Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR – IS184853

PREDIKSI KETERHUBUNGAN ANTAR PRODUK MAKANAN DENGAN GRAPH EMBEDDING MENGGUNAKAN ALGORITMA RANDOM FOREST

LINK PREDICTION BETWEEN FOOD PRODUCTS WITH GRAPH EMBEDDING USING RANDOM FOREST ALGORITHM

IRFAN RIFQI SUSETYO
NRP. 05211740000064

Dosen Pembimbing
Nur Aini Rakhmawati S.Kom, M.Sc.Eng., Ph.D.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2021

TUGAS AKHIR – IS184853

PREDIKSI KETERHUBUNGAN ANTAR PRODUK MAKANAN DENGAN GRAPH EMBEDDING MENGGUNAKAN ALGORITMA RANDOM FOREST

IRFAN RIFQI SUSETYO
NRP. 05211740000064

Dosen Pembimbing
Nur Aini Rakhmawati S.Kom., M.Sc.Eng., Ph.D.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2021

Halaman ini sengaja dikosongkan

UNDERGRADUATE THESIS – IS184853

LINK PREDICTION BETWEEN FOOD PRODUCTS WITH GRAPH EMBEDDING USING RANDOM FOREST ALGORITHM

IRFAN RIFQI SUSETYO
NRP. 05211740000064

Dosen Pembimbing
Nur Aini Rakhmawati S.Kom, M.Sc.Eng., Ph.D.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Elektro Dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2021

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

PREDIKSI KETERHUBUNGAN ANTAR PRODUK MAKANAN DENGAN GRAPH EMBEDDING MENGGUNAKAN ALGORITMA RANDOM FOREST

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer (S.Kom)
pada
Departemen Sistem Informasi
Fakultas Teknologi Elektro dan Informatika Cerdas (ELECTICS)
Institut Teknologi Sepuluh Nopember

Oleh

Irfan Rifqi Susetyo
05211740000064

Surabaya, 12 Agustus 2021

Kepala Departemen Sistem Informasi

Dr. Majdjahidin, ST., MT.
NIP. 197010102003121001

Halaman ini sengaja dikosongkan

LEMBAR PERSETUJUAN

Prediksi Keterhubungan Antar Produk Makanan Dengan Graph Embedding Menggunakan Algoritma Random Forest

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Oleh :

Irfan Rifqi Susetyo
NRP: 05211740000064

Disetujui Tim Penguji: Tanggal Ujian: 14 July 2021
Periode Wisudas: September 2021

Nur Aini Rakhmawati, S.Kom, M.Sc.Eng, Ph.D (Pembimbing 1)

Radityo Prasetianto.W, S.Kom, M.Kom (Penguji 1)

Faizal Johan Atletiko, S.Kom, M.T (Penguji 2)

PREDIKSI KETERHUBUNGAN ANTAR PRODUK MAKANAN DENGAN GRAPH EMBEDDING MENGGUNAKAN ALGORITMA GRAPH EMBEDDING

Nama : Irfan Rifqi Susetyo
NRP : 05211740000064
Departemen : Sistem Informasi ITS
Pembimbing I : Nur Aini Rakhmawati S.Kom., M.Sc.Eng., Ph.D.

ABSTRAK

Pada tahun 2020, agama Islam tercatat sebagai agama terbesar kedua di dunia dengan jumlah populasi sebanyak 1,9 miliar orang. Di Indonesia sendiri, populasi muslim diperkirakan berjumlah hingga 263 juta orang atau 87,2% dari total penduduk Indonesia. Dengan semakin banyaknya jumlah muslim yang ada, maka kebutuhan akan produk dan makanan halal yang merupakan salah satu ajaran dari agama Islam akan semakin banyak dibutuhkan. Dengan banyaknya organisasi penyeretfikasi halal yang ada, maka sumber informasi terkait daftar produk makanan menjadi terpisah-pisah dan tidak terintegrasi dengan baik. Dampaknya adalah sulit untuk mencari makanan yang sudah tersertifikasi halal, khususnya ketika mengunjungi negara baru yang minim akan produk makanan dengan sertifikasi halal. Meskipun sudah terdapat penelitian terkait integrasi data makanan halal, masih terdapat tantangan yaitu terkait dengan tingkat redundancy data yang tinggi. Selain itu, belum tersedianya informasi mengenai hubungan antar produk makanan. Salah satu cara yang dapat dilakukan untuk mengatasi tantangan tersebut adalah dengan melakukan link prediction. Link prediction merupakan prediksi adanya keterhubungan antara dua entitas atau node pada sebuah jaringan. Di dalam tugas akhir ini akan dilakukan link prediction dengan menggunakan data produk makanan yang diperoleh dari LODHalal, KlikIndomaret, dan HalalMUI. Data

tersebut kemudian diubah ke bentuk vektor dengan menggunakan algoritma graph embedding yaitu Node2Vec dan RDF2Vec. Hasil dari graph embedding tersebut akan menjadi fitur untuk link prediction yang menggunakan Algoritma Random Forest. Diperoleh model link prediction Random Forest yang menggunakan vektor embedding Node2Vec memiliki performa yang lebih baik dengan nilai f1-score 82.7% dan accuracy 83.4%, sedangkan model link Prediction Random Forest yang menggunakan vektor embedding RDF2Vec memiliki performa yang sedikit lebih kecil dengan nilai f1-score 79.7% dan accuracy 80%. Didapatkan juga model link prediction Random Forest dengan data produk makanan halal mendapatkatkan performa yang lebih baik dibandingkan model dengan algoritma Gaussian Naïve Bayes, Logistic Regression, dan Gradient Boosting Classifier.

Kata Kunci: linked open data, link prediction, graph embedding, random forest, node2vec, rdf2vec

LINK PREDICTION BETWEEN FOOD PRODUCTS WITH GRAPH EMBEDDING USING RANDOM FOREST ALGORITHM

Nama : Irfan Rifqi Susetyo
NRP : 05211740000064
Departemen : Sistem Informasi ITS
Pembimbing I : Nur Aini Rakhmawati S.Kom., M.Sc.Eng., Ph.D.

ABSTRACT

In 2020, Islam was recorded as the second biggest religion in the world with 1,9 billion followers. In Indonesia, the moslem population reaches 263 million people, or 87,2% of the total population. As Islam growing, the need for a halal food products is increasing. Because of the many organizations that created halal certificates in their own country, the source of information regarding halal food products can't be integrated. The result is obtaining information regarding halal food products is getting harder. To solve this problem, in the previous research, created Linked Open Data Halal (LODHalal) to collect data from a lot of sources and try to integrate those data. But, there is still a lot of redundancy happening and there is still no information about the relationship between food products. So to predict the relationship, this research will create a link prediction for food products. Link prediction is predicting any link between two entities or nodes in a network. Link prediction will be created with data from LODHalal, KlikIndomaret, and HalalMUI. Data will be transformed to vector using graph embedding algorithms which are Node2Vec and RDF2Vec. Those vectors will be used as features for link prediction using Random Forest algorithm. The result is random forest model with Node2Vec has a better performance with 82.7% f1-score and 83.4% accuracy, mmeanwhile Random Forest model with RDF2Vec has a slightly lower performance with 79.7% f1-score and 80% accuracy. The link prediction model with random forest also

resulted as the best performance with halal food product data compared to Gaussian Naïve Bayes, Logistic Regression, and Gradient Boosting Classifier.

Keywords: linked open data, link prediction, graph embedding, random forest, node2vec, rdf2ve

SURAT PERNYATAAN BEBAS PLAGIARISME

Saya yang bertandatangan di bawah ini:

Nama	: Irfan Rifqi Susetyo
NRP	: 05211740000064
Tempat/Tanggal lahir	: Jakarta / 23 Juni 1999
Fakultas/Departemen	: FTEIC / Sistem Informasi
Nomor Telp/Hp/email	: 082113365187/irfnrifqi@gmail.com

Dengan ini menyatakan dengan sesungguhnya bahwa penelitian/makalah/tugas akhir saya yang berjudul

PREDIKSI KETERHUBUNGAN ANTAR PRODUK MAKANAN DENGAN GRAPH EMBEDDING MENGGUNAKAN ALGORITMA RANDOM FOREST

Bebas Dari Plagiarisme Dan Bukan Hasil Karya Orang Lain.

Apabila dikemudian hari ditemukan seluruh atau sebagian penelitian/makalah/tugas akhir tersebut terdapat indikasi plagiarisme, maka saya bersedia menerima sanksi sesuai peraturan dan ketentuan yang berlaku.

Demikian surat pernyataan ini saya buat dengan sesungguhnya dan untuk dipergunakan sebagaimana mestinya.

Surabaya, 30 Juli 2021

Irfan Rifqi Susetyo
NRP.05211740000064

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah swt. atas izin-Nya penulis dapat menyelesaikan Tugas Akhir dengan judul "**Prediksi Keterhubungan Antar Produk Makanan Dengan Graph Embedding Menggunakan Algoritma Random Forest**" yang menjadi syarat kelulusan di Departemen Sistem Informasi Fakultas Teknologi Elektro dan Infromatika Cerdas Institut Teknologi Sepuluh Nopember Surabaya.

Pengerjaan Tugas Akhir ini dapat dikerjakan oleh karena dukungan, saran, motivasi, dan semangat yang diberikan oleh berbagai pihak. Secara khusus penulis akan menyampaikan ucapan terima kasih banyak kepada :

1. Seluruh keluarga besar penulis terutama orang tua dan kedua saudara penulis yang senantiasa mendoakan, memberikan motivasi dan semangat, sehingga penulis mampu menyelesaikan pendidikan sarjana ini dengan baik.
2. Ibu Nur Aini Rakhmawati, S.Kom., M.Sc.Eng.m Ph.D., dan Ibu Dr. Rarasmaya Indraswari, S.Kom selaku dosen pembimbing yang telah memberikan arahan dan bimbingan, serta memberikan motivasi dalam pengerjaan Tugas Akhir ini.
3. Bapak Faizal Johan Atletiko, S.Kom, M.T. dan Bapak Radityo Prasetianto Wibowo, S.Kom, M.Kom. selaku dosen penguji yang telah memberikan kritik dan saran dalam penyempurnaan Tugas Akhir ini.
4. Ibu Anisah Herdiyanti, S.Kom., M.Sc. yang telah banyak meluangkan waktu untuk membimbing, mengarahkan, dan mendukung dengan memberikan ilmu, petunjuk, dan motivasi dalam penyelesaian Tugas Akhir ini.
5. Vaya Annisa yang selalu menjadi main support system penulis dalam pengerjaan tugas akhir

6. Teman-teman Mekgyver, Bingcit, MGMT, dan Yuarnek yang telah memberikan motivasi dan support selama perkuliahan dan kehidupan
7. Teman-teman laboratorium ADDI dan terkhususnya Girraz yang telah memberikan dukungan dan menjadi teman diskusi selama pengerjaan Tugas Akhir ini.
8. Teman-teman Sistem Informasi angkatan 2017 (Dhistakarna) yang senantiasa menemani dan memberikan motivasi bagi penulis selama perkuliahan hingga dapat menyelesaikan Tugas Akhir ini.
9. Seluruh pihak-pihak lainnya yang tidak dapat disebutkan satu per satu, yang telah membantu penulis selama perkuliahan hingga dapat menyelesaikan tugas akhir ini.

Penyusunan Tugas Akhir ini masih memiliki banyak kekurangan dan belum sempurna, sehingga penulis menerima adanya kritik maupun saran yang membangun untuk perbaikan di masa yang akan datang. Semoga buku Tugas Akhir ini dapat memberikan manfaat bagi pembaca.

Surabaya, 6 Juli 2021
Irfan Rifqi S.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB I
PENDAHULUAN

Pada bab pendahuluan ini akan membahas terkait latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, dan relevansi terhadap pengerjaan tugas akhir.

1.1 Latar Belakang Masalah

Pada tahun 2020, agama Islam tercatat sebagai agama terbesar kedua di dunia dengan jumlah populasi sebanyak 1,9 miliar orang. Di Indonesia sendiri, populasi muslim diperkirakan berjumlah hingga 263 juta orang atau 87,2% dari total keseluruhan penduduk Indonesia [1]. Dengan semakin banyaknya jumlah muslim yang ada, maka kebutuhan akan produk dan makanan halal yang merupakan salah satu ajaran dari agama Islam akan semakin banyak dibutuhkan [2].

Untuk memfasilitasi kebutuhan produk dan makanan halal tersebut, didirikanlah *World Halal Food Council* (WHFC) untuk menghimpun organisasi-organisasi yang bergerak di bidang halal agar lebih mengenal satu sama lain [3]. Misi dari WHFC adalah untuk meregulasi standar halal dalam keseluruhan proses pembuatan produk khususnya makanan. Saat ini sudah terdapat 55 organisasi dari berbagai negara yang tergabung ke dalam WHFC [3].

Organisasi-organisasi yang tergabung pada WHFC umumnya mengeluarkan sertifikasi halal kepada makanan yang beredar di negaranya masing-masing. Dengan banyaknya organisasi yang ada, maka sumber informasi terkait daftar produk makanan menjadi terpisah-pisah dan tidak terintegrasi dengan baik. Dampaknya adalah sulit untuk mencari makanan yang sudah tersertifikasi halal, khususnya ketika mengunjungi negara baru yang minim akan produk makanan dengan sertifikasi halal. Untuk menyelesaikan masalah tersebut, pada penelitian sebelumnya dibuatlah *Linked Open Data Halal* (LODHalal) yang bertujuan untuk mengumpulkan data dari berbagai macam sumber dan mengintegrasikan data tersebut [4].

Meskipun sudah terdapat penelitian terkait integrasi data makanan halal, masih terdapat tantangan yaitu terkait dengan tingkat *redundancy* data yang tinggi. Hal ini dikarenakan data diambil dari sumber yang sangat banyak, sehingga memungkinkan produk makanan yang sama memiliki keterhubungan dengan komposisi, manufaktur, dan status kehalalan yang berbeda. Selain itu, belum tersedianya informasi mengenai hubungan antar produk makanan. Maka dari itu, salah satu cara yang dapat dilakukan untuk mengatasi tantangan tersebut adalah dengan melakukan *link prediction*.

Link prediction merupakan prediksi adanya keterhubungan antara dua entitas atau *node* pada sebuah jaringan [5]. Dalam kasus produk makanan halal, *link prediction* dapat digunakan untuk memprediksi adanya keterhubungan antara suatu makanan dengan makanan lainnya. Dampaknya adalah tingkat *redundancy* dapat dikurangi. Selain itu, penggunaan *link prediction* juga sesuai dengan prinsip *linked data* yang keempat, yaitu menghubungkan data dengan *Uniform Resource Identifie*r (URI) data lainnya sehingga pengguna dapat mendapatkan informasi lebih banyak [12].

Di dalam tugas akhir ini akan dilakukan *link prediction* dengan menggunakan data yang diperoleh dari LODHalal dan KlikIndomaret. Data tersebut kemudian diubah ke bentuk vektor dengan menggunakan algoritma *graph embedding* yaitu Node2Vec dan RDF2Vec. Hasil dari *graph embedding* tersebut akan menjadi *input* untuk *link prediction* yang menggunakan Algoritma *Random Forest*.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka berikut ini merupakan rumusan masalah yang akan diselesaikan pada penelitian ini adalah:

1. Bagaimana menemukan keterhubungan antar produk makanan berdasarkan komposisi, manufaktur, dan lembaga penyertifikasi halal dengan menggunakan hasil *graph embedding*?

2. Bagaimana penggunaan algoritma *graph embedding* yang berbeda mempengaruhi performa dari *link prediction*?

1.3 Batasan Masalah

Batasan permasalahan pada tugas akhir ini yaitu:

1. Data yang diambil merupakan *linked open data* yang berasal dari LODHalal, dan data hasil scraping dari KlikIndomaret dan HalalMUI.
2. Data yang dikumpulkan berupa data produk makanan beserta komposisi, manufaktur, dan lembaga penyertifikasi halal produk makanan tersebut.

1.4 Tujuan

Berdasarkan rumusan masalah yang telah dijelaskan sebelumnya, tujuan dari tugas akhir ini adalah:

1. Mengimplementasikan *link prediction* dengan menggunakan hasil *graph embbeding* untuk menemukan keterhubungan antar produk makanan.
2. Mengevaluasi kinerja algoritma *graph embedding* yang digunakan dalam melakukan *link prediction*.

1.5 Manfaat

Dari pengerjaan tugas akhir ini, adapun manfaat yang dapat diberikan antara lain:

1. Bagi pengguna, dapat mendapatkan informasi terkait rekomendasi makanan yang saling berhubungan.
2. Bagi peneliti, dapat mengembangkan pengetahuan dalam melakukan *link prediction* dengan hasil menggunakan *graph embedding*.

1.6 Relevansi

Tugas akhir ini mengangkat topik mengenai *knowledge graph* di bidang halal, dimana topik ini memiliki kontribusi pada Laboratorium Akuisisi Data dan Diseminasi Informasi (ADDI) Departemen Sistem Informasi Institut Teknologi Sepuluh Nopember Surabaya. Tugas akhir ini memiliki relevansi dengan mata kuliah yang dipelajari di Departemen Sistem Informasi,

Institut Teknologi Sepuluh Nopember, yaitu mata kuliah Teknologi Web terkait semantik web dan *linked data*, serta mata kuliah Penambangan Data dan Analitika Bisnis terkait penggunaan *machine learning* untuk melakukan pekerjaan analitik data.

Tugas akhir ini layak dijadikan sebagai tugas akhir pada tingkat S1, karena tugas ini memecahkan masalah yaitu dalam menemukan keterhubungan antar produk makanan yang ada di dalam LODHalal.

BAB II
TINJAUAN PUSTAKA

Pada bab ini dijelaskan mengenai teori-teori terkait yang bersumber dari buku, jurnal, ataupun artikel yang berfungsi sebagai dasar dalam melakukan pengerjaan tugas akhir agar dapat memahami konsep atau teori penyelesaian permasalahan yang ada.

2.1 Studi Literatur

Penelitian "Drug-Drug Interaction Prediction Based on Knowledge Graph Embeddings and Convolutional-LSTM Network" oleh Md. Rezaul Karim, Michael Cochez, Joao Bosco Jares, Mamtaz Uddin, Oya Beyan, dan Stefan Decker [9]. Peneliti mencoba untuk melakukan prediksi *drug-drug interactions* (DDI) yang meminimalisir adanya kesalahan medis dalam penggunaan obat-obatan. Prediksi DDI dilakukan menggunakan pendekatan *machine learning* dengan menggunakan beberapa sumber data. Data yang digunakan sebanyak 12.000 fitur obat yang berasal dari DrugBank, PharmGKB, dan KEGGdrugs yang diintegrasikan menggunakan *knowledge graphs*. Sebelum melakukan pelatihan untuk model prediksi, dilakukan *embedding* terlebih dahulu untuk *node-node* dengan berbagai macam teknik *graph embedding*. Berdasarkan penelitian tersebut, ditemukan bahwa model kedua terbaik secara keselutuhan dan model machine learning terbaik diperoleh oleh metode embedding ComplEx menggunakan Random Forest. Rata-rata performa model dari 3 klasifikasi terbaik adalah 0.94, 0.92, dan 0.80 untuk masing-masing *AUPR*, *F1-score*, dan *MCC* dengan menggunakan pengujian *5-fold cross-validation*.

