen, geometriset koordinaatit, paikkatyypit ja tiedon siitä, onko paikka auki hakuhetkellä, arvostelun, hintatason ja kuvat.


```
{
"business_status": "OPERATIONAL",
"geometry": {
"location": {
"lat": 60.16890069999999,
"lng": 24.9414597
},
"viewport": {
"northeast": {
"lat": 60.17049867989273,
"lng": 24.94364025
},
"southwest": {
```

```
"lat": 60.16779902010729,
"lng": 24.94073285000001
}
}
},
"icon": "https://maps.gstatic.com/mapfiles/place_api/icons/v1/png_71/restaurant-71.png",
"icon_background_color": "#FF9E67",
"icon_mask_base_uri": "https://maps.gstatic.com/mapfiles/place_api/icons/v2/restaurant_pinlet",
"name": "Kiila Food & Bar",
"opening_hours": {
"open_now": true
},
"photos": [
{
"height": 1080,
"html_attributions": [
"<a href=\"https://maps.google.com/maps/contrib/100833178962452800213\">World.Citizen Travel.Diaries</a>"
],
"photo_reference": "ATplDJbvwbWfxs8S25An1UY5nQm2eLMPvHkjr62qXC3YCWpRmMQZ3vxto2KYcAcLVZrKgNJqGZr6o-vGCK-honRjjY7Ofqu7D7Sn2pAgGBiW1kwlchJ51p1ISD9IBnlF6PKOseJKfTxQ2nAtYeblPiU_LYXn1eLxys8m6bbztUdVrcW4Lsmb",
"width": 1920
}
],
"place_id": "ChIJExa9ecwLkkYRfFjZ6gFDGU4",
"plus_code": {
"compound_code": "5W9R+HH Helsinki, Finland",
"global_code": "9GG65W9R+HH"
},
"price_level": 2,
"rating": 3.9,
"reference": "ChIJExa9ecwLkkYRfFjZ6gFDGU4",
"scope": "GOOGLE",
"types": [
"restaurant",
"bar",
"point_of_interest",
"food",
"establishment"
],
"user_ratings_total": 1309,
```

```
"vicinity": "Aleksanterinkatu 21, Helsinki"
}
```

Esimerkkikoodi 1. API-kutsun palauttama objekti ennen esikäsittelyä.

Kuten objektista näkyy, attribuutteja on paljon. Kaikkia attribuutteja ei sovellukseen tarvita, joten seuraavaksi data pitää esikäsitellä.

4.2 Datan esikäsittely

Sovellusta varten relevantteja attribuutteja ovat kohteen nimi, arvosana, tyypit, arvostelujen lukumäärä sekä kohteen osoite. API-kutsun jälkeen, muut paitsi edellä mainitut attribuutit suodatetaan pois kutsun palauttamasta datasta. Kuva 5 esittää funktiota, jolla tarpeettomat attribuutit suodatetaan.

```
function extractRelevantAttributes(results) {
	return results.map(place => {
		const { name, rating, types, user_ratings_total, vicin-
	ity } = place;
		return { name, rating, types, user_ratings_total, vi-
	cinity };
	});
}
```

Kuva 5. Funktio, jolla irrelevantit attribuutit suodatetaan pois API-kutsun palauttamasta datasta.

Yksi paikkaobjekti voi sisältää monia tyyppejä. Siksi täytyy saaduista tuloksista suodattaa ne paikat, jotka on jo tallennettu jo aiemmin,

Haettu data suodatetaan myös jo tallennetuilta kohteilta sekä majoituspaikoilta, jotka eivät ole sovelluksen kannalta olennaisia. Kohteet, jotka voivat olla jo tallennettu, ovat esimerkiksi kohteet, joiden paikkatyyppiin kuuluu enemmän kuin yksi paikkatyyppi, joita API-kutsulla haetaan. Kuvassa 6 näkyvällä funktiolla tehdään kaikki esikäsittelyssä mainitut operaatiot sekä itse API-kutsu kaikille aiemmin mainituille paikkatyypeille.

```
async function fetchPlacesByLocation(location) {
       let results = [];
       let seenNames = new Set();

       for (let type of allPossibleTypes) {
                do {
           try {
             const response = await makeNearbySearchRequest(location, type);
             const  relevantPlaces  =  extractRelevantAttributes(response.re-
sults);
             relevantPlaces.forEach(place => {
               if     (!seenNames.has(place.name)     &&     !place.types.in-
cludes('lodging')) {
                 results.push(place);
                 seenNames.add(place.name);
               }
             });
           } catch (error) {
             console.error(`Failed to fetch data for type ${type}:`, error);
             break;
           }
         } while (false);
       }
       return results;
}
```

Kuva 6. Funktio, joka sisältää API-kutsun ja datan suodatuksen.