Penelitian "Neural Networks For Link Prediction in Realistic Biomedical Graphs: a Multi-dimensional Evaluation of Graph Embedding-based Approaches" oleh Gamal Crichton, Yufan Guo, Sampo Pyysalo, dan Anna Korhonen [10]. Peneliti mencoba mengetahui bagaimana hasil dari beberapa macam pendekatan *graph embedding* mempengaruhi performa dari *neural link predictor* dengan menggunakan *biomedical graph*

yang memiliki informasi relevan berkaitan dengan *Drug-Target Interactions* (DTI), *Protein-Protein Interaction* (PPI), dan *Literature Based Discovery* (LBD). Berdasarkan penelitian tersebut, ditemukan bahwa ketika data dan jumlah *nodes* yang tidak terhubung jumlahnya cukup untuk menggunakan metode *neural networks*, pendekatan *neural networks* memiliki performa yang lebih baik dibandingkan pendekatan *baseline*. Pada level *recall* rendah, kedua pendekatan memiliki performa yang sebanding, namun ketika level *recall* yang lebih tinggi, pendekatan *neural networks* terbukti lebih baik.

Penelitian "Perangkingan Entitas Linked Open Data Menggunakan Pendekatan Resolusi Entitas Untuk Peningkatan Relevansi" oleh Ahmad Choirun [13]. Peneliti mencoba melakukan pendekatan *Entity Resolution* dan *Entity Ranking* untuk meningkatkan relevansi pencarian pada *Halal Nutrition Food*. Pendekatan *Entity Resolution* dilakukan dengan menggunakan *graph embedding Node2Vec* dan *DeepWalk* untuk mengubah setiap *node* menjadi luara berupa vektor representatif. Luaran ini digunakan sebagai kandidat untuk melakukan resolusi dan prediksi link. Untuk proses perangkingan dilakukan dengan pendekatan *query-independent* (QI) dan *query-dependent* (QD). Kombinasi perhitungan QI dan QD akan menghasilkan skor final yang akan dijadikan bobot acuan dalam perangkingan entitas untuk meningkatkan relevansi pencarian. Hasil penelitian ini menunjukkan bahwa faktor yang paling dominan yaitu nilai skor dependen, kemudian diikuti oleh skor independent. Semakin besar nilai skor dependen maka semakin besar peluang muncul sebuah dokumen untuk ditampilkan pada sistem dan menjadi hasil pencarian yang relevan. Semakin besar nilai skor independent akan mempengaruhi peringkat ranking dari dokumen pada hasil pencarian. Performa pendekatan *Node2Vec* menunjukkan hasil yang lebih baik dibandingkan dengan *DeepWalk*, hal ini dikarenakan adanya

kualitas *embedding* dari *Node2Vec* yang lebih baik daripada *DeepWalk.*

Untuk lebih jelasnya terkait penelitian sebelumnya yang telah dilakukan dapat dilihat pada Tabel 2.1

Tabel 2.1 Penelitian Sebelumnya

No	**Penelitian Terdahulu**	
1	Nama Peneliti	Md. Rezaul Karim, Michael Cochez, Joao Bosco Jares, Mamtaz Uddin, Oya Beyan, dan Stefan Decker
	Tahun Penelitian	2019
	Judul Penelitian	Drug-Drug Interaction Prediction based on Knowledge Graph Embeddings and Convolutional-LSTM Network
	Isi Penelitian	Melakukan prediksi *drug-drug interaction* (DDI) dengan cara melakukan *graph embedding* dan melakukan prediksi menggunakan model *machine learning*.
	Keterkaitan dengan Tugas Akhir	Melakukan *graph embedding* untuk melakukan link prediction dengan menggunakan *machine learning.* Untuk graph embedding, salah satu algoritma yang digunakan adalah *RDF2Vec*.
2	Nama Peneliti	Gamal Crichton, Yufan Guo, Sampo Pyysalo, dan Anna Korhonen
	Tahun Penelitian	2018
	Judul Penelitian	Neural Networks for Link Prediction in Realistic Biomedical Graphs: A Multi-dimensional Evaluation of Graph Embedding-based Approaches

	Isi Penelitian	Mengetahui performa dari pendekatan *neural networks* dengan proses *graph embedding* menggunakan *biomedical graph* dibandingkan dengan pendekatan *baseline*.
	Keterkaitan dengan Tugas Akhir	Melakukan *graph embedding* untuk melakukan link prediction dengan menggunakan *machine learning.*
3	Nama Peneliti	Ahmad Choirun Najib, Nur Aini Rakhmawati
	Tahun Penelitian	2020
	Judul Penelitian	Perangkingan Entitas Linked Open Data Menggunakan Pendekatan Resolusi Entitas Untuk Peningkatan Relevansi
	Deskripsi Umum Penelitian	Melakukan perangkingan entitas dengan melakukan pendekatan *Entity Resolution* dan *Entity Ranking* guna meningkatkan relevansi pencarian pada Halal Nutrition Food.
	Keterkaitan dengan Tugas Akhir	Entity Resolution dilakukan dengan melakukan graph embedding yang luarannya digunakan untuk prediksi link. Untuk graph embedding, salah satu algoritma yang digunakan adalah *Node2Vec*.

2.2 Dasar Teori

Pada sub bab ini akan dibahas mengenai dasar teori yang berhubungan dengan tugas akhir ini.

2.2.1 Linked Data

Secara umum, yang dimaksud dengan *linked data* adalah menggunakan web untuk membuat hubungan antar data dari berbagai macam sumber. *Linked data* merujuk pada data yang

dipublikasikan dalam web dengan cara yang dapat dipahami oleh mesin atau *machine-readable*, yang artinya data tersebut secara eksplisit dijelaskan dan terhubung dengan data-data lainnya.

Tim Berners-Lee telah mengemukakan beberapa ketentuan terkait dengan publikasi data pada web agar data tersebut bisa menjadi bagian dari *single global data space* yang kemudian disebut sebagai prinsip-prinsip *linked data* [14], yaitu:

1. Menggunakan URI sebagai penamaan untuk berbagai hal
2. Menggunakan HTTP URI agar orang-orang dapat mencari hal tersebut
3. Ketika seseorang mencari URI, tersedia informasi yang berguna menggunakan standar seperti RDF maupun SPARQL

2.2.2 Linked Open Data Halal

Linked Open Data Halal (LODHalal) adalah sistem *linked open data* berbasis web dan android yang digunakan untuk mengambil berbagai data terkait dengan produk makanan dari berbagai macam sumber dan mengintegrasikan data tersebut. Data-data yang diperoleh diubah ke bentuk RDF dan dihubungkan dengan beberapa *dictionary* seperti DBpedia, PubChem, dan Mesh. LODHalal ini dibuat untuk menyediakan rekomendasi terkait kehalalan dan status nutrisi dari suatu produk makanan.

Proses pengembangan sendiri mencakup mengusulkan kosakata makanan halal yang dikembangkan dari kosakata dua makanan yang sudah ada, mentransformasikan sekumpulan data makanan halal menjadi sebuah *linked data*, mengintegrasikan data ke *linked open data* yang lain, dan pembuatan aplikasi LODHalal itu sendiri. Akses data yang diberikan pada sistem ini beragam, seperti *SPARQL endpoint*, *RDF dump*, dan *web service* [4].

2.2.3 Resource Description Framework

Resource Description Framework (RDF) merupakan kerangka kerja untuk menerangkan informasi tentang sumber-sumber data. Sumber data bisa berbentuk apapun, termasuk dokumen, orang, objek fisik, dan konsep abstrak. RDF dimaksudkan untuk situasi dimana informasi pada web butuh untuk diproses oleh aplikasi daripada hanya ditampilkan saja untuk penggunanya. RDF menyediakan kerangka kerja umum untuk menerangkan informasi sehingga bisa dilakukan pertukaran data antar aplikasi tanpa kehilangan arti. Memiliki kemampuan untuk bertukar informasi antar aplikasi berarti informasi dapat dibuat tersedia di aplikasi lainnya tanpa membuat informasi tersebut sebelumnya [15].

Salah satu RDF data model yang umum digunakan dinamakan *triples*. Model ini memudahkan pengguna untuk membuat pernyataan tentang *resources* yang ada. Untuk bentuk *triples* lebih jelas dapat dilihat ilustrasi di bawah ini:

Irfan membaca buku.

<subyek> <predikat><obyek>

Sebuah *RDF statement* menjelaskan tentang hubungan antar 2 buah *resource*. Subyek dan obyek menunjukkan *resource* yang dirujuk, sedangkan predikat menunjukkan hubungannya dari kedua *resource* tersebut. Hubungan yang ada dituliskan dari subyek ke obyek, dan statement inilah yang didalam RDF disebut sebagai *property*. Karena *RDF statement* mengandung tiga buah elemen, maka dari itu data model ini disebut sebagai *triples* [15].

2.2.4 Praproses Data

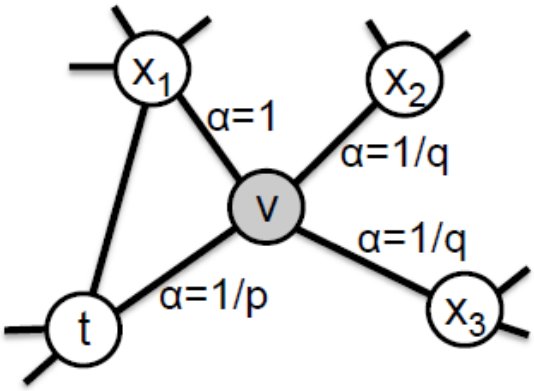
Graph embedding merupakan pendekatan untuk mengkonversi data dari *graph* ke dalam bentuk *low-dimensional* dimana informasi *structural graph* dan properti dari *graph* disimpan dengan maksimal [16]. *Embedding* yang dilakukan dapat menghasilkan *graph topology*, hubungan antar *vertex*, dan informasi lainnya terkait *graphs*, *subgraphs*, dan *vertices*.

Semakin banyak properti yang *embedder* hasilkan, maka semakin baik pula hasil yang dapat dihasilkan. Terdapat beberapa algoritma untuk melakukan *graph embedding*, yaitu:

a. Node2Vec

Node2Vec merupakan *framework* algoritma *semi-supervised* yang digunakan untuk merepresentasikan pembelajaran berkelanjutan dari *nodes* pada sebuah jaringan. Model ini dapat memetakan *nodes* yang ada ke dalam bentuk *low-dimensional* untuk memaksimalkan probabilitas *node* yang berdekatan dalam sebuah jaringan [17]. Algoritma Node2Vec mempelajari secara berkelanjutan representasi untuk *nodes* pada *undirected* dan *unweighted graph*. Hasil representasi dari *nodes* pada *graph* nantinya dapat digunakan untuk berbagai pekerjaan *machine learning* [17].

Node2Vec memiliki *random walks* yang dibuat sebagai parameter untuk menyediakan *trade-off* antara memprioritaskan *breadth-first walk* (BFS) atau *depth-first walk* (DFS). Memilih parameter yang tepat dapat menghasilkan *walks* yang lebih informatif, yang mengarah ke *embeddings* yang superior.



Gambar 2.1 Ilustrasi random walks pada Node2Vec

Pada Gambar 2.1 diilustrasikan cara kerja *random walks* dari algoritma Node2Vec. Untuk mengatur eksekusi *random walks* tersebut, terdapat dua parameter yang digunakan, yaitu:

1. **Return parameter**, p. Parameter p mengatur kemungkinan mengunjungi *node* yang sama ketika *walk*. Membuat nilai p semakin besar (>max(q,1)) akan memastikan semakin kecil kemungkinan untuk mengunjungi *node* yang sama dalam dua langkah ke depan (kecuali *node* selanjutnya tidak memiliki *neighbor* lainnya). Strategi ini membuat eksplorasi yang cukup dan menghindari *2-hop redundancy* dalam *sampling*. Sebaliknya, jika nilai p kecil (<min(q,1)), akan membuat walk melakukan *backtrack* dan hal ini membuat walk tidak akan jauh dari *node* v.
2. **In-out parameter**, q. Parameter q memungkinkan pencarian untuk membedakan antara *node inward* dan *outward*. Jika q > 1, *random walk* akan bias terhadap *nodes* yang dekat dengan *node* t. Sebaliknya, jika q < 1, walk akan lebih memungkinkan untuk mengunjungi *node* yang jauh dari *node* 2.

Singkatnya, parameter menentukan seberapa sering *random walk* melakukan *backtrack*. *In-out parameter* mengontrol jika *random walk* lebih fokus pada eksplorasi lokal seperti *Breadth First Search* (BFS) atau lebih condong pada eksplorasi luar seperti *Depth First Search* [17].

b. RDF2Vec

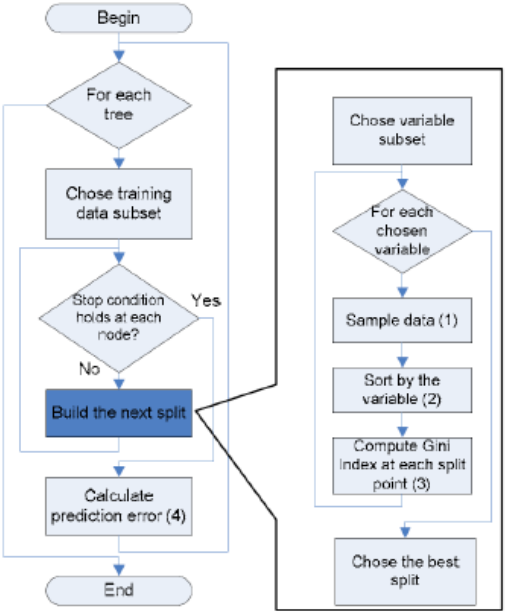
RDF2Vec merupakan sebuah *tool* untuk representasi vektor dari sebuah *RDF graph*. Secara singkat, *RDF2Vec* membuat vektor numerik untuk tiap *node* yang ada pada

RDF graph [18]. RDF2Vec terinspirasi dari pendekatan *Word2Vec* dalam melakukan representasi kata ke dalam bentuk vektor numerik. *Word2Vec* menggunakan kumpulan kata sebagai *input* dan melatih *neural networks* untuk memprediksi kata berdasarkan konteks kata yang dimiliki atau memprediksi konteks kata dari kata yang dimiliki.

Dalam awal penelitiannya, random walks pada *RDF graph* digunakan untuk membuat sekuensial dari *RDF nodes* yang mana digunakan sebagai input untuk algoritma *Word2Vec*. Representasi yang dihasilkan oleh *Word2Vec* sudah diaplikasikan dibeberapa hal seperti dalam penggunaan *knowledge graph* sebagai *background knowledge* dalam kegiatan penambangan data, atau untuk membangun *content-base recommender systems* [18].

2.2.5 Random Forest

Algoritma *Random Forest* (RF) merupakan salah satu metode berbasis klasifikasi dan regresi dari *machine learning* dimana terdapat proses agregasi pohon keputusan. *Machine learning* sendiri merupakan suatu pendekatan dalam *Artifical Intelligence* dengan menirukan perilaku manusia dalam menyelesaikan masalah dan juga mengotomasinya. Tujuan utamanya adalah membuat komputer dapat belajar secara otomatis tanpa intervensi manusia [19]. Spesifiknya, RF merupakan kumpulan dari *trees* yang dibangun dari dataset training dan divalidasi secara internal untuk menghasilkan prediksi dari respon yang diberikan prediktor untuk pengamatan di masa depan [20]. Langkah kerja dari algoritma RF dapat dilihat pada Gambar 2.2.



Gambar 2.2 Alur kerja algoritma Random Forest

Pada Gambar 2.2 dijelaskan alur kerja dari algoritma *Random Forest*, yaitu:

1. Dimisalkan N adalah jumlah data latih dan M adalah jumlah fitur serta m adalah jumlah fitur yang akan digunakan untuk menentukan keputusan pemilahan pada sebuah *node*. Dimana m nilainya lebih kecil dari M.
2. Pilih satu *set training* acak dengan memilih N kali dengan pengganti dari semua data N, ini disebut juga *sample bootstrap*, hanya 2/3 dari data asli yang digunakan. Sisa dataset digunakan untuk *testing* tujuan yang disebut *Out-of-Bag* yang digunakan untuk memperkirakan *error OOB* dalam melakukan klasifikasi.
3. Untuk setiap *node* dari *tree*, *node* dipisah dengan fitur terbaik di antar fitur yang terpilih secara acak menggunakan

kriteria pemisah *impurity*, lalu hitung pemisahan terbaik berdasarkan m fitur dalam data latih.

4. Setiap *tree* tumbuh penuh dengan metode *Decision Tree* tanpa adanya pemangkasan. Untuk prediksi, sampel yang baru, dimasukkan ke *node*, lalu diberi label sampel *training* di *node* tempat sebuah sampel baru tersebut berada. Prosedur ini diulangi untuk semua *tree* di dalam kumpulan *forest*, lalu hasil prediksi semua *tree* diaplikasikan *majority vote* untuk hasil prediksi klasifikasi [21][22].

Algoritma RF digunakan karena memiliki performa yang baik dibandingkan algoritma ML lainnya dalam melakukan *link prediction*, bahkan dalam kasus tertentu dapat melampaui performa dari *Deep Learning* [9].

2.2.6 Link Prediction

Dalam teori jaringan, *link prediction* merupakan prediksi adanya keterhubungan antara dua entitas atau *nodes* pada sebuah jaringan. Penggunaan *link prediction* sudah digunakan di berbagai bidang seperti sosial media untuk memberikan rekomendasi pertemanan dan konten [5][6], keamanan jaringan untuk mengidentifikasi *card fraud* [7][8], *biology information science* untuk memprediksi keterhubungan antar protein [9], *e-commerce* untuk memberikan rekomendasi barang ataupun film [23], hingga mengidentifikasi grup teroris atau kriminal yang tersembunyi berdasarkan aktivitasnya [11][24]. Penerapan *link prediction* juga sesuai dengan prinsip *linked data* yang keempat, yaitu menghubungkan data dengan *Uniform Resource Identifier* (URI) data lainnya sehingga pengguna dapat mendapatkan informasi lebih banyak [12].

2.2.7 Naïve Bayes

Klasifikasi Bayes adalah klasifikasi statistic yang dapat memprediksi keluar suatu anggota probabilitas. Untuk klasifikasi Bayes sederhana lebih dikenal sebagai Naïve Bayes Classifier dapat diasumsikan bahwa efek dari suatu nilai atribut

sebuah kelas yang diberikan adalah bebas dari atribut-atribut lain. Asumsi ini disebut class conditional independence yang dibuat untuk memudahkan perhitungan, pengertian ini dianggap 'Naïve'. Dalam Bahasa sederhana, 'Naïve' mengasumsikan bahwa kemunculan suatu term kata dalam suatu kalimat tidak dipengaruhi kemungkinan-kemungkinan kata yang lain dalam kalimat, walaupun dalam kenyataan bahwa kemungkinan kata dalam kalimat sangat dipengaruhi keberadaan kata-kata yang ada dalam kalimat [28].

Algoritma Bayes mampu memberikan perspektif yang unik dalam memahami banyak algoritma yang secara lansgung menggunakan probabilitas . Pendekatan didasarkan pada teori Bayes yang dinyatakan pada persamaan (2.1)

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \qquad (2.1)$$

dimana,

$P(Y\|X)$	Adalah probabilitas kategori Y berdasarkan kondisi X (posterior probability)
$P(X\|Y)$	Adalah probabilitas X dengan kemungkinan Y (likelihood)
$P(Y)$	Adalah nilai probabilitas pada data training melalui perhitungan kategori Y (prior probability)
$P(X)$	Adalah total probabilitas untuk semua data training X

Naïve Bayes Classifier (NBC) atau Multinomial Naivee Bayes merupakan model penyerdahanaan dari algoritma Bayes yang cocok dalam pengklasifikasian teks atau dokumen. Algoritma Naïve Bayes Classifier merupakan algoritma yang digunakan untuk mencari nilai probabilitas tertinggi untuk mengklasifikasi data uji pada kategori yang paling tepat [29]. Kelebihan NBC

adalah algoritmanya sederhana, tetapi memiliki akurasi yang tinggi.