5 Suosittelujärjestelmä sovellukselle

Algoritmin toteutuskielenä on JavaScript, koska käsiteltävän datan määrä ei ole erityisen suuri ja tarkoituksena on myöhemmin kehittää web-sovellus. JavaScriptin käyttö mahdollistaa sujuvan integraation web-sovelluksen kehitykseen.

5.1 Tavoite

Tavoitteena on kehittää kosinin samankaltaisuuteen perustuva algoritmi, joka optimoi suositusprosessin matkasuunnittelusovelluksessa. Algoritmi otetaan käyttöön sen jälkeen, kun käyttäjä on syöttänyt matkakohteensa sijainnin ja sovellus

on luonut tämän perusteella listan kymmenestä potentiaalisesta vierailukohteesta. Saatuaan listan käyttäjät voivat tarkentaa mieltymyksiään poistamalla kiinnostamattomat kohteet päivittäen jatkuvasti järjestelmän käsitystä heidän preferensseistään. Tämä vuorovaikutus auttaa sovellusta mukauttamaan suosituksensa tarjoamaan yhä relevantimpia ja henkilökohtaisempia ehdotuksia.

Algoritmin soveltuvuus voidaan laajentaa myös käyttäjien tykkäysten käsittelyyn, jolloin tykättyjen kohteiden kategorioiden painoarvo kasvaa käyttäjän mieltymysvektorissa. Tässä toteutuksessa keskitytään kuitenkin ainoastaan algoritmin poisto-ominaisuuden kehittämiseen.

5.2 Prototyypin toteutus

5.2.1 Käyttäjäprofiili

Käyttäjäprofiili koostuu seuraavista attribuuteista, jotka ovat käyttäjän id, käyttäjän syöttämä matkakohteen sijainti, mieltymysvektori, lista mahdollisista vierailukohteista, top 10 -lista sekä lista poissuljetuista kohteista.

Uuden käyttäjäprofiilin luomisen yhteydessä luodaan käyttäjän mieltymysvektori, joka sisältää jokaisen paikkatyypin alustettuna neutraalilla arvolla 0,5 (kuva 7). Tämä mieltymysvektori päivittyy käyttäjän vuorovaikutuksen myötä.

```
function createUserPrefVector() {
	return allPossibleTypes.map(() => 0.5);
}
```

Kuva 7. Funktio, jolla mieltymysvektori alustetaan.

Lista mahdollisista vierailukohteista sisältää aiemmin esikäsitellyn datan paikoista. Top 10 -lista koostuu kyseisen listan kymmenestä ensimmäisestä vierailukohteesta. Poissuljetuttujen kohteiden listaan kerätään kaikki käyttäjän myöhemmin poistamat paikat, joita tämä ei halua top 10 -listalleen.

Käyttäjäprofiilin luonti tapahtuu kuvan 8 mukaisella funktiolla. Luonnin yhteydessä suoritetaan API-kutsut, joiden perusteella luodaan lista mahdollisista vierailukohteista ja siitä johdetusti top 10 -lista.

```
async function createNewUser(id, location) {
	const fetchedPlaces = await fetchPlacesByLocation(location);
	const flatPlaces = Object.values(fetchedPlaces).flat();
	const vectorizedPlaces = await vectorizePlaces(flatPlaces);
	const sortedPlaces = sortPlaces(vectorizedPlaces);
	const top10Lists = getTop10(sortedPlaces);

	usersData[id] = {
			location,
			preferenceVector: createUserPrefVector(),
			places: vectorizedPlaces,
			top10Lists: top10Lists,
			excludedPlaces: []
	};
```

Kuva 8. Käyttäjäprofiilin luontifunktio.

Kuvassa 9 on esimerkki uuden käyttäjäprofiilin perusmuodosta ennen vuorovaikutuksia sovelluksen kanssa. Käyttäjäprofiilin esittämisen yksinkertaistamiseksi profiilista on jätetty pois 'top10Lists' ja 'places' -attribuuttien sisällöt.

```
"1": {
              "location":"60.1699,24.9384",
       "preferenceVector": [
       0.5,
       0.5,
       0.5,
       0.5,
       0.5,
       0.5,
       0.5,
       0.5,
       0.5,
       0.5,
       0.5,
       0.5
       ],
       "places": [
       ],
       "top10Lists": [
       ],
       "excludedPlaces": []
       }
```


Kuva 9. Yksinkertaistettu uusi käyttäjäprofiili.

5.2.2 Paikkojen vektorointi ja tulosten järjestely

Paikkaobjektit vektoroidaan muuntamalla niiden tyypit numeeriseen muotoon. Tämä tehdään kuvassa 10 näkyvillä funktioilla.

```
function encodeTypes(inputTypes) {
	return allPossibleTypes.map(type => inputTypes.includes(type) ? 1 :
0);
}

function vectorizePlaces(places) {
	return places.map(place => ({
			...place,
			vector: encodeTypes(place.types)
	}));
}
```

Kuva 10. Funktiot, jolla vektoroidaan paikkojen tyypit.

Kuvassa 11 on esimerkki vektoroidusta paikkaobjektista.


```
{
"name": "Helsinki Cathedral",
"rating": 4.6,
"types": [
	"tourist_attraction",
	"church",
	"place_of_worship",
	"point_of_interest",
	"establishment"
],
"user_ratings_total": 10814,
"vicinity": "Unioninkatu 29, Helsinki",
"vector": [
	0,
	0,
	0,
	1,
	0,
	0,
	0,
	0,
	0,
	0,
	1,
	0
 ]
},
```

Kuva 11. Vektoroitu paikkaobjekti.

Kuvan 11 objektin vektorissa on kaksi arvoa, 1, jotka vastaavat tyyppejä ‘church’ ja ‘tourist_attraction’.

Kun käyttäjäprofiili luodaan, suoritetaan jokaiselle paikkatyypille erillinen API-kutsu. Vaikka API palauttaa tulokset oletusjärjestyksessä suositummuuden perusteella, tämä järjestys pätee vain kunkin tyypin sisällä, ei koko listassa. Tästä syystä paikat järjestellään uudelleen kaikkien tyyppien API-kutsujen jälkeen kuvassa 12 olevalla funktiolla.

```
function sortPlaces(places) {
	return places.sort((a, b) => {
		const scoreA = a.rating * Math.log(a.user_ratings_total
	+ 1);
		const scoreB = b.rating * Math.log(b.user_ratings_total
	+ 1);
		return scoreB - scoreA;
	});
}
```

Kuva 12. Funktio, joka järjestelee paikat suositummuuden mukaan.

Funktio järjestää paikat uudelleen niiden arvion ja käyttäjien arvioiden kokonaismäärän mukaan laskevaan järjestykseen. Jokaiselle paikalle lasketaan pistemäärä seuraavasti:

$$pistemäärä = rating \times ln(userRatingsTotal + 1)$$

jossa

- rating on paikan saama arvosana
- userRatingsTotal on käyttäjien antamien arvioiden kokonaismäärä.

Arvioiden kokonaislukumäärään lisätään 1 varmistamaan, etteivät paikat, joissa ei ole ollenkaan arvosteluja, saa arvoksi log(0).

Tämä menetelmä varmistaa myös, että esimerkiksi paikka, jolla on arvosana 4 mutta arvosteluja 200, saa paremman pistemäärän kuin vastaavan arvosanan paikka, jolla on vain 10 arvostelua, sillä suurempi arvostelumäärä tekee pistemäärästä luotettavamman.