2.2.8 Logistic Regression

Metode regresi merupakan analisis data yang digunakan untuk mencari hubungan antara variabel respon (y) dengan satu atau lebih variabel predictor (x) [30]. Tujuan dari metode ini adalah memperoleh model yang baik dan sederhana yang menggambarkan variabel respon dengan sekumpulan variabel predictor. Regresi logistik merupakan suatu analisis regresi yang digunakan untuk menggambarkan hubungan antara variabel respon yang bersifat dikotomus (bereskala nominal atau ordinal dengan lebih dari dua kategori) dengan sekumpulan variabel predictor bersifat kontinu atau kategorik [31].

Persamaan regresi logistik yang digunakan dari bentuk taksiran fungsi peluang $\pi(x) = E(Y|x)$ dinyatakan pada persamaan 2.2 [30].

$$\pi(x) = \frac{e^{\beta 0+\beta 1x1+\beta 2x2+\cdots+\beta nxn}}{1+e^{\beta 0+\beta 1x1+\beta 2x2+\cdots+\beta nxn}} \qquad (2.2)$$

2.2.9 Gradient Boosting Classifier

Meta-algoritma adalah algoritma yang menggunakan algoritma lain sebagai perwakilan, dan juga merupakan algoritma yang memiliki sub-algoritma sebagai peubah dan parameter yang dapat diganti. Contoh-contoh yang termasuk meta-algoritma adalah Boosting. Boosting merupakan meta-algoritma dalam machine learning untuk melakukan supervised learning. Secara umum, boosting terjadi dalam iterasi, secara incremental menambahkan weak learner ke dalam satu strong learner. Pada setiap iterasi, satu weak learner belajar dari suatu data latihan. Kemudian, weak learner itu ditambahkan ke dalam strong learner. Setelah weak learner ditambahkan, data-data kemudian diubah masing-masing bobotnya. Data-data yang mengalami kesalahan klasifikasi akan mengalami penambahan bobot, dan

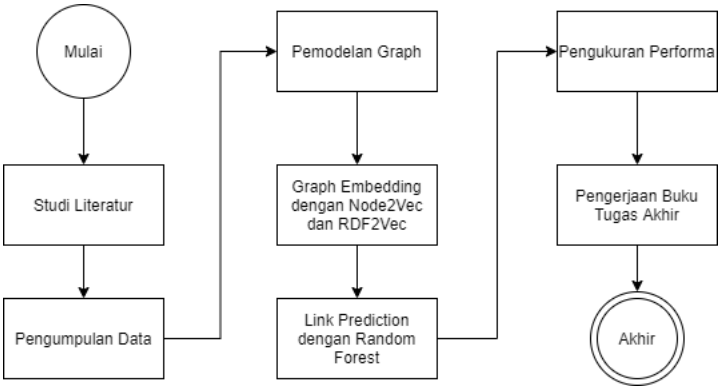
data-data yang terklasifikasi 13 dengan benar akan mengalami pengurangan bobot [32]. Oleh karena itu, weak learner pada iterasi selanjutnya akan lebih terfokus pada data-data yang mengalami kesalahan klasifikasi oleh weak learner yang sebelumnya. Gradient boosting adalah salah satu dari algoritma boosting, dimana mengasilkan model prediksi dari weak learner berbentuk decision tree. Gradient boosting melatih decision tree untuk meminimalkan loss function, Gradient boosting menangan fitur kategoris, perluasan dari multiclass classification setting, tidak memerlukan fitur scaling, dan dapat menangkap fitur nonlinearities dan interaksi [33].

BAB III
METODOLOGI

Pada bab metodologi akan menjelaskan bagaimana langkah pengerjaan tugas akhir dengan disertakan deskripsi dari setiap penjelasan untuk masing-masing tahapan beserta jadwal kegiatan pengerjaan tugas akhir.

3.1 Tahapan Pengerjaan Tugas Akhir

Pada sub bab ini akan dijelaskan terkait metodologi dalam pelaksanaan tugas akhir. Tahapan dari metodologi tersebut dapat dilihat pada Gambar 3.1



Gambar 3.1 Tahapan pengerjaan tugas akhir

3.1.1 Studi Literatur

Pada tahap ini sumber literatur yang mendukung pengerjaan tugas akhir dikumpulkan sebanyak mungkin, sebagai acuan dan dasar melakukan penelitian, dalam hal ini terkait dengan dengan *linked data*, *Node2Vec* dan *RDF2Vec* untuk *graph embedding*, serta *Random Forest* untuk *link prediction*. Hal ini bertujuan untuk memahami konsep, teori, landasan, dan teknologi yang digunakan sebagai acuan pada penelitian ini.

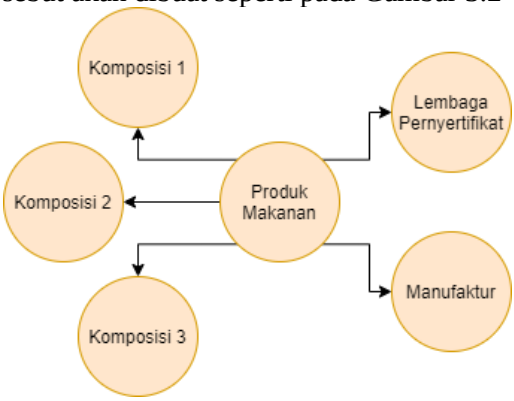
3.1.2 Pengumpulan Data

Pengumpulan data pada penelitian ini berupa pengumpulan data produk-produk makanan. Data produk makanan diambil dari LODHalal dan *website* KlikIndomaret. Pengumpulan data

dilakukan dengan cara melakukan *crawling* dari masing-masing sumber. Untuk data yang berasal dari KlikIndomaret memiliki tipe data html, sehingga perlu diubah ke bentuk RDF dengan melakukan RDFization. Setelah data dari kedua sumber dikumpulkan, selanjutnya dibuat model graph untuk produk makanan yang berisikan *node* produk makanan, komposisi, manufaktur, dan lembaga-lembaga penyertifikasi halal.

3.1.3 Pemodelan Graph

Dari data yang telah diperoleh, dimodelkan menjadi sebuah *graph* yang berisikan produk makanan dan fitur-fiturnya seperti komposisi, manufaktur, serta lembaga penyertifikasi halalnya. Model tersebut akan dibuat seperti pada Gambar 3.2



Gambar 3.2 Model graph produk makanan

3.1.4 Graph Embedding

Pada tahap ini, *graph* yang telah dimodelkan pada tahapan sebelumnya diubah ke bentuk *low dimensional* atau bentuk vektor dengan cara melakukan *graph embedding*. *Graph embedding* dilakukan dengan dua cara yang berbeda, yaitu menggunakan Node2Vec dan RDF2Vec guna menghasilkan hasil *embedding* yang berbeda.

3.1.5 Link Prediction

Pada tahap ini, *node-node* yang sudah diubah dari bentuk *graph* ke bentuk vektor selanjutnya diberikan label berdasarkan hubungan antar produknya. Proses labeling akan dilakukan dengan menggunakan hasil label pada penelitian "Perangkingan Entitas Linked Open Data Menggunakan Pendekatan Resolusi Entitas Untuk Peningkatan Relevansi" untuk hubungan antar *node* yang sudah ada pada penelitian tersebut. Untuk label hubungan *node* produk makanan yang belum terdapat pada penelitian tersebut, maka akan digunakan pelabelan hasil dari jaccard similarity mengggunakan data LODHalal [27]. Pada penelitian tersebut dilakukan analisa *similarity* antar *node* produk makanan dengan menggunakan algoritma *similarity*. Jika sebuah nilai *similarity* dari dua buah *node* produk makanan memiliki nilai tingkat kebenaran di atas 80%, maka nilai similarity tersebut masih masuk ke dalam treshhold dimana akan diberikan label 1 atau kedua *node* saling terhubung.

Selanjutnya dilakukan *link prediction* dilakukan dengan menggunakan salah satu algoritma *machine learning*, yaitu *Random Forest*. Hasil dari *link prediction* adalah menentukan adanya keterhubungan antara satu *node* dengan *node* lainnya berdasarkan hubungan yang dimiliki kedua *node* tersebut. Untuk setiap pasangan *node* produk makanan yang yang memiliki keterhubungan, akan ditambahkan properti *halalf:similarTo* yang merujuk kepada pasangan *node* masing-masing.

3.1.6 Pengukuran Performa

Pada tahap ini, dilakukan pengujian terhadap data yang telah diperoleh dalam proses sebelumnya. Dimensi yang akan diuji dari data adalah sebagai berikut [25][26]:

a. Accuracy
Merupakan rasio prediksi benar (positif dan negatif) dengan keseluruhan data. Untuk rumus mencary nilai *accuracy* dapat dilihat pada persamaan (3.1)

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \qquad (3.1)$$

b. Precision
Merupakan rasio prediksi benar positif dibandingkan dengan keseluruhan hasil yang diprediksi positif. Untuk rumus mencari nilai *precision* dapat dilihat pada persamaan (3.2)

$$Precision = \frac{TP}{TP + FP} \qquad (3.2)$$

c. Recall
Merupakan rasio prediksi benar positif dibandingkan dengan keseluruhan data yang benar positif. Untuk rumus mencari nilai *recall* dapat dilihat pada persamaan (3.3)

$$Recall = \frac{TP}{TP + FN} \qquad (3.3)$$

d. F1-Score
Merupakan perbandingan rata-rata *precision* dan *recall* yang dibobotkan. Untuk rumus mencari nilai *F1-Score* dapat dilihat pada persamaan (3.4)

$$F1\ Score = 2\ x\ \frac{Recall\ x\ Precision}{Recall + Precision} \qquad (3.4)$$

e. ROC Curve
Receiver operating characteristic (ROC) merupakan ilustasi grafik yang menunjukkan performa pengerjaan tugas klasifikasi di tiap nilai *treshhold*-nya. *ROC curve*

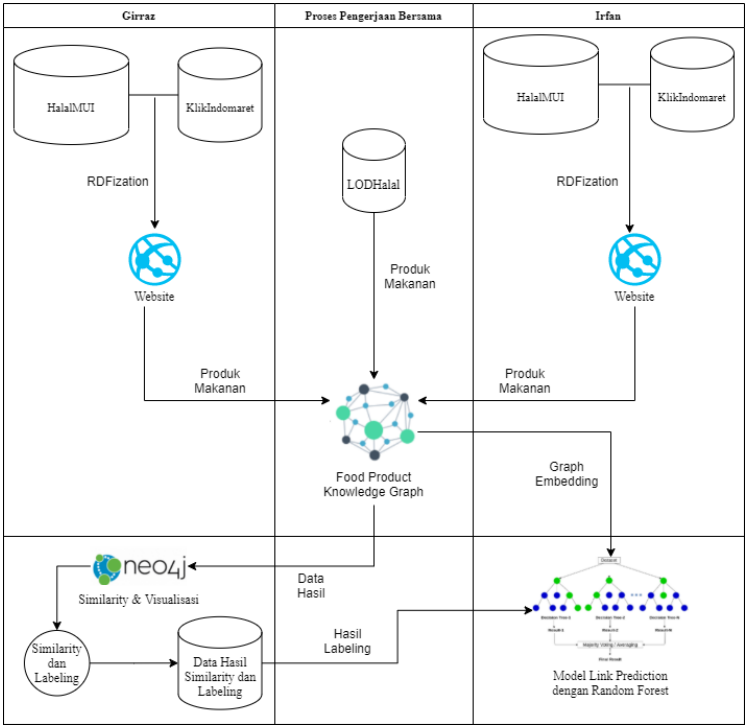
dibuat dengan membandingkan nilai *True Positive Rate* (TPR) dengan *False Positive Rate* (FPR). TPR biasa disebut juga dengan *sensitivity* ataupun recall. Sedangkan FPR atau yang biasa dikenal sebagai *false alarm*, dapat dihitung dengan mengkalkulasi nilai (1 – *specificity*).

3.1.7 Pengerjaan Buku Tugas Akhir

Pada tahap ini, seluruh pengerjaan tugas akhir dari awal hingga selesai didokumentasikan ke dalam sebuah buku tugas akhir. Buku tugas akhir dibuat berdasarkan panduan penyusunan buku tugas akhir yang telah ditentukan.

3.2 Arsitektur Sistem

Arsitektur dari sistem yang akan digunakan dalam pengerjaan tugas akhir ini diilustrasikan pada Gambar 3.3



Gambar 3.3 Gambaran sistem penelitian tugas akhir

Data produk makanan yang akan digunakan untuk tugas akhir ini berasal dari dua sumber data, yaitu LODHalal dan KlikIndomaret. Untuk data yang berasal dari KlikIndomaret memiliki tipe data html, sehingga perlu diubah ke bentuk RDF dengan melakukan *RDFization*. Setelah data dari kedua sumber dikumpulkan, selanjutnya dibuat model *graph* untuk produk makanan yang berisikan *node* produk makanan, komposisi, manufaktur, dan lembaga-lembaga penyertifikasi halal.

Model *graph* yang telah dibuat, selanjutnya diubah ke bentuk *low-dimensional* atau vektor numeriknya dengan melakukan *graph embedding*. Hasil dari *graph embedding* akan dijadikan sebagai masukan untuk melakukan *link prediction* antar produk makanan.

Selanjutnya dibuat model *link prediction* dengan menggunakan dataset training dan selanjutkan divalidasi menggunakan *cross validation*. Selanjutnya model tersebut akan diuji performanya menggunakan dataset test. Dari model tersebut akan menghasilkan prediksi apakah suatu produk makanan memiliki keterhubungan dengan produk makanan lainnya.

BAB IV
PERANCANGAN

Pada bab ini dijelaskan mengenai rancangan penelitian tugas akhir yang merupakan bagaimana proses penelitian mulai dari melakukan pengumpulan data hingga pengujian model *link prediction*.

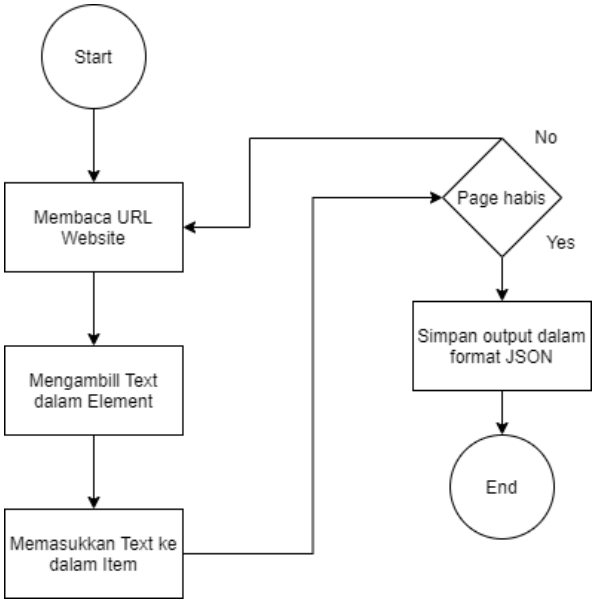
4.1 Pengumpulan Data

Tahap pengumpulan data dilakukan untuk mengumpulkan data produk-produk makanan. Data produk makanan tersebut akan diperoleh melalui dua sumber yaitu *website* KlikIndomart dan *website* Halal MUI. Data produk makanan yang diperoleh dari *website* Halal MUI akan digunakan untuk memberikan informasi yang lebih lengkap pada data produk makanan yang diperoleh dari *website* KlikIndomaret, meliputi nama produsen dan sertifikat halal jika produk makanan tersebut sudah tersertifikasi halal oleh MUI.

4.1.1 Melakukan Web Scraping

Proses *web scraping* akan dilakukan terhadap *website* KlikIndomaret dan *website* Halal MUI. Berikut merupakan tahapan dari *scraping data* yang akan dilakukan:

i. Membuat Kode Scraper
   Kode *scraper* dibuat secara terpisah untuk mendapatkan data dari *website* KlikIndomaret dan *website* Halal MUI dikarenakan perbedaan struktur html yang dimiliki oleh masing-masing *website*. Untuk alur kerja penulisan kode *scraper* dapat dilihat pada Gambar 4.1



Gambar 4.1 Alur kerja kode scraper

Dari Gambar 4.1 dapat dilihat pada alur kerja *scraper* dimana hal yang pertama dilakukan adalah dengan mengambil url data tiap produk makanan. Selanjutnya url tersebut akan dibuka satu per satu untuk mengambil *text* dari *element* html yang diperlukan untuk pengolahan data seperti nama produk, komposisi, produsen, hingga nomor sertifikat. Selanjutnya *text* tersebut akan disimpan pada *item* yang telah disiapkan. Kegiatan ini akan dilakukan berulang kali hingga tidak terdapat halaman web yang tersisa untuk diambil datanya.

ii. Melakukan Web Scraping

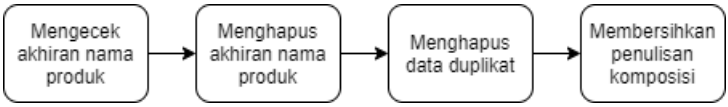
Web scraping dilakukan dengan menjalankan kode *scraper* yang sebelumnya telah dibuat menggunakan *Scrapy*. Kode *scraper* tersebut dibuat dan dijalankan untuk memperoleh data terkait dengan produk makanan yang dibutuhkan dari masing-masing *website*

KlikIndomaret dan *website* Halal MUI. Hasil dari pengumpulan data produk makanan tersebut akan disimpan dalam format JSON.

4.1.2 Membersihkan Data Hasil Scraping

Pembersihan data hasil *scraping* ini dilakukan agar data hasil *scraping* dapat digunakan pada kegiatan selanjutnya. Untuk perlakuan pembersihan dibedakan antara data hasil *scraping* dari website KlikIndomaret dan Halal MUI.

Gambar 4.2 merupakan tahap pembersihan data yang dilakukan pada data hasil *scraping* dari website KlikIndomaret:



Gambar 4.2 Tahap pembersihan data hasil scraping KlikIndomaret

1. Mengecek apakah produk memiliki akhiran berupa takaran ukuran
   Melakukan pengecekan apakah kata terakhirnya berupa ukuran atau takaran. Jika akhirannya bukan merupakan takaran, maka ditambahkan sebuah satu kata yang nantinya akan dihapus. Tahapan penghapusan ini dilakukan guna meningkatkan nilai *similarity* yang dapat dihasilkan ketika proses *matching data* antara data KlikIndomaret dan data HalalMUI. Hal ini dikarenakan pada data HalalMUI, tidak terdapat ukuran atau takaran produk sehingga dengan menghapus ukuran pada data KlikIndomaret dapat mempengaruhi nilai *similarity* yang diperoleh secara signifikan.

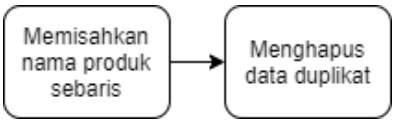
2. Menghapus takaran ukuran di seluruh akhir nama produk
   Menghapus seluruh kata terakhir dari nama produk makanan

3. Menghapus data duplikat

Menghapus nama produk makanan yang terduplikat

4. Melakukan pembersihan manual terhadap bahan-bahan dalam di dalam komposisi
   Melakukan *cleaning* pada data komposisi dengan menyeragamkan komposisi-komposisi yang ada secara manual. Tujuannya adalah agar komposisi yang sama dengan penulisan yang berbeda terminimalisir sehingga *knowledge graph* yang terbentuk semakin baik.

Gambar 4.3 merupakan tahap pembersihan data yang dilakukan data hasil *scraping* dari *website* HalalMUI:



Gambar 4.3 Tahap pembersihan data hasil scraping Halal MUI

1. Memisahkan nama produk sebaris menjadi beberapa kolom
   Dikarenakan hasil *scraping* dari Halal MUI berupa satu *string* panjang yang dipisahkan oleh tanda "|", maka string tersebut dipisahkan sehingga menjadi kolom-kolom yang sesuai.

2. Menghapus data duplikat
   Menghapus nama produk makanan yang terduplikat

4.1.3 Melakukan Matching Data

Setelah data hasil *scraping* dari kedua sumber telah dibersihkan, selanjutnya dilakukan *matching data* antara data yang diperoleh dari *website* KlikIndomaret dan *website* Halal MUI. Tujuan dari langkah *matching data* ini adalah untuk memberikan informasi yang lebih lengkap pada data yang diperoleh dari *website* KlikIndomaret, yaitu dengan menambahkan produsen serta nomor sertifikat halal jika produk makanan tersebut telah tersertifikasi halal oleh MUI.

Matching data dilakukan dengan menggunakan algoritma *Jaccard Similarity* untuk menentukan apakah suatu produk makanan dari *website* KlikIndomaret merupakan produk makanan yang sama yang diperoleh dari *website* Halal MUI. Algoritma *jaccard similarity* digunakan sebagai algoritma untuk *matching data* dikarenakan algoritma *jaccard similarity* merupakan salah satu algoritma yang terbaik dalam melakukan *text similarity* dibandingkan beberapa algoritma lainnya seperti *cosine similarity*. Selanjutnya dilakukan *labeling* untuk memastikan hasil dari *matching data* menggunakan *Jaccard Similarity* menghasilkan kesamaan produk makanan yang tepat.

4.2 Graph Modelling

4.2.1 Mengubah Data Ke Bentuk RDF

Data produk makanan yang telah diproses dan digabungkan pada tahap sebelumnya, selanjutnya diubah ke bentuk RDF dengan format sesuai dengan bentuk RDF dari LODHalal. Tujuannya adalah agar data hasil *scraping* dapat dengan mudah diintegrasikan dengan data dari LODHalal. Pada pembuatan RDF ini juga dilakukan penghapusan tanda baca menggunakan *regex* dan melakukan *lower-casing* agar dalam prosesnya tidak mengalami permasalahan.

Gambar 4.4 merupakan contoh bentuk format RDF yang diharapkan agar sesuai dengan bentuk dari RDF milik LODHalal:

```
halalf:Dua_Kelinci_Snack_Krip_Krip_Tortilla_Sapi_Panggang a
foodlirmm:FoodProduct;
    a food:Food;
    foodlirmm:code "331";
    rdfs:label "dua kelinci snack krip krip tortilla sapi panggang";
    gr:hasManufacturer halalm:Pt_Dua_Kelinci .

halalf:Dua_Kelinci_Snack_Krip_Krip_Tortilla_Sapi_Panggang
foodlirmm:certificate halalc:00100017571001 .

halalf:Dua_Kelinci_Snack_Krip_Krip_Tortilla_Sapi_Panggang
food:containsIngredient
halali:jagung,halali:minyak_sawit,halali:antioksidan_tbhq,halali:pati_jagung,
halali:bumbu_sapi_panggang,halali:gula,halali:garam,halali:sayuran_bubuk,hala
li:maltodekstrin,halali:protein_kedelai_terhidrolisis,halali:rempahrempah,hal
ali:natrium_sitrat,halali:asam_sitrat,halali:antikempal_silikon_dioksida,hala
li:dinatrium_inosinat,halali:dinatrium_guanilat,halali:glikosida_steviol,hala
li:antioksidan_asam_askorbat,halali:alfalfa_tokoferol,halali:lemak_daging_sap
i .

halalf:Dua_Kelinci_Snack_Krip_Krip_Tortilla_Sapi_Panggang
food:ingredientsListAsText "jagung", "minyak sawit", "antioksidan tbhq",
"pati jagung", "bumbu sapi panggang", "gula", "garam", "sayuran bubuk",
"maltodekstrin", "protein kedelai terhidrolisis", "rempah-rempah", "natrium
sitrat", "asam sitrat", "antikempal silikon dioksida", "dinatrium
inosinat", "dinatrium guanilat", "glikosida steviol", "antioksidan asam
askorbat", "alfalfa tokoferol", "lemak daging sapi" .
```

Gambar 4.4 Tampilan format RDF dari data LODHalal

Setelah data hasil *scraping* telah selesai diproses dan diubah menjadi bentuk RDF, data tersebut digabungkan dengan RDF dari LODHalal sehingga seluruh data tergabung ke dalam satu kesatuan *knowledge graph* yang akan digunakan untuk melakukan *graph embedding* dan juga *link prediction*.

4.3 Graph Embedding

4.3.1 Node2Vec

Data yang sudah diubah ke bentuk RDF selanjutnya diubah ke bentuk *low-dimensional* atau bentuk vektor atau yang dinamakan juga dengan *graph embedding* menggunakan *Node2Vec*. *Node2Vec* dapat memetakan *nodes* yang ada di dalam *graph* ke bentuk *low-dimensional* dengan memaksimalkan probabilitas *node* yang berdekatan dalam sebuah jaringan. *Graph embedding* dengan *Node2Vec* akan dilakukan dengan menggunakan Neo4J. Hasil dari *graph embedding* tersebut adalah vektor dari tiap *node* produk makanan yang terdapat di dalam graph yang telah dibuat sebelumnya.

4.3.2 RDF2Vec

Data yang sudah diubah ke bentuk RDF selanjutnya diubah ke bentuk *low-dimensional* atau bentuk vektor atau yang dinamakan juga dengan *graph embedding* menggunakan RDF2Vec. RDF2Vec merupakan salah satu algoritma untuk melakukan *graph embedding* yang terinspirasi dari pendekatan Word2Vec dalam melakukan representasi kata ke dalam bentuk vektor numerik. *Graph embedding* dengan RDF2Vec akan dilakukan dengan menggunakan salah satu *package* python yaitu pyRDF2Vec. Hasil dari *graph embedding* tersebut adalah vektor dari tiap entitas produk makanan yang didefinisikan di dalam *knowledge graph* yang digunakan. Secara *default*, RDF2Vec akan menghasilkan 100 buah vektor *embedding* untuk tiap *node*-nya.

4.4 Link Prediction

4.4.1 Mengubah Bentuk Data Pasangan Produk

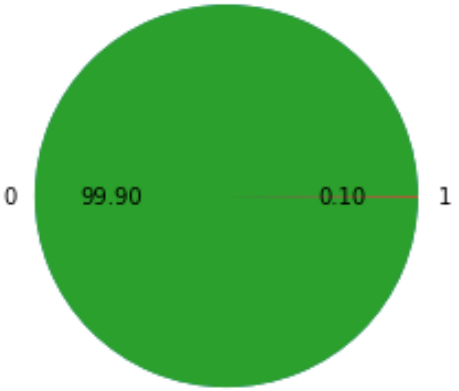
Data awal yang digunakan adalah hasil *generate* tiap pasangan yang ada antara dua buah uri produk makanan dari *knowledge graph* yang telah dibuat sebelumnya. Dari tiap pasangan tersebut akan diubah menjadi menjadi bentuk vektor *embedding* dari masing-masing *node* produk makanannya. Jumlah vektor *embedding* dari masing-masing algoritma yang digunakan adalah 100, yang berarti untuk setiap pasangan produk makanan akan memiliki total 200 fitur berupa vektor *embedding* yang berasal dari kedua produk makanan tersebut.

Selanjutnya untuk menentukan label pada tiap pasangan produk makanan, digunakan nilai similarity dari algoritma Jaccard Similarity [27]. Pada penelitian tersebut, tiap pasangan produk makanan dibersikan nilai similarity. Selanjutnya dilakukan pengecekan manual pada tiap hasil *similarity* antar pasangan produk makanan. Jika tingkat akurasi dari nilai *similarity* antar kedua produk makanan masih bernilai lebih dari 0.8, maka tiap pasangan produk makanan pada nilai similarity tersebut akan diberikan label 1 atau mirip, sedangkan jika tingkat akurasinya sudah dibawah 0.8, maka nilai *similarity* tersebut akan menjadi

nilai *treshhold* yang menandakan pasangan produk makanan tersebut akan diberikan label 0 atau tidak mirip.

4.4.2 Oversampling dan Undersampling Data

Dari hasil labeling yang telah dilakukan, didapati bahwa terdapat perbedaan yang sangat jauh antara jumlah data dari kedua label yang digunakan. Berdasarkan hasil eksplorasi data, jumlah data dengan label 1 hanya berjumlah 0.1% dari keseluruhan data pasangan produk makanan yang dimiliki, sehingga model akan mengalami kesulitan untuk memprediksi hubungan makanan yang berlabel 1. Gambar 4.5 berikut merupakan perbandingan jumlah label yang ada:



Gambar 4.5 Perbandingan jumlah data tiap label

Untuk mengatasi permasalahan tersebut, pada penelitian ini akan dilakukan metode *oversampling* dan *undersampling*. *Oversampling* merupakan cara untuk menambah jumlah data pada label minoritas dengan menambahkan data sintetis pada label tersebut, sedangkan *undersampling* merupakan cara untuk mengurangi jumlah data pada label mayoritas dengan melakukan *sampling* pada data berlabel mayoritas tersebut.

Pada penelitian ini, dalam melakukan *oversampling* data sintetis yang akan dibuat dibatasi sehingga jumlah data sintetis akan tidak melebihi jumlah data asli dari *knowledge graph* yang

digunakan. Selain itu akan dicoba berbagai kombinasi perbandingan antara jumlah data label 0 dan label 1 untuk mengetahui performa model pada masing-masing kombinasi.

4.4.3 Melakukan Tuning Parameter

Pencarian parameter yang paling baik untuk membuat model *link prediction* menggunakan algoritma *Random Forest* akan dilakukan dengan menentukan nilai pada beberapa parameternya, yaitu *max_depth*, *min_samples_leaf*, *min_samples_split*, *n_estimatros*, dan *criterion*. Proses tuning parameter akan dilakukan dengan menggunakan *GridSearchCV* dari *library sklearn*. Tabel 4.1 merupakan *parameter grid* yang akan digunakan untuk mencari parameter yang terbaik:

Tabel 4.1 Parameter grid untuk tuning parameter terbaik

Parameter	Value
max_depth	80, 90, 100, 110, None
min_samples_leaf	1, 2, 3, 4
min_samples_split	2, 4, 6, 8, 10
n_estimators	100, 200, 400, 1000
criterion	gini, entropy

Parameter yang terbaik akan diambil berdasarkan nilai *f1-score*-nya. Tujuan tuning parameter ini adalah untuk memaksimalkan nilai rata-rata dari *accuracy*, *precision*, *recall*, dan *f1-score* pada model akhirnya nanti.Klasifikasi *Support Vector Machine* tersebut akan diuji performa nya apakah model yang sudah dilakukan itu baik atau tidak. Analisa kinerja ini akan menggunakan evaluasi akurasi, *recall* dan presisi.

4.4.4 Membagi Train Set dan Test Set

Pembagian data menjadi *train set* dan *test set* akan dilakukan sebelum melakukan pembuatan model. *Train set* akan digunakan untuk melatih model *link prediction*, sedangkan *test*

set akan digunakan untuk mengevalusasi performa dari model *link prediction* yang telah dibuat. Pembagian *train set* dan *test set* akan dilakukan dengan menggunakan *library sklearn* yaitu *train_test_split*. Pembagian data akan dilakukan dengan menggunakan beberapa bentuk kombinasi. Tabel 4.2 menunjukkan kombinasi antara *train se*t dan *test set* yang akan digunakan untuk menemukan kombinasi yang terbaik:

Tabel 4.2 Kombinasi jumlah train set dan test set

Train Set	Test Set
0.8	0.2
0.75	0.25
0.7	0.3
0.6	0.4

Untuk menentukan kombinasi yang menghasilkan performa *link prediction* yang terbaik, akan dilakukan pengujian model *link prediction* dengan menggunakan metode *5-fold cross validation*.

4.4.5 Link Prediction Menggunakan Random Forest

Random Forest akan dilakukan dengan menggunakan inisiasi parameter *default* dan juga parameter yang didapatkan dari hasil *tuning* parameter. Skenario lainnya seperti pembagian komposisi *train set* dan test set, perbandingan jumlah data antar label, serta algoritma *graph embedding* yang digunakan akan dibuatkan modelnya masing-masing untuk menentukan model terbaik dari keseluruhan skenario yang ada. Model yang terbaik akan digunakan untuk melakukan *link prediction* dan akan dibandingkan dengan algoritma *machine learning* lainnya.

4.5 Pengukuran Performa

Hasil model *link prediction* selanjutnya akan diukur performanya untuk menentukan apakah model yang sudah dibuat itu baik atau tidak. Pengukuran performa akan dilakukan dengan mengevaluasi nilai akurasi, *recall*, *precision*, dan *f1-score*-nya.

Selain itu, dibuat juga akan dibuatkan visualisasi ROC dari berbagai model yang digunakan serta visualiasi dari hasil *link prediction* yang dilakukan. Visualisasi ROC akan digunakan untuk menentukan model *link prediction* yang memiliki performa kinerja klasifikasi yang paling baik dengan menganalisa tiap visualisasi *curve* yang dihasilkan.

Model *Random Forest* juga akan dibandingkan dengan menampilkan hasil *link prediction* menggunakan beberapa algoritma *machine learning* lainnya seperti *Gaussian Naïve Bayes*, *Logistic Regression*, *Support Vector Machine*, dan juga *Gradient Boost Classifier*.

Halaman ini sengaja dikosongkan

BAB V
IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai implementasi tugas akhir yang merupakan bagaimana proses penelitian dilakukan, mulai dari pengumpulan data hingga pengujian model *link prediction*.

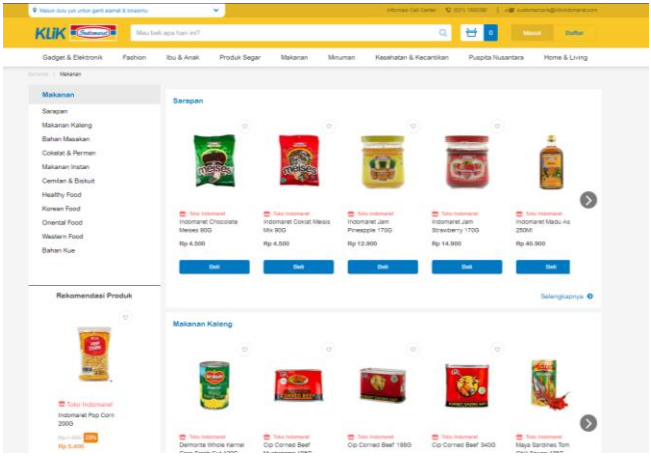
5.1 Pengumpulan Data

Seperti yang telah dijelaskan pada sub bab 4.1, data yang akan dikumpulkan berasal dari dua sumber yaitu *website* KlikIndomaret dan *website* Halal MUI. Untuk tahap *scraping* data yang dilakukan pada kedua *website* cenderung sama, yang membedakan hanya pada struktur dan elemen html yang dimiliki *website* tersebut.

5.1.1 Melakukan Web Scraping

a. Website KlikIndomaret

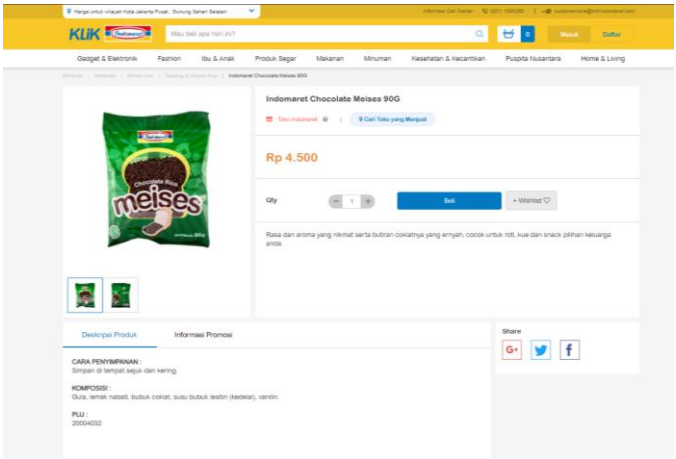
Berikut gambaran dari tampilan halaman *website* KlikIndomaret:



Gambar 5.1 Tampilan produk makanan per sub-kategori

Gambar 5.1 merupakan tampilan *website* KlikIndomaret dimana akan diambil data produk

makanan dari kategori makanan dan kategori minuman. Dari kedua kategori tersebut terdapat beberapa subkategori. Dari tampilan subkategori tersebut, akan diambil url masing-masing produk makanan.



Gambar 5.2 Tampilan per produk makanan

Gambar 5.2 merupakan tampilan dari tiap individual produk makanan yang diperoleh dari url produk makanan pada gambar Gambar 5.1. Dari tampilan ini akan diambil data mengenai nama produk makanan dan komposisi produk makanan tersebut.

Untuk itu terdapat beberapa tahapan yang dilakukan dalam pembuatan kode scraping data dari *website* KlikIndomaret, yaitu:

i. Penulisan Tautan
Pada bagian ini menuliskan tautan yang akan digunakan untuk scraping pada awal kode *scraper*. Tautan yang digunakan disesuaikan dengan tiap subkategori produk makanan dan minuman yang dimaksud. Untuk tautan yang memiliki keberlanjutan di halaman-halaman berikutnya, dapat memanfaat fitur *pagination* untuk memperoleh datanya. Kode 5.1

menunjukkan penulisan tautan pada kode *scraper* KlikIndomaret.

```
import scrapy
from ..items import IndomaretItem
from urllib.parse import urljoin

class crawlindomaret(scrapy.Spider):
    name = 'indomaret'
    page_number = 2
    start_urls = [

'https://www.klikindomaret.com/category/minuman-
tradisional'
        ]
```

Kode 5.1 Penulisan tautan KlikIndomaret

ii. Penulisan Selector

```
def parse(self, response):
        indomaret =
response.xpath("//*[contains(@class,
'item')]/a/@href").extract()
        for i in indomaret:
            url = urljoin(response.url, i)
            yield scrapy.Request(url,
callback=self.parse_item)

    def parse_item(self, response):
        for info in response.css('.produk-detail'):
            item = IndomaretItem()

            nama_produk   = info.css('h3.desktop-
xs::text').extract_first(),
            komposisi     =
info.css('.spec_id_KOMPOSISI::text').get(),

            item['nama_produk'] = nama_produk
            item['komposisi'] = komposisi
            yield item
```

Kode 5.2 Pembuatan selector dan item KlikIndomaret

Kode 5.2 diatas menjelaskan cara iterasi untuk melakukan pengambilan data yang telah dipilih sesuai dengan posisinya menurut elemen html. Selanjutnya dibuatkan item untuk menyimpan data hingga hasilnya dapat terstruktur dengan baik. Pendefinisian *nama_produk* dan *komposisi* menunjukkan letak bagian yang akan diambil dari halaman *website*, yaitu menjunjuk bagian tersebut menggunakan css tepatnya pada bagian 'h3.dekstop-xs' untuk nama produk dan '.spec_id_KOMPOSISI' untuk komposisi.

```
import scrapy
class IndomaretItem(scrapy.Item):
    nama_produk = scrapy.Field()
    komposisi   = scrapy.Field()
    pass
```

Kode 5.3 Pembuatan item field pada file item KlikIndomaret

Kode 5.3 diatas merupakan data *file item* yang ditulis untuk mendefinisikan *item field*. Penulisan *item field* sesuai dengan kode *crawler* yang dibuat pada *file scraper*.

iii. Tahap Pagination

Pada *website* KlikIndomaret, produk makanan yang termuat di dalam halaman *website* tidak semua ditampung ke dalam satu halaman, namun terbagi menjadi beberapa halaman dengan jumlah maksimal 50 produk makanan per halamannya. Sehingga pada kode *scraper* ini dilakukan perulangan untuk mengambil data produk makanan di tiap halaman dengan menambah angka di setiap halamannya.