5.2.4 Top 10 -lista

Tulosten uudelleenjärjestelyn jälkeen valitaan listalta ensimmäiset kymmenen paikkaa, jotka muodostavat käyttäjälle näkyvän top 10 -listan. Tältä listalta käyttäjä voi myöhemmin poistaa häntä kiinnostamattomat kohteet. Tämä toteutetaan kuvan 13 mukaisella funktiolla.

```
function getTop10(places) {
       let top10Lists;
                if (places.length > 10) {
                  top10Lists = places.slice(0,
       10);
           places.splice(0, 10);
         } else {
           top10Lists = places.slice();
           places.splice(0, places.length);
         }
       return top10Lists;
}
```

Kuva 13. Funktio, joka valitsee paikkalistalta ensimmäiset kymmenen paikkaa järjestyksen mukaan.

5.2.5 Suositusten dynaaminen päivitys

Paikan poistamisen ja uuden paikan ehdottamisen prosessi alkaa käyttäjätietojen lataamisella. Järjestelmä lataa olemassa olevat käyttäjätiedot JSON-tiedostosta loadUserData-nimisellä funktiolla (kuva 14).

```
async function loadUserData() {
       try {
                if (fs.existsSync(userDataPath)) {
           const rawData = fs.readFileSync(userDataPath, 'utf8');
           const data = JSON.parse(rawData);
           Object.assign(usersData, data.users);
         } else {
           console.log("No user data file found.");
                }
       } catch (err) {
         console.error('Error loading user data:', err);
       }
}
```

Kuva 14. Käyttäjätietojen latausfunktio.

Paikan poistamisprosessi aloitetaan lisäämällä poistettava paikka käyttäjän ‘excludedPlaces’-listalle, ettei samaa paikkaa suositella enää myöhemmin. Tämän jälkeen käyttäjän mieltymysvektoria säädetään poistetun paikan vektorin perusteella (kuva 15). Jos paikan vektorissa on kategorian arvo 1, vähennetään kyseisestä kategoriasta mieltymysvektorissa 10 %. Tämä heijastaa käyttäjän vähentynyttä mielenkiintoa kyseistä kategoriaa kohtaan.

```
function adjustPreferenceVector(userVector, deletedVector) {
       for (let i = 0; i < deletedVector.length; i++) {
         if (deletedVector[i] === 1) {
           userVector[i] *= 0.9;
           }
         }
       return userVector;
}
```

Kuva 15. Funktio, jolla säädetään käyttäjän mieltymysvektoria poistetun paikkavektorin perusteella.

Määritetty paikka poistetaan sen jälkeen käyttäjän top 10 -suosituksista, ja tilalle ehdotetaan uutta paikkaa.

Uuden paikan suosittelu perustuu jäljellä olevien paikkojen kosinin samankaltaisuuden arviointiin käyttäjän päivitetyssä mieltymysvektorissa.

5.2.6 Kosinin samankaltaisuuden laskeminen

Kullekin jäljellä olevalle paikalle lasketaan kosinin samankaltaisuus vertaamalla paikan vektoria käyttäjän mieltymysvektoriin. Kosinin samankaltaisuus lasketaan jakamalla vektorien elementtien pistetulo niiden magnitudien tulolla. Pistetulo saadaan summaamalla vektorien vastaavien elementtien tulot, ja magnitudi on vektorin elementtien neliöiden summan neliöjuuri. Nämä operaatiot tehdään kuvassa 16 näkyvillä funktioilla.

```
function calculateDotProduct(vector1, vector2) {
	let dotProduct = 0;
	for (let i = 0; i < vector1.length; i++) {
		dotProduct += vector1[i] * vector2[i];
	}
	return dotProduct;
}

function calculateMagnitude(vector) {
	let sumOfSquares = 0;
	for (let i = 0; i < vector.length; i++) {
		sumOfSquares += vector[i] * vector[i];
	}
	return Math.sqrt(sumOfSquares);
}

function calculateCosineSimilarity(vector1, vector2) {
	const dotProduct = calculateDotProduct(vector1, vector2);
	const magnitude1 = calculateMagnitude(vector1);
	const magnitude2 = calculateMagnitude(vector2);

	return dotProduct / (magnitude1 * magnitude2);
}
```

Kuva 16. Kosinin samankaltaisuutta laskevat funktiot.