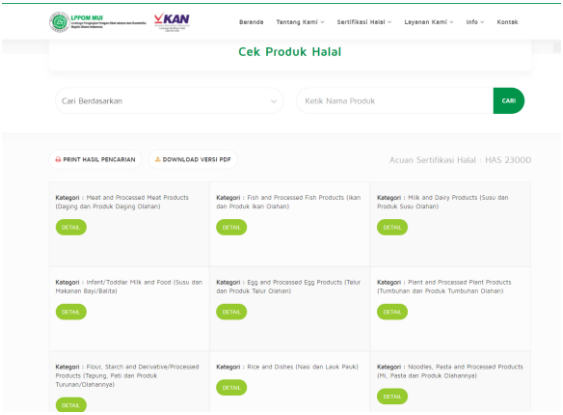
```
next_page =
'https://www.klikindomaret.com/category/cemilan--
biskuit?categories=&sortcol=PROMO&page=' +
str(crawlindomaret.page_number) +
'&pagesize=50&attributes=&productbrandid=&startp
rice=&endprice='
        if crawlindomaret.page_number < 3:
            crawlindomaret.page_number += 1
            yield response.follow(next_page, callback =
self.parse)
```

Kode 5.4 Penulisan kode untuk halaman berikutnya

Pada Kode 5.4 terdapat variabel *page_number* yang sebelumnya telah ditulis pada Kode 5.1, sehingga variabel tersebut dapat langsung digunakan. Selain itu, variabel ini hanya dilakukan untuk perulangan lalu dilakukan *callback* agar tautan utamanya muncul dan ditambah hingga ke *page* dimana *page* tersebut merupakan *page* maksimal yang ada pada halaman *website*.

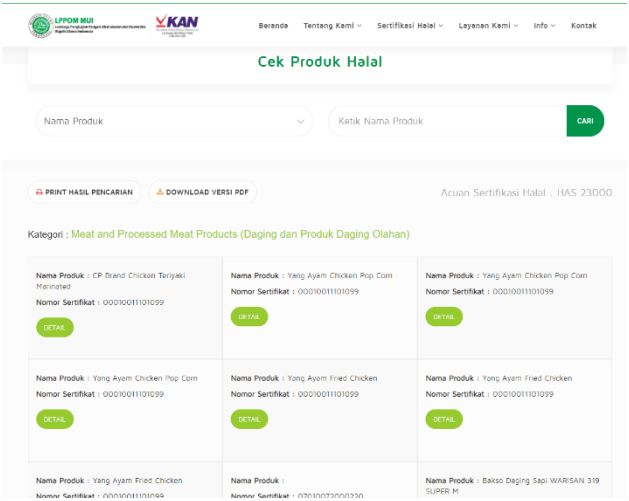
b. Website HalalMUI

Berikut gambaran dari tampilan halaman *website* Halal MUI:



Gambar 5.3 Tampilan kategori produk pada website Halal MUI

Gambar 5.3 merupakan tampilan *website* KlikIndomaret dimana akan diambil data produk makanan dari 36 kategori produk halal.



Gambar 5.4 Tampilan tiap produk dalam kategori

Gambar 5.4 merupakan tampilan dari tiap individual produk halal yang ada di masing-masing kategori produk halal pada Gambar 5.3. Dari tampilan ini akan diambil data mengenai nama produk makanan, produsen, dan sertifikat dari produk makanan tersebut.

Untuk itu terdapat beberapa tahapan yang dilakukan dalam pembuatan kode scraping data dari *website* HalalMUI, yaitu:

i. Penulisan Tautan

Pada bagian ini menuliskan tautan yang akan digunakan untuk scraping pada awal kode *scraper*. Tautan yang digunakan disesuaikan dengan isi dari tiap kategori produk halal yang dimaksud. Untuk tautan yang memiliki keberlanjutan di halaman-halaman berikutnya, dapat memanfaat fitur *pagination* untuk

memperoleh datanya. Kode 5.5 menunjukkan penulisan tautan pada kode *scraper* Halal MUI.

```
import scrapy
from ..items import Halal2Item
from urllib.parse import urljoin

class crawlhalal2(scrapy.Spider):
    name = 'halal2'
    page_number = 2
    start_urls = [

'https://www.halalmui.org/mui14/searchproduk/search
/detailgroupkategori/?groupcode=27'
        ]
```

Kode 5.5 Penulisan tautan HalalMUI

ii. Penulisan Selector

Kode 5.7 menjelaskan cara iterasi untuk melakukan pengambilan data yang telah dipilih sesuai dengan posisinya menurut elemen html. Selanjutnya dibuatkan item untuk menyimpan data hingga hasilnya dapat terstruktur dengan baik. Pendefinisian tiap *nama_produk* menunjukkan letak bagian yang akan diambil dari halaman *website*, yaitu menunjuk bagian tersebut menggunakan xpath tepatnya pada bagian "/html/body/div[6]/div/div/div[2]/table/tbody/tr[(1-10)]/td/a" untuk tiap produknya. Digunakan penomoran 1 hingga 10 dikarenakan di tiap halamannya terdapat 10 produk makanan yang berbeda.

```
def parse(self, response):
        items = Halal2Item()
        nama_produk1 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
1]/td/a").extract()
        nama_produk2 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
2]/td/a").extract()
        nama_produk3 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
3]/td/a").extract()
        nama_produk4 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
4]/td/a").extract()
        nama_produk5 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
5]/td/a").extract()
        nama_produk6 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
6]/td/a").extract()
        nama_produk7 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
7]/td/a").extract()
        nama_produk8 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
8]/td/a").extract()
        nama_produk9 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
9]/td/a").extract()
        nama_produk10 =
response.xpath("/html/body/div[6]/div/div/div[2]/table/tbody/tr[
10]/td/a").extract()
```

```
        items['nama_produk1'] = nama_produk1
        items['nama_produk1'] = nama_produk2
        items['nama_produk1'] = nama_produk3
        items['nama_produk1'] = nama_produk4
        items['nama_produk1'] = nama_produk5
        items['nama_produk1'] = nama_produk6
        items['nama_produk1'] = nama_produk7
        items['nama_produk1'] = nama_produk8
        items['nama_produk1'] = nama_produk9
        items['nama_produk1'] = nama_produk10
        yield items
```

Kode 5.7 Penulisan selector dan item HalalMUI

Kode 5.6 merupakan data *file item* yang ditulis untuk mendefinisikan *item field*. Penulisan *item field* sesuai dengan kode *crawler* yang dibuat pada *file scraper*.

```
import scrapy
class Halal2Item(scrapy.Item):
    nama_produk1 = scrapy.Field()
    pass
```

Kode 5.6 Penulisan item field pada file item HalalMUI

iii. Tahap Pagination

Pada *website* Halal MUI, produk makanan yang termuat di dalam suatu kategori produk halal, tidak semua ditampung ke dalam satu halaman, namun terbagi menjadi beberapa halaman dengan jumlah maksimal 10 produk makanan per halamannya.

```
next_page =
'https://www.halalmui.org/mui14/searchproduk/search/det
ailgroupkategori/?groupcode=27&page=' +
str(crawlhalal2.page_number)
        if crawlhalal2.page_number < 324:
            crawlhalal2.page_number += 1
            yield response.follow(next_page, callback =
self.parse)
```

Kode 5.8 Penulisan kode untuk halaman berikutnya

Pada Kode 5.8 terdapat variabel *page_number* yang sebelumnya telah ditulis pada Kode 5.5, sehingga variabel tersebut dapat langsung digunakan. Selain itu, variabel ini hanya dilakukan untuk perulangan lalu dilakukan *callback* agar tautan utamanya muncul dan ditambah hingga ke *page* dimana *page* tersebut merupakan *page* maksimal yang ada pada halaman *website*.

5.1.2 Membersihkan Data

a. Hasil Scraping KlikIndomaret

Proses *web scraping* pada *website* KlikIndomaret menghasilkan data sebanyak 1.867 produk makanan dan minuman. Gambar 5.5 merupakan beberapa contoh data produk makanan hasil *scraping* dengan format json dari *website* KlikIndomaret:


```
{"nama_produk": "Goody Pudding With Nata De Coco 3's Mango 240G ",
"komposisi": "Air , Nata De Coco , Gula , Sari Buah Mangga , Susu Bubuk ,
Ekstrak Rumput Laut , Pengatur Keasaman "},
{"nama_produk": "Dale's Farm Creamy Peanut Butter 250G", "komposisi":
"kacang tanah panggang (89%), gula, minyak nabati, minyak kacang tanah,
garam."},
{"nama_produk": "Pondan Sponge Cake Mix Vanilla 200G", "komposisi": null},
{"nama_produk": "Nutella Jam Spread Hazelnut With Cocoa 350G", "komposisi":
 "Gula, Lemak Nabati, Kacang Hazel 13%, Susu SKIM Bubuk 8.7%, Kakao Bubuk
7.4%, Pengemulsi : Lesitin (Kedelai), Perisa Sintetik Vanila."},
{"nama_produk": "Goody Pudding With Nata De Coco 3's Mixed 240G",
"komposisi": "AIR, NATA DE COCO, GULA, SARI BUAH LECI, STRAWBERRY, MANGGA,
SUSU BUBUK, EXTRAXT RUMPUT LAUT, ASAM SITRAT, PERISAI SINTETIK (LECI,
STRAWBERRY,MANGGA), PEWARNA (TARTRAZIN CI 19140) MERAH ALLURA CI 16035"},
{"nama_produk": "Dale's Farm Crunchy Peanut Butter 250G", "komposisi":
"kacang tanah panggang (89%), gula, minyak nabati, minyak kacang tanah,
garam."},
{"nama_produk": "Goody Pudding With Nata De Coco 3's Lychee 240G ",
"komposisi": "Air , Nata De Coco , Gula , Sari Buah Lychee , Susu Bubuk ,
Ekstrak Rumput Laut , Pengatur Keasaman "},
{"nama_produk": "Nutella Jam Spread Hazelnut With Cocoa 680G", "komposisi":
 "Gula, Lemak Nabati, Kacang Hazel 13%, Susu SKIM Bubuk 8.7%, Kakao Bubuk
7.4%, Pengemulsi : Lesitin (Kedelai), Perisa Sintetik Vanila."},
{"nama_produk": "Pondan Sponge Cake Mix Pandan 200G", "komposisi": null},
{"nama_produk": "Ovomaltine Crunchy Cream 680g", "komposisi": "Ovomaltine
Bubuk 33%, Ekstrak barley Malt, Susu Skim kental, serum susu kental, 13.3%
kako bubuk rendah lemak, gula, fruktosa, mineral, minyak rapeseed, Vitamin
A,E,C, thiamin, riboflavin, niasin, B6, asam folat, B12, biotin, asam
pantotenat, garam, perisa sintetik vanila, kacang Hazel, pengemulsi
nabati, penstabil fosfat, minyak biji bunga matahari."},
{"nama_produk": "Kraft All In One Cheddar 165G", "komposisi": null},
{"nama_produk": "Pondan Brownies Kukus / Panggang Chocolate 230G",
"komposisi": null},
```


Gambar 5.5 Tampilan data hasil scraping KlikIndomaret

Gambar 5.5 di atas menampilkan hasil *scraping* yang terdiri dari dua buah atribut, yaitu *nama produk* dan *komposisi*. Seperti yang terlihat pada gambar, untuk seluruh nama produk diberikan akhiran berupa ukuran dari produk tersebut. Maka dari itu, data perlu diproses dengan cara menghapus seluruh kata terakhir yang ada pada nama produk. Untuk mempermudah pembersihan nama produk tersebut, data diubah terlebih dahulu ke bentuk csv agar dapat dibuka dengan bantuan Microsoft Excel. Selanjutnya dilakukan pembersihan nama produk dengan langkah sebagai berikut:

1. Mencari nama produk yang tidak berakhiran dengan ‘g’ atau ‘l’

   Dikarenakan setiap nama produk diakhiri dengan ukuran gram ataupun liter, maka langkah pertama adalah mencari nama produk yang tidak memiliki huruf akhir ‘g’ dan ‘l’

2. Menambahkan satu acak pada nama produk yang tidak memiliki ukuran

   Untuk nama produk yang tidak diakhiri dengan ukuran, selanjutnya diberikan sebuah kata sementara yang berfungsi untuk dihapus pada langkah selanjutnya.

3. Menghapus seluruh kata terakhir pada nama produk

   Langkah terakhir adalah menghapus seluruh kata terakhir yang ada pada kolom nama produk.

Tabel 5.1 berikut merupakan hasil dari proses pembersihan nama produk data KlikIndomaret:

Tabel 5.1 Data KlikIndomaret hasil cleaning nama_produk

nama_produk	komposisi
Floridina Juice Pulp Orange	Air, gula, bulir jeruk (4%), konsentrat jeruk (22%), perisa jeruk, pengatur keasaman (asam sitrat, natrium sitrat), vitamin C dan pewarna natural
Yeo's Drink Grass Jelly	Air, gula, ekstrak cincau, Tepung jagung, pewarna karamel, Ekstrak licorice
Indomilk Kental Manis Putih	Gula, susu sapi segar, susu bubuk skim, minyak nabati, butter milk bubuk, air, laktosa, perisa identik alami, antioksidan (tokoferol), vitamin A, B1 dan D3
Finna Snack Kerupuk Udang Pars Seaweed	Tepung tapioka, minyak nabati, udang, gula, tepung beras, bumbu rumput laut, garam, perisa alami minyak atsiri papika, pengembang

Selanjutnya dilakukan *drop duplicates* untuk seluruh produk yang memiliki nama produk yang sama sehingga menghasilkan 1.281 produk makanan yang berbeda.

Langkah yang terakhir dilakukan pada tahap preprocessing hasil scraping dari *website* KlikIndomaret adalah melakukan *cleaning* secara manual untuk menyeragamkan penamaan komposisi yang ada. Dikarenakan penulisan bahan makanan yang ada pada kolom komposisi tidak teratur dari *website* terkait, maka dibutuhkan penyeragaman nama pada bahan-bahan makanan tersebut. Penyeragaman nama dilakukan secara manual. Tujuannya adalah agar sebuah hubungan antara komposisi dan produk

makanan yang ada pada data hasil scraping menjadi lebih baik untuk diproses pada tahap selanjutnya.

b. Hasil Scpraing HalalMUI

Setelah dilakukan web scraping pada *website* KlikIndomaret, maka diperoleh data sebanyak 401.650 produk makanan dan minuman. Gambar *5.6* merupakan contoh hasil *scraping* dengan format json dari *website* Halal MUI:


```
{"nama_produk": "id=\"109|21949|\tcyber park karawaci lt. 2 (head
office)|Pt. matahari putra prima,
tbk|Hs1a20415/032020/mpp|17-march-2022\">" },{ "nama_produk":
"id=\"121|21949|\tfoodmart surabaya town square|Pt. matahari putra prima,
tbk|Hs1a20415/032020/mpp|17-march-2022\">" },{ "nama_produk":
"id=\"95|21949|\thypermart batam tanjung pinang|Pt. matahari putra prima,
tbk|Hs1a20415/032020/mpp|17-march-2022\">" },{ "nama_produk":
"id=\"66|21949|\thypermart bau bau|Pt. matahari putra prima,
tbk|Hs1a20415/032020/mpp|17-march-2022\">" },{ "nama_produk":
"id=\"45|21949|\thypermart ciputra world surabaya|Pt. matahari putra
prima, tbk|Hs1a20415/032020/mpp|17-march-2022\">" },{ "nama_produk":
"id=\"46|21949|\thypermart kediri|Pt. matahari putra prima,
tbk|Hs1a20415/032020/mpp|17-march-2022\">" },{ "nama_produk":
"id=\"61|21949|\thypermart maluku city mall|Pt. matahari putra prima,
tbk|Hs1a20415/032020/mpp|17-march-2022\">" },{ "nama_produk":
"id=\"92|21949|\thypermart pentacity balikpapan|Pt. matahari putra prima,
tbk|Hs1a20415/032020/mpp|17-march-2022\">" },{ "nama_produk":
"id=\"11|32388|Air minum dalam kemasan (amdk)|Pt. hokkan deltapack
industri|Hs1a21961/112020/hkd|17-november-2022\">" },{ "nama_produk":
"id=\"1|32388|Airness|Pt. hokkan deltapack
industri|Hs1a21236/082020/hkd|18-august-2022\">"},
{"nama_produk": "id=\"6|32388|Alfa one|Pt. hokkan deltapack
industri|Hs1a21236/082020/hkd|18-august-2022\">" },{ "nama_produk":
"id=\"17|32388|Alham|Pt. hokkan deltapack
industri|Hs1a21961/112020/hkd|17-november-2022\">" },{ "nama_produk":
"id=\"24|32388|Alham|Pt. hokkan deltapack
industri|Hs1a21961/112020/hkd|17-november-2022\">" },{ "nama_produk":
"id=\"14|32388|Amoz|Pt. hokkan deltapack
industri|Hs1a21961/112020/hkd|17-november-2022\">" },{ "nama_produk":
"id=\"21|32388|Amoz|Pt. hokkan deltapack
industri|Hs1a21961/112020/hkd|17-november-2022\">" },{ "nama_produk":
"id=\"3|5355|Cabang cakung|Pt. iron
bird|Hs1a21874/112020/irb|10-november-2022\">" },{ "nama_produk":
"id=\"6|5355|Cabang cikarang|Pt. iron
bird|Hs1a21874/112020/irb|10-november-2022\">" },{ "nama_produk":
"id=\"1|47551|Chil go morinaga|Pt. kalbe milko
```


Gambar 5.6 Tampilan data hasil scraping HalalMUI

Gambar *5.6* di atas menampilkan data hasil *scraping* pada *website* Halal MUI yang agak berbeda dibandingkan dengan data hasil scraping pada *website* KlikIndomaret. Pada data hasil *scraping website* Halal MUI, seluruh atribut yang dibutuhkan tergabung ke dalam satu atribut "*nama_produk*" yang dipisahkan

oleh tanda '|'. Atribut tersebut meliputi nama produk, nama produsen, nomor sertifikasi halal, serta tanggal kadaluarsa dari sertifikat halal produk makanan terkait.

Maka dari itu, Kode 5.9 digunakan untuk mengolah data dengan membagi string tersebut ke masing-masing atributnya yang sesuai:

```
new = df["nama_produk"].str.split("|", n = 6, expand = True)
```

Kode 5.9 Membagi data menjadi 6 kolom terpisah

Selanjutnya tiap-tiap kolom diberikan nama atribut yang sesuai sehingga data yang sebelumnya ditampilkan diubah ke bentuk seperti pada Tabel 2.1 berikut:

Tabel 5.2 Data HalalMUI hasil cleaning

no	**id**	**nama_produk**	**produsen**	**sertif**	**tanggal**
114	5095	Biscuit kokola narita mari susu roll	Pt. Unimos	00100083330617	23-july-2021
110	5095	Biskuit kokola rose cream coklat	Pt. Unimos	00100083330617	23-july-2021
132	5095	Biskuit kokola rose cream lemon	Pt. Unimos	00100083330617	23-july-2021
134	5095	Biskuit kokola rose cream pineapple	Pt. Unimos	00100083330617	23-july-2021
162	5095	Biscuit kokola silaturahmi assorted	Pt. Unimos	00100083330617	23-july-2021

Langkah yang terakhir adalah melakukan *drop duplicates* untuk seluruh produk dengan nama yang sama sehingga menghasilkan 236.660 produk yang berbeda.

5.1.3 Melakukan Matching Data

Proses matching data dilakukan untuk menentukan kehalalan suatu produk makanan pada *website* KlikIndomaret dengan mencocokan produk makanan tersebut dengan produk yang diperoleh dari *website* Halal MUI. Untuk melakukan *matching* digunakan algoritma *Jaccard Similarity* dengan mencocokan nama produk pada data KlikIndomaret dan nama produk pada data Halal MUI. Kode 5.10 digunakan untuk melakukan *matching data*:

```
def jaccard_similarity(list1, list2):
    s1 = set(list1)
    s2 = set(list2)
    return float(len(s1.intersection(s2)) / len(s1.union(s2)))

def get_response(q):
            highsim=0.0
            highnum=0
            for i,line in enumerate(data):
                        if (type(line)==str):
                                    qlist=cleandata(q).lower().split()
                                    linelist=cleandata(line).lower().split()

            simnow=jaccard_similarity(qlist,linelist)
                                    if(simnow>highsim):
                                                highnum=i
                                                highsim=simnow
            return(highnum)
```

```
def get_response2(q):
        highsim=0.0
        highnum=0
        for i,line in enumerate(data):
                if (type(line)==str):
                        qlist=cleandata(q).lower().split()
                        linelist=cleandata(line).lower().split()

        simnow=jaccard_similarity(qlist,linelist)
                        if(simnow>highsim):
                                highnum=i
                                highsim=simnow
        return(highsim)
```

Kode 5.10 Mendapatkan nilai similarity terbaik

Fungsi *get_response1* pada Kode 5.10 di atas digunakan untuk menentukan produk dari Halal MUI yang memiliki nilai *jaccard similarity* jika dibandingkan dengan produk dari KlikIndomaret. Sedangkan fungsi *get_response2* digunakan untuk menampilkan nilai *jaccard similarity* antara produk makanan dari kedua sumber tersebut. Gambar 5.7 merupakan contoh hasil dari matching dari kedua sumber data menggunakan algoritma *Jaccard Similarity*:

```
,Produk,produkindo,produkmui,produsen,sertif,nilai
487,Dua Kelinci Kacang Sangrai|Dua kelinci kacang
sangrai|Pt. dua kelinci|00100017571001|0.8,Dua Kelinci
Kacang Sangrai,Dua kelinci kacang sangrai,Pt. dua
kelinci,00100017571001,0.8
483,Dua Kelinci Kacang Polong Polongmas Ayam Bawang|Dua
kelinci polongmas kacang polong rasa ayam bawang|Pt. dua
kelinci|00100017571001|0.7777777777777778,Dua Kelinci
Kacang Polong Polongmas Ayam Bawang,Dua kelinci
polongmas kacang polong rasa ayam bawang,Pt. dua
kelinci,00100017571001,0.7777777777777778
394,Dua Kelinci Jagung Presto Bawang Putih|Dua kelinci
jagung presto rasa bawang putih|Pt. dua
kelinci|00100017571001|0.75,Dua Kelinci Jagung Presto
Bawang Putih,Dua kelinci jagung presto rasa bawang
putih,Pt. dua kelinci,00100017571001,0.75
548,Frisian Flag Susu Bubuk Full Cream|Frisian flag susu
bubuk full cream|Pt. frisian flag
indonesia|00040040520606|0.75,Frisian Flag Susu Bubuk
Full Cream,Frisian flag susu bubuk full cream,Pt.
frisian flag indonesia,00040040520606,0.75
395,Dua Kelinci Jagung Presto Pedas Manis|Dua kelinci
jagung presto rasa pedas manis|Pt. dua
kelinci|00100017571001|0.75,Dua Kelinci Jagung Presto
Pedas Manis,Dua kelinci jagung presto rasa pedas
manis,Pt. dua kelinci,00100017571001,0.75
91,Frisian Flag Krimer Kental Manis Omela|Omela krimer
kental manis|Pt. frisian flag
indonesia|00040040520606|0.75,Frisian Flag Krimer Kental
Manis Omela,Omela krimer kental manis,Pt. frisian flag
indonesia,00040040520606,0.75
```

Gambar 5.7 Tampilan hasil matching menggunakan Jaccard Similarity

Data hasil jaccard similarity di atas selanjutnya dipisahkan dengan menggunakan '|' untuk tiap barisnya sehingga menghasilkan data dengan 5 buah kolom, yaitu '*produkindo*', '*produkmui*', '*produsen*', '*sertif*', dan '*nilai*'. Kode 5.11 digunakan untuk memisahkan nama menjadi kolom-kolom yang sesuai.

```
new = match.Produk.str.split("|", n = 5, expand=True,)
match["produkindo"]= new[0]
match["produkmui"] = new[1]
match["produsen"] = new[2]
match["sertif"] = new[3]
match["nilai"] = new[4]
match
```

Kode 5.11 Membagi data ke atribut yang sesuai

Untuk memastikan data hasil matching dengan menggunakan *jaccard similarity* akurat, dilakukan labeling ulang secara manual. Labeling manual tersebut dilakukan ketika nilai *jaccard similarity* yang dihasilkan bernilai minimal 0.3. *Treshhold* tersebut digunakan karena pada nilai *similarity* 0.3, masih banyak terdapat hasil *match data* yang benar setelah dilakukan labeling secara manual. Bagi data produk makanan dari KlikIndomaret yang hasil *matching*-nya dilabeli salah, maka produk makanan tersebut hanya akan memiliki atribut nama produk dan komposisi.

5.2 Graph Modelling

Data hasil *scraping* yang telah diproses pada tahap-tahap sebelumnya, selanjutnya diubah ke bentuk RDF dengan format *Turtle*. Tujuannya adalah format data tersebut sesuai dengan format RDF dari data milik LODHalal. Dengan melakukan penyamaan format dengan data milik LODHalal, kedua sumber data dapat dengan mudah untuk diintegrasikan.

Kode 5.12 berikut merupakan *prefix* yang digunakan guna membuat *file* RDF dengan bentuk Turtle:

```
prefix =
"""@prefix  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#>.
@prefix  rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix  owl: <http://www.w3.org/2002/07/owl#>.
@prefix  xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix  foaf: <http://xmlns.com/foaf/0.1/>.
@prefix  halalv: <http://halal.addi.is.its.ac.id/halalv.ttl#>.
@prefix  halalf: <http://halal.addi.is.its.ac.id/foodproducts/>.
@prefix  halali: <http://halal.addi.is.its.ac.id/ingredients/>.
@prefix  halalc: <http://halal.addi.is.its.ac.id/certificates/>.
@prefix  halals: <http://halal.addi.is.its.ac.id/sources/>.
@prefix  halalm: <http://halal.addi.is.its.ac.id/manufactures/>.
@prefix  food: <http://purl.org/foodontology#> .
@prefix  foodlirmm: <http://data.lirmm.fr/ontologies/food#> .
@prefix  gr: <http://purl.org/goodrelations/v1#> .
@prefix  dbr: <http://dbpedia.org/resource/> ."""
```

Kode 5.12 Prefix untuk file RDF

Pada RDF yang akan dibangun, terdapat beberapa node yang akan dibuat, yaitu Node FoodProduct, Manufacture, Certificates, dan Ingredient.

1. Node FoodProduct
   Node FoodProduct dibuat untuk mendefinisikan seluruh produk makanan yang telah didapatkan melalui proses scraping dari website KlikIndomaret. Kode 5.13 digunakan untuk membuat bentuk RDF dari *node FoodProduct*:

```
def createProductTTL(row, path):
   productString = """halalf:{slugName} a
foodlirmm:FoodProduct;
            a food:Food;
            foodlirmm:code "{code}";
            rdfs:label "{name}";
            gr:hasManufacturer halalm:{slugManufacture} .

halalf:{slugName} foodlirmm:certificate
halalc:{slugCertificate} .

halalf:{slugName} food:containsIngredient
{concateIngredientsEntity} .

halalf:{slugName} food:ingredientsListAsText
{concateIngredientsString} . """
```

Kode 5.13 Membuat node FoodProduct

2. Node Manufacture

   Node Manufacture dibuat untuk mendefinisikan seluruh produsen dari tiap produk makanan yang ada. Produsen-produsen tersebut didapatkan melalui proses scraping dari website HalalMUI. Kode 5.14 digunakan untuk membuat bentuk RDF dari Node Manufacture:

```
def createManufactureTTL(manufacture, path):
   manufactureText = """halalm:{slug} a
foaf:Organization;
                    rdfs:label "{name}" .

                    """
```

Kode 5.14 Membuat node Manufacture

3. Node Certificate

   Node Certificate dibuat untuk mendefinisikan seluruh nomor sertifikat dari tiap produk makanan yang telah tersertifikasi halal. Nomor sertifikat tersebut didapatkan melalui proses scraping dari website

HalalMUI. Kode 5.15 digunakan untuk membuat bentuk RDF dari Node Certificates:

```
def createCertificateTTL(row, path):
    certString = """halalc:{slugCertificate} a
halalv:HalalCertificate;
                                halalv:halalCode
"{certificate}";
                                halalv:halalExp
"{expired}"^^xsd:date;
                                halalv:OrgCert
halals:{slugOrg} .

                                """
```

Kode 5.15 Membuat node Certificate

4. Node Ingredient

   Node Ingredient dibuat untuk mendefinisikan seluruh komposisi makanan dari tiap produk makanan yang ada. Komposisi-komposisi tersebut didapatkan melalui proses scraping dari website KlikIndomaret. Dikarenakan tiap baris komposisi makanan mengandung berbagai macam bahan makanan, maka dibutuhkan pemisahan tiap makanan dengan menyimpan bahan makanan tersebut sebagai bahan makanan yang unik. Kode 5.16 digunakan untuk mendefinisikan tiap bahan makanan dari seluruh komposisi makanan yang ada:

```
def getUniqueIngredients(list):
    ingredientList = []
    for data in list:
        #print("Data: "+data)
        for ingredient in str(data).split(","):
            #print("Get ingredient: "+ingredient)
            ingredient = ingredient.strip()
            ingredientList.append(ingredient)
    #newingredientList =
list(dict.fromkeys(ingredientList))
    return ingredientList
```

Kode 5.16 Menyimpan bahan makanan dalam atribut komposisi

Selanjutnya Kode 5.17 digunakan untuk membuat bentuk RDF dari Node Ingredient:

```
def createIngredientSeparate(ingredient, path):
    ingText = """halali:{sluging} a foodlirmm:Ingredient;
   rdfs:label "{nameing}" .
   """
```

Kode 5.17 Membuat node Ingredient

Setelah tiap node selesai dibuat, dilakukan *merge* antar *nodes* nya sehingga sebuah file RDF yang berisikan seluruh data yang hasil scraping yang telah diproses dapat diintegrasikan dengan data LODHalal dan digunakan untuk proses selanjutnya.

5.3 Graph Embedding

5.3.1 Node2Vec

Graph embedding dengan algoritma *Node2Vec* dilakukan dengan menggunakan aplikasi Neo4J. Untuk melakukan *graph embedding*, dibutuhkan *plugin neosemantics* untuk melakukan *import file* RDF berbentuk *Turtle* yang telah dibuat pada tahap sebelumnya. Selain itu juga dibutuhkan *plugin Graph Data Science Library* yang akan digunakan untuk mengaplikasikan algoritma *Node2Vec* pada *knowledge graph* yang telah di-*import* menggunakan *neosemantics*.

Untuk melakukan *import*, *file* RDF yang telah dibuat sebelumnya di-*upload* menggunakan github. Selanjutnya, untuk melakukan *import* cukup dilakukan dengan menggunakan url dari url github tersebut. Kode 5.18 digunakan untuk melakukan import dari file:

```
CALL
n10s.rdf.import.fetch("https://raw.githubusercontent.com/utomo
girraz/Halal-Tutorial/main/allturtle.ttl","Turtle")
```

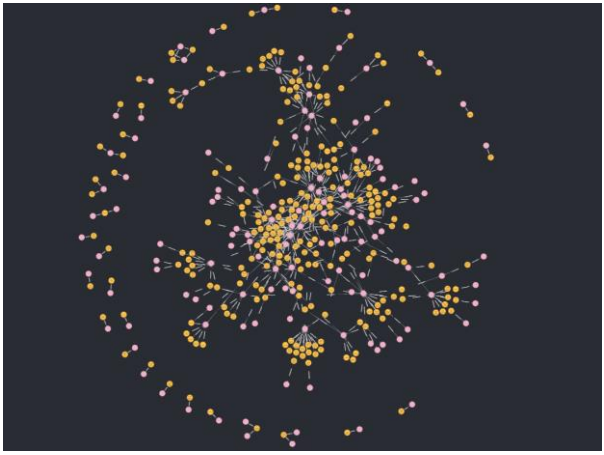
Kode 5.18 Melakukan import file RDF

Setelah melakukan *import fetch*, Neo4J akan mendefinisikan *node* pada Neo4J berjumlah 90.000 dan relasi berjumlah 50.000 yang ada pada database hasil *import fetch* dari url sebelumnya. Dikarenakan pada data LODHalal terdapat banyak sekali produk makanan yang tidak memiliki atribut selain nama produk, maka untuk tiap node produk makanan yang tidak memiliki komposisi pada LODHalal akan dihapus dari Neo4J. Kode 5.19 digunakan untuk menghapus node tersebut:

```
MATCH (n:Food)
OPTIONAL MATCH (n)-
[r:food__containsIngredient]>(i:Resource)
WITH n, i
WHERE i is Null
DETACH DELETE n
```

Kode 5.19 Menghapus node FoodProduct

Dengan melakukan penghapusan *node* dengan Kode 5.19 di atas, jumlah *node* yang ada menjadi berjumlah 20.000 dan relasi yang berjumlah 30.000. Gambar 5.8 merupakan bentuk visualisasi dari seluruh *node* dan relasi yang telah di-*import* menggunakan Neo4J:



Gambar 5.8 Visualisasi nodes dan edges dari file RDF pada Neo4J

Selanjutnya dibuat sebuah *graph* dengan nama '*myGraph*' dari seluruh *node* dan relasi yang sudah ada dalam database Neo4J. Kode 5.20 digunakan untuk membuat graph baru:

```
CALL gds.graph.create('myGraph', '*', '*');
```

Kode 5.20 Membuat graph dari nodes dan edges yang sudah diimport

Setelah *graph* telah didefinisikan, selanjutnya dilakukan proses *graph embedding* dengan menggunakan algoritma *Node2Vec* yang tersedia pada Neo4J. Kode 5.21 digunakan untuk melakukan proses *graph embedding*:

```
CALL gds.alpha.Node2Vec.stream('myGraph',
{embeddingDimension: 100})
YIELD nodeId, embedding
RETURN nodeId, embedding
```

Kode 5.21 Melakukan graph embedding dengan Node2Vec

Dari hasil *graph embedding* yang dilakukan, dapat ditentukan jumlah vektor *embedding* yang akan dihasilkan dengan melakukan penyesuain angka pada parameter *embeddingDimension*. Dari kode diatas akan dihasilkan uri dan 100 vektor *embedding* dari *node* yang dirujuk.

Langkah terakhir adalah memasukkan tiap vektor *embedding* hasil dari *graph embedding* ke kolom yang berbeda. Tujuannya adalah nantinya vektor-vektor tersebut dapat digunakan sebagai *independent variabel* pada tahap *link prediction*. Gambar 5.9 merupakan bentuk *embedding* yang dapat digunakan untuk tahap *link prediction*:

	uri	1	2	3	4	5	6	7	8	9	...	91
0	http://halal.addi.is.its.ac.id/manufactures/Je...	-0.895698	-0.652424	-0.352661	0.977000	0.158936	0.748668	0.517554	0.480040	-0.693911	...	-0.962404
1	http://halal.addi.is.its.ac.id/manufactures/Am...	-0.413752	-0.425428	-0.118045	-0.487334	-0.981375	0.024288	0.156467	0.573134	0.345220	...	0.044928
2	http://halal.addi.is.its.ac.id/manufactures/St...	0.978251	-0.121332	0.924200	0.431506	-0.762944	0.891782	-0.375167	0.448796	-0.183074	...	0.742334
3	http://halal.addi.is.its.ac.id/manufactures/Li...	-0.080296	0.476265	-0.516736	0.671542	-0.995670	-0.889545	-0.308378	-0.197450	-0.649190	...	-0.209189
4	http://halal.addi.is.its.ac.id/manufactures/Yi...	-0.596870	0.193437	0.035567	0.992794	-0.433264	-0.736693	0.580241	-0.682021	-0.738618	...	-0.680717
...	...	...	...	...	...	...	...	...	...	...	...	...
21234	http://halal.addi.is.its.ac.id/manufactures/Pf...	0.223399	-0.565066	0.818777	0.644069	0.507565	-0.288127	0.381643	0.460203	0.319395	...	-0.298752
21235	http://halal.addi.is.its.ac.id/manufactures/Te...	-0.090774	0.389922	0.539096	0.732077	0.215295	0.640650	-0.196415	0.773062	-0.438253	...	0.914787
21236	http://halal.addi.is.its.ac.id/manufactures/By...	0.984142	0.970452	0.360086	0.820587	0.173289	0.341781	0.078333	-0.235417	0.755542	...	0.747236
21237	http://halal.addi.is.its.ac.id/manufactures/My...	0.805863	-0.648494	0.735636	-0.475341	0.651464	-0.005647	0.834465	0.636916	-0.669472	...	0.019946
21238	http://halal.addi.is.its.ac.id/manufactures/Mr...	-0.668620	-0.667673	-0.711002	-0.137318	0.548262	0.411388	0.560332	0.932369	0.999155	...	0.474497

21239 rows × 101 columns

Gambar 5.9 Tampilan graph embedding Node2Vec untuk link prediction

5.3.2 RDF2Vec

Graph embedding dengan algoritma RDF2Vec dilakukan dengan menggunakan python, yaitu dengan bantuan *library pyrdf2vec* versi 0.1.1. Algoritma RDF2Vec menggunakan bantuan dari algoritma Word2Vec untuk melakukan transformasi tiap kata yang ada ke bentuk vektornya.