Korkeimman kosinin samankaltaisuuden omaava paikka valitaan uudeksi suositukseksi ja sijoitetaan poistetun paikan paikalle top 10 -listassa kuvan 17 mukaisella funktiolla.

```
async function replaceDeletedPlace(user, deletedIndex) {
       const candidates = user.places.filter(place =>
         !user.excludedPlaces.includes(place.name));

       let highestSimilarity = -1;
       let mostSimilarPlaceIndex = -1;

       for (let i = 0; i < candidates.length; i++) {
         const similarity = calculateCosineSimilarity(user.preferenceVec-
tor, candidates[i].vector);

       if (similarity > highestSimilarity) {
         highestSimilarity = similarity;
         mostSimilarPlaceIndex = i;
       }
}

       if (mostSimilarPlaceIndex !== -1) {
         const mostSimilarPlace = candidates[mostSimilarPlaceIndex];
         user.top10Lists.splice(deletedIndex, 0, mostSimilarPlace);
         user.places.splice(user.places.indexOf(mostSimilarPlace), 1);
       } else {
         console.error("No suitable replacement found.");
       }
}
```

Kuva 17. Uuden suositeltavan paikan sijoitus top 10 -listaan.

Kuvan 17 funktiota kutsutaan kuvan 18 mukaisessa funktiossa. Tässä näkyvä funktio on ylin funktio, jota kutsutaan siinä vaiheessa, kun käyttäjä poistaa top 10 -listalta paikan.

```
async function DeletePlaceAndSuggestNew(userId, deletePlaceName) {
       await loadUserData();
       const user = usersData[userId];

       if (!user) {
         console.error("User not found");
         return;
       }

       const deletePlaceIndex = user.top10Lists.findIndex(place =>
place.name === deletePlaceName);
       const deletePlaceVector = user.top10Lists[deletePlaceIndex].vector;

       if (deletePlaceIndex === -1) {
         console.error("Place not found in top10Lists.");
         return;
       }

       user.excludedPlaces.push(deletePlaceName);
       user.preferenceVector = adjustPreferenceVector(user.preferenceVec-
tor, deletePlaceVector);

       user.top10Lists.splice(foundIndex, 1);

       await replaceDeletedPlace(user, foundIndex);
       saveUserData();
}
```

Kuva 18. Ylin funktio, jota kutsutaan, kun käyttäjä poistaa paikan top 10 -listalta. Kaikki käyttäjädata tallennetaan kuvassa 19 olevalla funktiolla JSON-tiedostoon.

```
function saveUserData() {
	try {
		const data = JSON.stringify({ users: usersData }, null,
	2);
		fs.writeFileSync(userDataPath, data);
	} catch (err) {
		console.error('Error saving user data:', err);
	}
}
```

Kuva 19. Käyttäjätietojen tallennusfunktio.

6 Suosittelujärjestelmän arviointi

Algoritmin toimivuuden testaamiseksi luodaan uusi käyttäjä, joka on kiinnostunut lapsiperheille sopivista paikoista, kuten huvipuistoista ja puistoista. Käyttäjän valitsema sijainti on Helsinki.

Tavoitteena on opettaa algoritmi suosittelemaan käyttäjää kiinnostavia kohteita, eli tässä tapauksessa huvipuistoja ja puistoja. Odotettu tulos on, että nämä paikkatyypit, huvipuistot ja puistot, dominoivat lopulta top 10 -listaa, mikä osoittaisi algoritmin kyvyn tarkentaa ehdotuksiaan käyttäjäpalautteen perusteella.

Aluksi käyttäjälle luotu mieltymysvektori (kuva 20) näyttää neutraalilta.


```
"preferenceVector": [
  0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  0.5,
  0.5
],
```

Kuva 20. Uuden käyttäjän mieltymysvektori.

Käyttäjän alustava top 10 -lista (kuva 21) sisältää sekalaisen valikoiman kohteita.

```
Current Top 10 Places:
0: Helsinki Cathedral (tourist_attraction, church, place_of_worship, point_of_interest, establishment)
1: Temppeliaukio Church (tourist_attraction, church, place_of_worship, point_of_interest, establish-
ment)
2: Kamppi Helsinki (shopping_mall, gym, cafe, clothing_store, home_goods_store, hair_care, health,
electronics_store, restaurant, food, point_of_interest, store, establishment)
3: Sello Shopping Centre (shopping_mall, library, pharmacy, movie_theater, cafe, grocery_or_super-
market, clothing_store, gym, restaurant, food, store, health, point_of_interest, establishment)
4: Uspenski Cathedral (tourist_attraction, church, place_of_worship, point_of_interest, establishment)
5: Forum (shopping_mall, point_of_interest, establishment)
6: Fazer Café Kluuvikatu (cafe, bakery, food, store, point_of_interest, establishment)
7: Itis (shopping_mall, point_of_interest, establishment)
8: Sibelius Park (park, tourist_attraction, point_of_interest, establishment)
9: Kappeli (restaurant, cafe, point_of_interest, food, establishment)
```

Kuva 21. Uuden käyttäjän ensimmäinen top 10 -lista.

Poistetaan ensin kaikki kohteet, jotka sisältävät ‘restaurant’ tai ‘cafe’-tyypit. Mieltymysvektori näyttää operaation jälkeen kuvan 22 mukaiselta. Kuvassa on vierekkäin mieltymysvektori sekä sen arvoja vastaavat tyypit.

```
"preferenceVector": [
  0.2657205000000001,
  0.2657205000000001,
  0.45,
  0.5,
  0.36450000000000005,
  0.5,
  0.5,
  0.45,
  0.405,
  0.5,
  0.45,
  0.5
],
```

```
const allPossibleTypes = [
    'restaurant',
    'cafe',
    'bar',
    'church',
    'shopping_mall',
    'park',
    'natural_feature',
    'art_gallery',
    'museum',
    'landmark',
    'tourist_attraction',
    'amusement_park'
];
```

Kuva 22. Mieltymysvektori sekä sen arvoja vastaavat tyypit ‘restaurant’ ja ‘cafe’-tyyppisten kohteiden poistojen jälkeen.

Kuten nähdään, erityisesti ‘restaurant’ ja ‘cafe’-kategorioiden arvot pienenevät huomattavasti.

Seuraavaksi poistamme kaikki paikat, joiden tyyppi on ‘church’. Mieltymysvektori näyttää tämän jälkeen kuvan 23 mukaiselta.

```
"preferenceVector": [
  0.2657205000000001,
  0.2657205000000001,
  0.45,
  0.32805000000000006,
  0.36450000000000005,
  0.5,
  0.5,
  0.45,
  0.405,
  0.5,
  0.2952450000000001,
  0.5
],
```

```
const allPossibleTypes = [
    'restaurant',
    'cafe',
    'bar',
    'church',
    'shopping_mall',
    'park',
    'natural_feature',
    'art_gallery',
    'museum',
    'landmark',
    'tourist_attraction',
    'amusement_park'
];
```

Kuva 23. Mieltymysvektori sekä sen arvoja vastaavat tyypit ‘church’-tyyppisten kohteiden poistojen jälkeen.

Tämän muutoksen myötä mieltymysvektori näyttää, että ‘church’ -kategorian arvo on laskenut alkuperäisestä arvosta 0,5 arvoon 0,32805.

Poistetaan lopuksi kaikki jäljellä olevat paikat, joissa ei ole ‘amusement_park’ tai ‘park’-tyyppejä. Tämän jälkeen mieltymysvektori näyttää kuvan 24 mukaiselta.

```
"preferenceVector": [
  0.23914845000000007,
  0.23914845000000007,
  0.405,
  0.32805000000000006,
  0.2952450000000001,
  0.5,
  0.5,
  0.45,
  0.405,
  0.5,
  0.2952450000000001,
  0.5
],
```

```
const allPossibleTypes = [
    'restaurant',
    'cafe',
    'bar',
    'church',
    'shopping_mall',
    'park',
    'natural_feature',
    'art_gallery',
    'museum',
    'landmark',
    'tourist_attraction',
    'amusement_park'
];
```

Kuva 24. Mieltymysvektori sekä sen arvoja vastaavat tyypit viimeisen poiston jälkeen.

Mieltymysvektori osoittaa tämän jälkeen, että suurin osa arvoista on laskenut, mutta ‘amusement_park’, ‘landmark’, ‘natural_feature’ ja ‘park’ säilyvät ennallaan tai kasvavat. Tämä kertoo, että nämä luontoon liittyvät tyypit ovat säilyneet listalla alusta asti, koska ne ovat todennäköisesti yhteydessä toisiinsa.