Untuk melakukan graph embedding dengan pyrdf2vec, dibutuhkan pendefinisian entitas yang jelas yang akan muncul dari knowledge graph. Maka dari itu, dibuat *file* csv yang berisikan seluruh uri dari *node* FoodProduct dari *file* RDF yang telah dibuat. Selanjutnya *file* csv tersebut dimasukkan ke dalam python dengan Kode 5.22 sebagai berikut:

```
data = pd.read_csv('export.csv')
entities = data['product']
```

Kode 5.22 Melakukan import file export.csv

Dengan kode di atas, maka seluruh uri dari FoodProduct sudah didefinisikan. Selanjutnya adalah membuat *Knowledge Graph* dari *file* RDF. Dikarenakan pyrdf2vec hanya dapat digunakan menggunakan *file* RDF dengan format *n-triples*, maka *file* RDF dengan format *Turtle* yang sebelumnya telah dibuat perlu untuk diubah terlebih dahulu ke format *n-triples*. Selanjutnya *file* RDF tersebut dapat dimasukkan ke dalam python untuk mendefinisikan *knowledge graph* dengan Kode 5.23 sebagai berikut:

```
from pyrdf2vec.graphs import KG
kg = KG('data.nt',
      label_predicates=[rdflib.URIRef(x) for x in
label_predicates])
```

Kode 5.24 Mendefinisikan file n-tiples menjadi knowledge graph

Selanjutnya dilakukan *graph embedding* untuk seluruh *entities* yang telah didefinisikan sebelum dari *file* csv dengan menggunakan Kode 5.23 sebagai berikut:

```
transformer =
RDF2VecTransformer(walkers=[RandomWalker(2, 5)])
walk_embeddings = transformer.fit(kg,
entities).transform(entities)
```

Kode 5.23 Melakukan graph embedding dengan RDF2Vec

Hasil dari kode di atas akan menghasilkan vektor *embedding* untuk seluruh *entities* yang telah didefinisikan. Untuk vektor *embedding* yang dihasilkan secara *default* berjumlah 100 buah vektor. Urutan dari vektor *embedding* akan sesuai dengan urutan *entities* yang telah didefinisikan sebelumnya untuk penyesuaian. Selanjutnya hasil dari *graph embedding* dan *entities*-nya digabungkan sehingga tiap *entities* memiliki nilai vektor *embedding* yang jelas. Selain itu, untuk tiap vektor *embedding* hasil dari *graph embedding* akan dibagi ke kolom yang berbeda. Tujuannya adalah nantinya vektor *embedding* tersebut dapat digunakan sebagai *independent variabel* pada tahap *link prediction*. Gambar 5.10 merupakan bentuk *embedding* yang dapat digunakan untuk tahap *link prediction*:

	uri	1	2	3	4	5	6	7	8	9	...	91
0	http://halal.addi.is.its.ac.id/manufactures/Je...	-0.009982	-0.010785	0.001290	-0.002210	0.012950	-0.014055	-0.009258	0.002966	-0.011494	...	0.002688
1	http://halal.addi.is.its.ac.id/manufactures/Am...	0.001167	-0.009434	0.000645	-0.001031	0.008766	-0.000232	-0.008129	0.005552	-0.006562	...	0.003313
2	http://halal.addi.is.its.ac.id/manufactures/St...	-0.002369	-0.013097	0.002923	-0.011869	0.007506	-0.008881	-0.008147	0.002153	-0.007135	...	0.009371
3	http://halal.addi.is.its.ac.id/manufactures/Li...	-0.001649	-0.005344	-0.003339	-0.005358	-0.001656	0.002563	-0.005313	-0.002229	0.001380	...	-0.000795
4	http://halal.addi.is.its.ac.id/manufactures/Yi...	0.002971	-0.004289	-0.001187	-0.002908	0.000447	-0.001315	-0.005677	-0.002215	-0.003805	...	0.005115
...	...	...	...	...	...	...	...	...	...	...	...	...
21232	http://halal.addi.is.its.ac.id/manufactures/Pf...	-0.015791	-0.016935	0.000231	-0.002209	0.015589	-0.019267	-0.013298	0.003206	-0.012608	...	0.008705
21233	http://halal.addi.is.its.ac.id/manufactures/Te...	0.004632	-0.003782	0.004087	-0.002548	0.013088	-0.005177	-0.008472	-0.003216	-0.009668	...	0.012949
21234	http://halal.addi.is.its.ac.id/manufactures/By...	0.004317	-0.004459	0.001513	-0.001362	0.008050	-0.010236	-0.001776	-0.002006	-0.006190	...	0.009008
21235	http://halal.addi.is.its.ac.id/manufactures/My...	-0.003378	-0.014259	0.000830	-0.001121	0.007841	-0.007457	-0.013431	0.004210	-0.013931	...	0.008928
21236	http://halal.addi.is.its.ac.id/manufactures/Mr...	0.000205	-0.003251	-0.002245	-0.006142	0.001564	0.004454	0.001447	-0.001798	-0.004124	...	0.001156

21237 rows × 101 columns

Gambar 5.10 Tampilan graph embedding RDF2Vec untuk link prediction

5.4 Link Prediction

5.4.1 Mengubah Bentuk Data Pasangan Produk

Data awal yang akan digunakan adalah data pasangan dari seluruh produk makanan yang ada di dalam *knowledge graph*. Data pasangan tersebut terdiri dari tiga buah kolom yang berisikan uri dari masing-masing produk makanan dan labelnya. Gambar 5.11 merupakan bentuk awal dari data yang digunakan:

	URIFrom	URITo	Labeling
0	http://halal.addi.is.its.ac.id/foodproducts/Du...	http://halal.addi.is.its.ac.id/foodproducts/In...	1
1	http://halal.addi.is.its.ac.id/foodproducts/En...	http://halal.addi.is.its.ac.id/foodproducts/En...	1
2	http://halal.addi.is.its.ac.id/foodproducts/Sa...	http://halal.addi.is.its.ac.id/foodproducts/Sa...	1
3	http://halal.addi.is.its.ac.id/foodproducts/Ge...	http://halal.addi.is.its.ac.id/foodproducts/Ch...	1
4	http://halal.addi.is.its.ac.id/foodproducts/Re...	http://halal.addi.is.its.ac.id/foodproducts/Re...	1
...	...	...	...
1284817	http://halal.addi.is.its.ac.id/foodproducts/En...	http://halal.addi.is.its.ac.id/foodproducts/Ai...	0
1284818	http://halal.addi.is.its.ac.id/foodproducts/En...	http://halal.addi.is.its.ac.id/foodproducts/In...	0
1284819	http://halal.addi.is.its.ac.id/foodproducts/En...	http://halal.addi.is.its.ac.id/foodproducts/Ad...	0
1284820	http://halal.addi.is.its.ac.id/foodproducts/En...	http://halal.addi.is.its.ac.id/foodproducts/Cl...	0
1284821	http://halal.addi.is.its.ac.id/foodproducts/En...	http://halal.addi.is.its.ac.id/foodproducts/Fr...	0

1284822 rows × 3 columns

Gambar 5.11 Tampilan data awal pasangan antar produk makanan

Data tersebut terdiri dari total 1280696 pasang produk makanan. Dikarenakan model *link prediction* tidak dapat dibuat dengan menggunakan fitur yang berbentuk string, maka fitur yang akan digunakan untuk melakukan *link prediction* adalah

vektor *embedding* dari masing-masing uri produk makanan menggunakan dua algoritma *graph embedding* yang telah digunakan, yaitu *Node2Vec* dan RDF2Vec. Untuk menambahkan vektor *embedding* kepada data sebelumnya, maka digunakan Kode 5.1 sebagai berikut:

```
merge1 = data.merge(embedding_df, left_on='URIFrom',
right_on='uri').merge(embedding_df, left_on='URITo',
right_on='uri')
merge1 = merge1.drop(columns=['URIFrom', 'URITo', 'uri_x',
'uri_y'])
merge2 = data.merge(embedding_df2, left_on='URIFrom',
right_on='uri').merge(embedding_df, left_on='URITo',
right_on='uri')
merge2 = merge2.drop(columns=['URIFrom', 'URITo', 'uri_x',
'uri_y'])
```

Kode 5.25 Menambahkan vektor embedding untuk tiap uri

Kode 5.25 digunakan untuk menambahkan nilai vektor *embedding* dari hasil *graph embedding Node2Vec* dan RDF2Vec terhadap data hubungan produk makanan sebelumnya. Gambar 5.12 berikut merupakan contoh bentuk data yang akan digunakan untuk membuat model *link prediction* menggunakan algoritma *Random Forest*:

	Labeling	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	...	91_y	92_y	93_y	94_y
0	1	-0.702162	-0.644228	-0.767668	0.035389	-0.858089	-0.472290	0.064183	-0.151259	0.150700	...	0.821805	-0.199322	0.483569	0.359144
1	0	-0.470965	0.172671	0.725902	0.471167	-0.896934	0.873560	-0.533920	0.128320	-0.029754	...	0.821805	-0.199322	0.483569	0.359144
2	0	0.021881	0.607069	-0.516492	-0.865557	-0.078580	-0.194024	-0.996002	0.901389	0.755355	...	0.821805	-0.199322	0.483569	0.359144
3	0	-0.614335	0.902126	0.126998	0.232395	-0.659679	0.784474	-0.916558	-0.571246	0.649799	...	0.821805	-0.199322	0.483569	0.359144
4	0	-0.130539	0.417967	0.106355	0.693106	0.626931	-0.746763	-0.595157	-0.170845	-0.039102	...	0.821805	-0.199322	0.483569	0.359144
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1284817	0	0.275532	-0.636493	0.461502	-0.705234	-0.435801	-0.020532	0.890056	0.982835	0.135627	...	-0.118412	-0.053654	-0.230298	0.400629
1284818	0	0.085827	0.411739	0.045779	-0.508399	-0.449099	0.587044	0.081373	0.976098	-0.805346	...	-0.118412	-0.053654	-0.230298	0.400629
1284819	0	0.931260	0.011991	0.037045	0.323726	-0.554913	-0.830184	-0.143044	0.067267	0.739429	...	-0.118412	-0.053654	-0.230298	0.400629
1284820	0	-0.692618	0.796701	-0.700307	0.757550	-0.215853	0.738217	0.592107	-0.391838	0.422672	...	-0.118412	-0.053654	-0.230298	0.400629
1284821	0	-0.400689	-0.034113	0.072616	0.681925	0.137334	0.737745	-0.713743	-0.209269	0.769076	...	-0.118412	-0.053654	-0.230298	0.400629

1284822 rows × 201 columns

Gambar 5.12 Tampilan data hasil merging dengan vektor embedding

5.4.2 Oversampling Dan Undersampling Data

Oversampling dan *undersampling* data dilakukan karena label target yang digunakan memiliki jumlah yang tidak seimbang. Label target 1 yang merupakan kelas minoritas pada penelitian ini hanya berjumlah 0.1% dari jumlah keseluruhan data yang digunakan. Sehingga dibutuhkan *oversampling* untuk menambah jumlah data dari kelas minoritas dan dibutuhkan pula *undersampling* mengurangi jumlah data dari kelas mayoritas.

Oversampling dan *Undersampling* ini dilakukan dengan menggunakan *SMOTE* dan *RandomUnderSampler* dari *library imblearn*. Kode 5.26 berikut merupakan baris kode yang digunakan untuk melakukan *oversampling* dan *undersampling* data:

```
Ss{1, 0.5, 0.33}
over = SMOTE(sampling_strategy=0.002)
under = RandomUnderSampler(sampling_strategy=1)
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)
X, y = pipeline.fit_resample(X, y)
```

Kode 5.26 Melakukan oversampling dan undersampling data

Berdasarkan Kode 5.26 di atas, *oversampling* akan dilakukan dengan membuat jumlah kelas minoritas yaitu label 1 berjumlah tidak lebih dari dua kali lipat jumlah kelas pada awalnya. Maka dari itu digunakan parameter *sampling strategy* pada *SMOTE* yaitu 0.005, yang berarti mengubah jumlah kelas minoritas menjadi sejumlah 0.005 kali dari jumlah kelas mayoritas awal. Selanjutnya digunakan parameter *sampling strategy* untuk *RandomUnderSampler* yaitu 1, 0.5, dan 0.33. Masing-masing *sampling strategy* dari *RandonUnderSampler* tujuannya adalah untuk menentukan berapa banyak jumlah kelas mayoritas yang akan dibentuk dari hasil undersampling. Jika *sampling strategy* yang digunakan adalah 1, maka perbandingan antara label 0 dan label 1 adalah 1:1, jika *sampling strategy* yang digunakan

adalah 0.5, maka perbandingan antara label 0 dan label 1 adalah 1:2, dan jika *sampling strategy* yang digunakan adalah 0.33, maka perbandingan antara label 0 dan label 1 adalah 1:3.

5.4.3 Melakukan Tuning Parameter

Penentuan nilai parameter yang digunakan untuk menemukan parameter yang menghasilkan *F1-Score* terbaik akan dilakukan dengan menggunakan *GridSearchCV* dari *library sklearn*. Parameter yang akan diuji cobakan dilakukan sesuai dengan *parameter grid* yang telah dijelaskan sebelumnya pada sub bab 4.4.3, yaitu dengan mengatur parameter *max_depth*, *min_samples_leaf*, *min_samples_split*, *n_estimatros*, dan *criterion*.

Tuning parameter akan dilakukan sebanyak dua kali, yaitu dengan melakukan *tuning* parameter menggunakan data hasil *graph embedding Node2Vec* dan data hasil *graph embedding* RDF2Vec. Untuk menentukan model dengan parameter yang terbaik, akan digunakan rata-rata nilai *F1-Score* dari hasil metode *5-fold cross validation* untuk pengukuran performanya. Kode 5.27 berikut merupakan baris kode yang digunakan untuk melakukan *tuning* parameter secara otomatis:

```
param_grid = {
    'bootstrap': [True],
    'max_features' : ['auto', 'sqrt'],
    'max_depth': [80, 90, 100, 110],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000],
    'criterion': ['gini', 'entropy']
}
rf = RandomForestClassifier()

grid_search = GridSearchCV(estimator = rf, param_grid =
param_grid, scoring='f1', cv = 5, n_jobs = -1)
```

Kode 5.27 Melakukan tuning hyperparameter

5.4.4 Link Prediction Menggunakan Random Forest

Pembuatan model *link prediction* dengan algoritma *Random Forest* dilakukan dengan menggunakan bahasa python dengan beberapa *library* seperti *pandas* dan *numpy* untuk pengolahan data, *sklearn* untuk membagi dataset serta membuat model *link prediction* beserta evaluasi performanya, serta *imblearn* untuk melakukan *oversampling* dan *undersampling* pada dataset awal. Semua *library* yang digunakan dipanggil seperti yang terdapat pada Kode 5.28 berikut:

```
import csv
import numpy as np
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
from sklearn import svm, linear_model, neighbors
from sklearn import tree, ensemble
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
from sklearn import model_selection
import imblearn
```

Kode 5.28 Library yang digunakan untuk melakukan link prediction

Untuk melakukan *link prediction*, digunakan *library sklearn* dimana *default* parameter dari *Random Forest* terdapat seperti pada Tabel 5.3 berikut:

Tabel 5.3 Parameter default Random Forest

Parameter	Value
N_estimators	100
Criterion	Gini
Max_depth	None
Min_samples_split	2
Min_samples_leaf	1
Max_features	Auto
Bootstrap	True

Link prediction menggunakan algoritma *Random Forest* dilakukan dengan menggunakan percobaan parameter *default* dan parameter hasil *tuning* menggunakan *GridSearchCV*. Kode 5.29 merupakan baris kode yang digunakan untuk membuat model *link prediction* dengan parameter *default* seperti pada Tabel 5.3

```
rf = RandomForestClassifier()
```

Kode 5.29 Membuat model random forest dengan default parameter

Kode 5.30 dan Kode 5.31 merupakan baris kode yang digunakan untuk membuat model *link prediction* menggunakan algoritma *Random Forest* dengan parameter hasil *tuning* menggunakan *GridSearchCV* untuk masing-masing algoritma *graph embedding*-nya:

```
rf = RandomForestClassifier(n_jobs = -1,
                            max_depth = 80, min_samples_leaf = 1,
                            min_samples_split = 2, n_estimators = 1000,
criterion='gini')
```

Kode 5.30 Membuat model random forest hasil tuning Node2Vec

```
rf = RandomForestClassifier(n_jobs = -1,
                    max_depth = 90, min_samples_leaf = 1,
                    min_samples_split = 2, n_estimators = 1000,
criterion='entropy')
```

Kode 5.31 Membuat model random forest hasil tuning RDF2Vec

Dari model yang telah didapatkan, selanjutnya dilakukan pengukuran *cross validation* dengan menggunakan metode *5-fold cross validation*. Kodee 5.32 merupakan baris kode yang digunakan untuk melakukan *5-fold cross validation* berdasarkan pembagian *train set* dan *test set* yang telah ditentukan sebelumnya:

```
X = merge1.loc[:, merge1.columns != 'Labeling']
y = merge1.loc[:, merge1.columns == 'Labeling']
y=y['Labeling']

over = SMOTE(sampling_strategy=0.002)
under = RandomUnderSampler(sampling_strategy=1)
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)

X, y = pipeline.fit_resample(X, y)

clfs = RandomForestClassifier()

print(cross_val_score(clfs, X, y, cv=kfold, scoring="f1").mean())
```

Kode 5.32 Melakukan 5-fold cross validation

Setelah model selesai dibuat, hasil link prediction diubah ke bentuk RDF untuk melengkapi informasi pada file RDF sebelumnya. Kode 5.33 digunakan untuk membuat property halalf:similarTo pada file RDF baru:

```
def createProductTTLSusulan(row, path):
    productString = """halalf:{slugName} a
foodlirmm:FoodProduct;
            halalf:similarTo halalf:{slugSimilar} .

"""
```

Kode 5.33 Membuat property halalf:similarTo

5.5 Pengukuran Performa

Hasil prediksi untuk setiap model yang dibuat akan dianalisis dengan menghitung nilai akurasi, *precision*, *recall*, dan *f1-score*. Kode 5.34 merupakan baris kode yang digunakan untuk menghitung nilai tiap metriks yang telah ditentukan sebelumnya dengan bantuan *package metrics* dari *library sklearn*:

```
score_test1 = metrics.f1_score(y_test, predict)
score_test2 = metrics.precision_score(y_test, predict)
score_test3 = metrics.recall_score(y_test, predict)
score_test4 = metrics.accuracy_score(y_test, predict)
```

Kode 5.34 Mendapatkan nilai tiap metriks

Untuk model terbaik yang dihasilkan dari masing-masing algoritma *graph embedding* akan dibuatkan visualisasi kurva ROC-nya. Kode 5.35 digunakan untuk membuat visualisasi ROC hasil dari model Random Forest:

```
rf_probs = RF.predict_proba(X_test)
rf_probs = rf_probs[:, 1]

rf_auc = roc_auc_score(y_test,rf_probs)
print('Random Forest = %.3f' % (rf_auc))

rf_rpr, rf_tpr, _ = roc_curve(y_test, rf_probs)

import matplotlib.pyplot as plt
plt.plot(rf_rpr, rf_tpr, marker= '.', label = 'Random Forest
(AUROC = %0.3f)' % rf_auc)
plt.title('ROC Plot')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

Kode 5.35 Membuat grafik ROC

Untuk membuat perbandingan hasil performa dari model *link prediction* menggunakan *Random Forest* dengan algoritma *machine learning* lainnya, maka dibuat beberapa model *link prediction* menggunakan algoritma lainnya seperti yang telah ditentukan sebelumnya. Masing-masing model dengan algoritma lain tersebut akan dibuat menggunakan parameter *default*-nya. Kode 5.36 merupakan baris kode yang digunakan untuk membuat model *link prediction* menggunakan algoritma *machine learning* yang lainnya tersebut:

```
NB = GaussianNB()
LR = linear_model.LogisticRegression()
GBT = ensemble.GradientBoostingClassifier()
```

Kode 5.36 Membuat model machine learning lain dengan parameter default

Hasil *link prediction* selanjutnya akan divisualisasikan ke bentuk *graph*. Visualisasi akan dibuat dengan menggunakan

library pyvis. Kode 5.37 digunakan untuk membuat visualisasi hasil link prediction.