Top 10 -lista näyttää kyseisen vektorin kanssa kuvan 25 mukaiselta.

```
Current Top 10 Places:
0: Kolmikulma (tourist_attraction, park, point_of_interest, establishment)
1: Lauttasaari Outdoor Park (tourist_attraction, park, point_of_interest, establishment)
2: Väinämöinen Park (park, tourist_attraction, point_of_interest, establishment)
3: Alppipuisto (tourist_attraction, park, point_of_interest, establishment)
4: Lenin Park (tourist_attraction, park, point_of_interest, establishment)
5: Hauhopark (park, tourist_attraction, point_of_interest, establishment)
6: Kaisaniemi Park (tourist_attraction, park, point_of_interest, establishment)
7: Linnanmäki (amusement_park, tourist_attraction, point_of_interest, establishment)
8: Sibelius Park (park, tourist_attraction, point_of_interest, establishment)
9: Flying Cinema Tour Of Helsinki (tourist_attraction, amusement_park, cafe, bar, food, point_of_interest, establishment)
```

Kuva 25. Lopullinen top 10 -lista.

Lopullinen top 10 -lista sisältää ainoastaan paikkoja, jotka kuuluvat joko

‘amusement_park’ tai ‘park’-kategorioihin. Tämä testi vahvistaa, että kosinin samankaltaisuusalgoritmi säätää tehokkaasti suosituksia käyttäjän mieltymysten mukaan, mikä parantaa personoitua käyttäjäkokemusta.

Tässä esimerkissä esitetty algoritmin toiminta on vain demonstraatio sen kyvystä sopeutua ja muokata suosituksia käyttäjän mieltymysten mukaisesti. Vaikka algoritmi on osoittanut kykynsä tarkentaa suosituksia käyttäjäpalautteen perusteella, todellisen käyttökelpoisuuden todentaminen vaatisi laajemman käyttäjäkokeilun. Tämä kokeilu käsittäisi systemaattisen arvioinnin useiden eri käyttäjäprofiilien ja matkakohteiden kontekstissa, mikä sisältää käyttäjäpalautteen personoinnin onnistumisesta. Perusteellinen validointi edellyttäisi myös seurantaa siitä, miten käyttäjät ovat vuorovaikutuksessa suositusten kanssa reaalimaailmassa, mikä tarjoaisi kattavampaa tietoa algoritmin suorituskyvystä ja käytännön hyödyllisyydestä.

7 Johtopäätös

Tässä opinnäytetyössä kehitettiin kosinin samankaltaisuuteen perustuva suositusjärjestelmä matkasuunnitelmasovellukselle, joka pyrkii parantamaan käyttäjäkokemusta tarjoamalla henkilökohtaisempia suosituksia nähtävyyksistä ja kohteista. Algoritmin kehitysprosessi hyödynsi Google Places API:n tarjoamaa dataa, mikä mahdollisti ajantasaisen tiedon saamisen kohteista. Tämä tieto yhdistettiin käyttäjien toimintaan sovelluksessa, mikä mahdollisti tarkempien ja personoidumpien suosituksien tarjoamisen käyttäjälle.

Kehitetyssä sovelluksessa suosittelualgoritmi on osoittautunut tehokkaaksi välineeksi käyttäjäkokemuksen räätälöinnissä. Esimerkiksi toteutetut testit, joissa algoritmi mukautettiin suosittelemaan lapsiperheille sopivia paikkoja, kuten huvipuistoja ja puistoja, ovat osoittaneet algoritmin kyvyn dynaamisesti muokata ehdotettuja kohteita käyttäjän mieltymysten mukaan. Tämä näkyy siinä, kuinka valitut paikkatyypit lopulta dominoivat suosituksien top 10 -listaa.

Tämän opinnäytetyön tulokset osoittavat, että kosinin samankaltaisuuteen perustuva suositusjärjestelmä on lupaava työkalu matkailualan sovellusten kehittämisessä. Järjestelmän avulla voidaan tarjota käyttäjille mielekkäitä suosituksia sekä parantaa heidän yleistä tyytyväisyyttään sovelluksen käyttöön, mikä voi johtaa sovelluksen käyttöasteen kasvuun.

Tulevaisuudessa voidaan tutkia esimerkiksi algoritmin yhdistämistä muihin suodatustekniikoihin, kuten kollaboratiiviseen suodatukseen tai hybridi-malleihin, jotka voivat tarjota monipuolisempia keinoja käyttäjäkokemuksen parantamiseksi. Lisäksi voitaisiin keskittyä algoritmin skaalautuvuuden parantamiseen suurissa datamäärissä, joka on välttämätöntä globaaleissa matkailusovelluksissa. Näiden toimien avulla kosinin samankaltaisuuteen perustuva suositusjärjestelmä voisi entistä paremmin vastata matkailualan tarpeisiin ja trendeihin.

Lähteet

1 Kiviranta Pia. 2023. Algoritmien ja AI:n merkitys sosiaalisen median markkinoinnissa. Verkkoaineisto. LinkedIn. <https://www.linkedin.com/pulse/algoritmien-ja-o-markkinoinnin-sosiaalisen-median-pia-kiviranta-shqse?trk=public_post_main-feed-card_feed-article-content>. 24.11.2023. Luettu 17.2.2024.

2 Gillis S. Alexander. 2023. Algorithm. Verkkoaineisto. TechTarget. <https://www.techtarget.com/whatis/definition/algorithm>. 7.2023. Luettu 17.2.2024.

3 Dorcas Adisa. 2023. Everything you need to know about social algorithms. Verkkoaineisto. Sprout Social. <https://sproutsocial.com/insights/social-media-algorithms/#:~:text=In%20social%20media%2C%20algorithms%20are,we%20see%20on%20social%20media>. 30.10.2023. Luettu 18.2.2024.

4 Utsav Desai. 2023. Recommendation Systems Explained: Understanding the Basics to Advance. Verkkoaineisto. Medium. <https://utsavdesai26.medium.com/recommendation-systems-explained-understanding-the-basic-to-advance-43a5fce77c47>. 27.4.2023. Luettu 18.2.2024.

5 Recommendation System. Verkkoaineisto. NVIDIA. <https://www.nvidia.com/en-us/glossary/recommendation-system/>. Luettu 20.2.2024.

6 Exploring Popular Algorithms for Recommendation Systems: Collaborative Filtering & Content-Based. Verkkoaineisto. Medium. <https://medium.com/@andrew_johnson_4/exploring-popular-algorithms-for-recommendation-systems-collaborative-filtering-content-based-aaf08aa5e14e>. Luettu 20.4.2024.

7 A guide to Content-Based Filtering in Recommender Systems. Verkkoaineisto. Turing. <https://www.turing.com/kb/content-based-filtering-in-recommender-systems>. Luettu 20.2.2024.

8 What is Content-Based Filtering? Benefits and Examples in 2024. Verkkoaineisto. Upwork. <https://www.upwork.com/resources/what-is-content-based-filtering>. 5.2.2024. Luettu 20.2.2024.

9 Han, Jiawei; Kamber, Micheline & Pei, Jian. 2012. Data Mining: Concepts and Techniques. E-kirja. Morgan Kaufmann Publishers.

10 Karabiber Faith. Cosine Similarity. Verkkoaineisto. LearnDataSci. <https://www.learndatasci.com/glossary/cosine-similarity/#WhatisCosineSimilarity>. Luettu 3.3.2024.

11 Seelam Sindhu. 2021. Machine Learning Fundamentals: Cosine Similarity and Cosine Distance. Verkkoaineisto. Medium. <https://medium.com/geekculture/cosine-similarity-and-cosine-distance-48eed889a5c4>. 26.4.2021. Luettu 4.3.2024.

12 Content-based Filtering Advantages & Disadvantages. Verkkoaineisto. Google Developers. <https://developers.google.com/machine-learning/recommendation/content-based/summary>. Luettu 4.3.2024.

13 Google Maps Place Web Service Overview. Verkkoaineisto. Google Developers. <https://developers.google.com/maps/documentation/places/web-service/overview>. Luettu 13.4.2024.

14 Google Maps Place Web Service Search Nearby. Verkkoaineisto. Google Developers. <https://developers.google.com/maps/documentation/places/web-service/search-nearby>. Luettu 13.4.2024.