```
import pyvis
from pyvis.network import Network
net = Network(notebook=True)
net.from_nx(g)
net.show('example.html')
```

Kode 5.37 Membuat visualisasi graph dari link prediction

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini akan berisikan tentang hasil dari penelitian beserta pembahasan dari implementasi yang sudah dilakukan sesuai dengan metode penelitian. Hasil yang akan dibahas pada bab ini adalah mengenai *oversampling* dan *undersampling* data menggunakan *imblearn*, menentukan parameter *Random Forest* terbaik, serta performa model *link prediction* menggunakan nilai *accuracy*, *precision*, *recall*, dan *F1-Score*-nya.

6.1 Hasil Oversampling dan Undersampling

Implementasi metode *oversampling* dan undersampling dilakukan dengan menggunakan *library imblearn*. Metode ini memiliki konsep yaitu membuat menambah data dari kelas minoritas hingga berjumlah sekian persen dari jumlah kelas mayoritas awal. Selanjutnya, diambil beberapa sampel dari kelas mayoritas hingga jumlah kelas mayoritasnya berukuran beberapa kali lipat dari jumlah kelas minoritas yang telah dilakukan *oversampling*.

Untuk *oversampling* akan dilakukan dengan membuat jumlah kelas minoritas yaitu label 1 berjumlah tidak lebih dari dua kali lipat jumlah kelas pada awalnya. Pada penelitian ini *oversampling* akan dilakukan dengan menggunakan *sampling strategy* 0.002 dengan beberapa perbandingan jumlah label. Pada Tabel 6.1 merupakan perbandingan jumlah data berdasarkan perbandingan label 0 (tidak mirip) dan label 1 (mirip) yang digunakan:

Tabel 6.1 Perbandingan jumlah data tiap label

Label	**Jumlah Awal**	**Jumlah Ukuran Dengan Perbandingan 1:1**	**Jumlah Ukuran Dengan Perbandingan 2:1**	**Jumlah Ukuran Dengan Perbandingan 3:1**
Tidak Mirip	1.283.490	2.566	5.132	7.698
Mirip	1.332	2.566	2.566	2.566
Total	1.284.822	6.132	7.698	10.264

Berikut merupakan performa model *link prediction* yang dihasilkan berdasarkan rencana *oversampling* dan *undersampling* yang telah dijelaskan sebelumnya. Pembagian *train set* dan *test set* yang digunakan pada model yang dibuat adalah 70:30. Tabel 6.2 dan Tabel 6.3 menampilkan performa model dari masing-masing algoritma *graph embedding* yang digunakan:

Tabel 6.2 Nilai model dengan embedding Node2Vec

Label0:Label1	**F1-Score**	**Precision**	**Recall**	**Accuracy**
1:1	0.823	0.858	0.791	0.831
2:1	0.723	0.926	0.593	0.849
3:1	0.652	0.971	0.491	0.872

Tabel 6.3 Nilai model dengan embedding RDF2Vec

Label0:Label1	**F1-Score**	**Precision**	**Recall**	**Accuracy**
1:1	0.794	0.805	0.783	0.798
2:1	0.683	0.891	0.554	0.830
3:1	0.624	0.939	0.467	0.862

Berdasarkan hasil performa model *link prediction* dari kedua *graph embedding* yang digunakan, nilai tertinggi dari *F1-Score* diperoleh ketika perbandingan jumlah data pada label yang digunakan adalah 1:1. Namun, nilai tertinggi dari akurasi diperoleh ketika perbandingan jumlah data label yang digunakan adalah 3:1.

Maka dari itu, perbandingan yang akan digunakan untuk model-model selanjutnya adalah menggunakan perbandingan jumlah data label 1:1. Hal ini dikarenakan pada perbandingan 1:1, meskipun nilai akurasinya sedikit lebih kecil, namun nilai *F1-Score* yang didapatkan lebih tinggi serta nilai *precision* dan *recall* yang dihasilkan dapat dikatakan cukup seimbang. Sementara ketika perbandingan yang digunakan adalah 2:1 atau 3:1, meskipun nilai akurasinya semakin tinggi, namun nilai *recall* dikorbankan untuk nilai *precision* yang lebih tinggi. Nilai *recall* yang semakin rendah menandakan sedikitnya data dari label 1 yang berhasil diprediksi benar label targetnya.

Seperti yang telah dijelaskan sebelumnya, oversampling dilakukan dengan menggunakan *sampling strategy* sebesar 0.002 sehingga data sintetis yang dihasilkan jumlahnya tidak melebih data asli yang ada. Namun ketika data sintetis yang dihasilkan untuk label 1 dibuat lebih banyak, performa dari model *link prediction* cenderung akan lebih baik. Ketika *sampling strategy* yang digunakan adalah 0.005 dengan perbandingan label 1:1, maka akan menghasilkan perbandingan jumlah data seperti pada Tabel 6.4 berikut:

Tabel 6.4 Perbandingan jumlah data tiap label

	Jumlah Awal	**Jumlah dengan SS 0.002**	**Jumlah dengan SS 0.005**
Label 0	1.283.490	2.566	6.417
Label 1	1.332	2.566	6.417
Total	1.284.822	6.132	12.843

Berdasarkan jumlah data hasil *oversampling* dan *undersampling* tersebut, Tabel 6.5 menampilkan hasil nilai performa dari model *link prediction* yang digunakan:

Tabel 6.5 Nilai model link prediction dengan SS 0.005

	F1-Score	**Precision**	**Recall**	**Accuracy**
Node2Vec	0.934	0.946	0.923	0.936
RDF2Vec	0.928	0.933	0.922	0.929

Berdasarkan hasil pengukuran performa tersebut, didapati bahwa nilai perorma dari tiap metriks yang dihasilkan lebih besar dibandingkan nilai yang didapatkan menggunakan *sampling strategy* 0.002. Namun, pada penelitian kali ini model akan tetap dibuat menggunakan *sampling strategy* 0.002 dikarenakan pada *sampling strategy* 0.005, jumlah data sintetis untuk label 1 bernilai hingga lebih dari 3 kali lipat data aslinya. Efeknya adalah ketika nantinya dilakukan visualisasi keterhubungan antar produk makanannya, akan banyak data ber label 1 yang hilang dikarenakan data tersebut bukan data riil dan tidak merepresentasikan *node* produk makanan yang ada di dalam *knowledge graph* sesungguhnya.A

6.2 Hasil Tuning Parameter

Pencarian nilai parameter terbaik untuk model *link prediction* dilakukan menggunakan *GridSearchCV*. Parameter tersebut akan dicari dengan membangun model dari tiap kemungkinan kombinasi parameter yang telah ditentukan sesuai bab 4.4.3. Parameter yang akan digunakan adalah parameter yang menghasilkan nilai *F1-Score* dari metode *5-fold cross validation* tertinggi dari masing-masing *graph embedding*.

6.2.1 Tuning Parameter Embedding Node2Vec

Digunakan vektor *embedding* yang dihasilkan oleh Node2Vec untuk mencari parameter yang dapat menghasilkan nilai *F1-Score* terbaik. Tabel 6.6 berikut merupakan parameter yang menghasilkan nilai *F1-Score* terbaik berdasarkan

GridSearchCV dengan menggunakan vektor *embedding* dari *Node2Vec*:

Tabel 6.6 Parameter hasil tuning untuk Node2Vec

Parameter	Value
Bootstrap	True
Criterion	Gini
Max_depth	80
Max_features	Auto
Min_samples_leaf	1
Min_samples_split	2
N_estimators	1000

Selanjutnya dibuat model *link prediction* menggunakan parameter hasil *GridSearchCV* sesuai pada Tabel 6.6 Parameter hasil tuning untuk Node2Vec di atas. Model *link prediction* dibuat masing-masing menggunakan vektor *embedding* dari Node2Vec dan RDF2Vec. Tabel 6.7 berikut merupakan perbandingan nilai *F1-Score* yang dihasilkan dari masing-masing model:

Tabel 6.7 Nilai f1-score dengan hasil parameter tuning Node2Vec

Graph Embedder	F1-Score
Node2Vec	0.8257
RDF2Vec	0.8063

6.2.2 Tuning Parameter Embedding RDF2Vec

Digunakan vektor *embedding* yang dihasilkan oleh RDF2Vec untuk mencari parameter yang dapat menghasilkan nilai *F1-Score* terbaik. Tabel 6.8 berikut merupakan parameter yang menghasilkan nilai *F1-Score* terbaik berdasarkan

GridSearchCV dengan menggunakan vektor *embedding* dari RDFVec:

Tabel 6.8 Parameter hasil tuning untuk RDF2Vec

Parameter	Value
Bootstrap	True
Criterion	Entropy
Max_depth	90
Max_features	Auto
Min_samples_leaf	1
Min_samples_split	2
N_estimators	1000

Selanjutnya dibuat model *link prediction* menggunakan parameter hasil *GridSearchCV* sesuai pada Tabel 6.8 Parameter hasil tuning untuk RDF2Vec di atas. Model *link prediction* dibuat masing-masing menggunakan vektor *embedding* dari RDF2Vec dan RDF2Vec. Tabel 6.9 berikut merupakan perbandingan nilai *F1-Score* yang dihasilkan dari masing-masing model:

Tabel 6.9 Nilai f1-score dengan hasil parameter tuning RDF2Vec

Graph Embedder	F1-Score
Node2Vec	0.8238
RDF2Vec	0.8093

Berdasarkan data yang didapatkan dari pengujian model *link prediction* menggunakan parameter terbaik dari masing-masing *graph embedding*, didapatkan bahwa masing-masing *graph embedding* memiliki parameter terbaik yang berbeda. Untuk model yang dibangun dengan vektor *embedding* dari Node2Vec, diperoleh nilai *F1-Score* yang lebih besar pada

model dengan parameter yang pertama yaitu 0.8257 dibandingkan dengan parameter yang kedua yaitu 0.8238. Sama halnya dengan model yang dibangun dengan vektor *embedding* dari RDF2Vec, diperoleh nilai *F1-Score* yang lebih besar pada model dengan parameter yang kedua yaitu 0.8073 dibandingkan dengan parameter yang pertama yaitu 0.8063.

Meskipun masing-masing model *link prediction* memiliki parameter terbaik yang berbeda berdasarkan *graph embedding*-nya, nilai *F1-Score* yang dihasilkan tidak jauh berbeda. Untuk model dengan *embedding* dari Node2Vec hanya memiliki perbedaan nilai *F1-Score* sebesar 0.0019 dari kedua parameter yang digunakan, sedangkan model dengan *embedding* dari RDF2Vec hanya memiliki perbedaan nilai *F1-Score* sebesar 0.0030 dari kedua parameter yang digunakan.

6.3 Hasil Perbandingan Model Link Prediction

Berdasarkan hasil dan pembahasan pada 6.2, model *link prediction* akan dibangun dengan menggunakan *sampling strategy* sebesar 0.002 dengan perbandingan jumlah data pada label sebesar 1:1. Maka selanjutnya akan dilakukan perbandingan model *link prediction* yang menggunakan parameter *default* dengan model yang dibangun menggunakan parameter hasil *tuning*. Selain itu, akan digunakan kombinasi *train set* dan *test set* yang beragam untuk menemukan kombinasi yang dapat memberikan performa yang paling baik. Hasil performa dari berbagai macam model *link prediction* tersebut akan dibandingkan satu sama lainnya dengan menggunakan nilai rata-rata *F1-Score* dari hasil *metode 5-fold cross validation*.

Tabel *6.10* berikut menunjukkan jumlah data pada *train set* dan *test set* untuk tiap kombinasinya:

Tabel 6.10 Pembagian jumlah data train set dan test set

	Kombinasi (Train Set : Test Set)			
	0.8 : 0.2	**0.75 : 0.25**	**0.7 : 0.3**	**0.6 : 0.4**
Train Set	4.906	4.599	4.292	3.679
Test Set	1.226	1.533	1.840	2.453
Total	6.132	6.132	6.132	6.132

6.3.1 Model Dengan Parameter Default

Model *link prediction* dibuat meggunakan algoritma *Random Forest* menggunakan parameter *default* yang telah dijelaskan pada sub bab 4.4.3. Model akan dibuat menggunakan berbagai macam kombinasi ukuran *train set* dan *test set*, serta juga menggunakan beberapa perbandingan jumlah label 0 dan label 1 untuk memberikan informasi tambahan mengenai performa dari model-model *link prediction* yang dibangun.

a. Node2Vec
   Tabel 6.11 berikut menunjukkan nilai *F1-Score* dari model *link prediction* yang dibangun dengan parameter *default* dan kombinasi *train set* dan *test set* sebesar 1:1 menggunakan vektor *embedding* hasil dari *Node2Vec*:

Tabel 6.11 Nilai f1-score model default dengan embedding Node2Vec

Per-cobaan	**Ukuran Train Set**			
	0.8	**0.75**	**0.7**	**0.6**
1	0.818	0.822	0.818	0.817
2	0.822	0.827	0.815	0.821
3	0.822	0.815	0.823	0.812
4	0.816	0.83	0.823	0.813
5	0.816	0.814	0.823	0.818
Mean	**0.818**	**0.821**	**0.820**	**0.816**

b. RDF2Vec

Tabel 6.12 berikut menunjukkan nilai *F1-Score* dari model *link prediction* yang dibangun dengan parameter *default* dan kombinasi *train set* dan *test set* sebesar 1:1 menggunakan vektor *embedding* hasil dari RDF2Vec:

Tabel 6.12 Nilai f1-score model default dengan embedding RDF2Vec

Per-cobaan	**Ukuran Train Set**			
	0.8	**0.75**	**0.7**	**0.6**
1	0.798	0.797	0.792	0.807
2	0.8	0.801	0.789	0.802
3	0.807	0.802	0.791	0.793
4	0.801	0.803	0.797	0.803
5	0.799	0.794	0.801	0.8
Mean	**0.801**	**0.799**	**0.794**	**0.8**

Berdasarkan Tabel 6.11 dan Tabel 6.12 di atas, didapatkan bahwa model *link prediction* yang memiliki nilai *f1-score* terbaik dengan menggunakan vektor *embedding* Node2Vec dihasilkan dengan menggunakan kombinasi *train set* dan *test set* sebesar 0.75:0.25 dengan nilai *f1-score* sebesar 0,821. Sedangkan model *link prediction* yang memiliki nilai *f1-score* terbaik dengan menggunakan vektor *embedding* RDF2Vec dihasilkan dengan menggunakan kombinasi *train set* dan *test set* sebesar 0.8:0.2 dengan nilai *f1-score* sebesar 0.801.

Dengan menggunakan parameter *default*, model yang menggunakan vektor *embedding* Node2Vec memiliki nilai *f1-score* yang lebih besar dibandingkan model yang menggunakan vektor *embedding* RDF2Vec. Perbedaan antar nilai terbaik dari kedua model tersebut adalah sebesar 0.02 atau 2%. Selain itu, nilai *f1-score* yang diperoleh pada masing kombinasi *train set* dan *test set* hanya terpaut maksimal sebesar 0.005 atau 0.5%

pada model NodeVec dan 0.007 atau 0.7% pada model RDF2Vec.

6.3.2 Model Dengan Parameter Tuning

a. Node2Vec
Tabel 6.13 berikut menunjukkan nilai *F1-Score* dari model *link prediction* yang dibangun dengan parameter hasil *tuning* dan kombinasi *train set* dan *test set* sebesar 1:1 menggunakan vektor *embedding* hasil dari *Node2Vec*:

Tabel 6.13 Nilai f1-score model hasil tuning dengan embedding Node2Vec

Per-cobaan	**Ukuran Train Set**			
	0.8	**0.75**	**0.7**	**0.6**
1	0.833	0.822	0.832	0.826
2	0.83	0.829	0.835	0.82
3	0.823	0.825	0.829	0.832
4	0.831	0.828	0.826	0.829
5	0.821	0.827	0.831	0.828
Mean	**0.827**	**0.826**	**0.830**	**0.827**

b. RDF2Vec
Tabel 6.14 berikut menunjukkan nilai *F1-Score* dari model *link prediction* yang dibangun dengan parameter hasil *tuning* dan kombinasi *train set* dan *test set* sebesar 1:1 menggunakan vektor *embedding* hasil dari *Node2Vec*:

Tabel 6.14 Nilai f1-score model hasil tuning dengan embedding RDF2Vec

Per-cobaan	Ukuran Train Set			
	0.8	**0.75**	**0.7**	**0.6**
1	0.799	0.806	0.804	0.812
2	0.812	0.810	0.814	0.806
3	0.803	0.810	0.809	0.807
4	0.802	0.818	0.809	0.805
5	0.812	0.804	0.804	0.808
Mean	**0.805**	**0.807**	**0.808**	**0.807**

Berdasarkan Tabel 6.13 dan Tabel 6.14 di atas, didapatkan bahwa model *link prediction* yang memiliki nilai *f1-score* terbaik dengan menggunakan vektor *embedding* Node2Vec dihasilkan dengan menggunakan kombinasi *train set* dan *test set* sebesar 0.7:0.3 dengan nilai *f1-score* sebesar 0,830. Sedangkan model *link prediction* yang memiliki nilai *f1-score* terbaik dengan menggunakan vektor *embedding* RDF2Vec dihasilkan dengan menggunakan kombinasi *train set* dan *test set* sebesar 0.7:0.3 dengan nilai *f1-score* sebesar 0.808.

Dengan menggunakan parameter hasil *tuning*, model yang menggunakan vektor *embedding* Node2Vec memiliki nilai *f1-score* yang lebih besar dibandingkan model yang menggunakan vektor *embedding* RDF2Vec. Perbedaan antar nilai terbaik dari kedua model tersebut adalah sebesar 0.022 atau 2.2%. Selain itu, nilai *f1-score* yang diperoleh pada masing kombinasi *train set* dan *test set* hanya terpaut maksimal sebesar 0.004 atau 0.4% pada model NodeVec dan 0.003 atau 0.3% pada model RDF2Vec.

Dari hasil perbandingan parameter pada berbagai macam model *link prediction* yang telah dibangun, model *link prediction* yang menggunakan parameter hasil *tuning* memiliki nilai *f1-score* yang lebih baik dibandingkan model yang menggunakan

parameter *default*. Meskipun nilai *f1-score*-nya lebih baik, namun perbedaan nilai *f1-score* antara kedua model tersebut tidak signifikan. Untuk model Node2Vec, nilai *f1-score* yang didapat dengan model hasil *tuning* bernilai lebih besar 0.009 atau 0.9% dibandingkan dengan model *default*. Untuk model RDF2Vec, nilai *f1-score* yang didapat dengan model hasil *tuning* bernilai lebih besar 0.007 atau 0.7% dibandingkan dengan model *default*. Dari kedua model tersebut, perbedaan hasil nilai *f1-score* nya dari tiap komposisi *train set* dan *test set* nya tidak ada yang mencapai angka 1%.

Untuk perbandingan antar algoritma *graph embedding* yang digunakan, model *link prediction* yang menggunakan vektor *embedding* dari Node2Vec memiliki nilai rata-rata *f1-score* yang lebih baik. Pada model terbaiknya, model dengan Node2Vec memperoleh nilai *f1-score* sebesar 0.830, sedangkan model dengan RDF2Vec hanya memperoleh nilai *f1-score* sebesar 0.808. Dari kedua model tersebut, perbedaan nilai *f1-score* yang didapat adalah sebesar 0.022 atau 2,2%.

6.3.3 Model Dengan Performa Terbaik

Berdasarkan dengan hasil performa tiap model yang telah diperoleh, didapatkan model terbaik untuk masing-masing *graph embedder* yang digunakan. Selanjutnya model terbaik tersebut diukur kembali performanya dengan menggunakan *5-fold cross validation*. Tabel 6.15 dan Tabel 6.16 berikut menunjukkan nilai masing-masing metriks pada tiap modelnya:

a. Node2Vec

Tabel 6.15 Performa model Node2Vec dengan parameter terbaik

Percobaan	Accuracy	Precision	Recall	F1-Score
1	84.8%	87.7%	80.2%	83.6%
2	84.7%	85.3%	79.8%	83.3%
3	84%	89.7%	77.8%	83.6%
4	82.3%	85.4%	77.3%	81.8%
5	82.6%	88.2%	76.5%	82.6%
Mean	**83.4%**	**87.2%**	**78.9%**	**82.7%**

b. RDF2Vec

Tabel 6.16 Performa model RDF2Vec dengan parameter terbaik

Percobaan	Accuracy	Precision	Recall	F1-Score
1	80.1%	79.2%	81.8%	80.2%
2	81.3%	78.8%	82.8%	80.4%
3	79.3%	82.3%	77%	79.8%
4	80.3%	79.6%	80.9%	79.2%
5	79.6%	83.4%	75.4%	80.1%
Mean	**80%**	**80.5%**	**79.7%**	**79.7%**

Berdasarkan hasil pengukuran performa dari model *link prediction* terbaik dari masing-masing *graph embedder* pada Tabel 6.15 dan tabel Tabel 6.16 di atas, didapatkan bahwa secara keseluruhan performa model *link prediction* yang menggunakan Node2Vec mendapatkan nilai yang lebih besar untuk setiap metriksnya. Untuk rata-rata *f1-score* yang diperoleh model Node2Vec mendapatkan nilai 3% lebih besar dibandingkan model RDF2Vec. Untuk rata-rata *accuracy* yang diperoleh model Node2Vec mendapatkan nilai 3.4% lebih besar dibandingkan model RDF2Vec. Untuk rata-rata *precision* yang

diperoleh model Node2Vec mendapatkan nilai 6.7% lebih besar dibandingkan model RDF2Vec. Dan Untuk rata-rata *recall* yang diperoleh model Node2Vec mendapatkan nilai 0.8% lebih kecil dibandingkan model RDF2Vec.

6.4 Hasil Perbandingan Dengan Model Lain

Dilakukan perbandingan model *link prediction* yang dibuat menggunakan algoritma *Random Forest* dengan model *link prediction* yang dibuat menggunakan beberapa algoritma *machine learning* lainnya, yaitu *Gaussian Naïve Bayes*, *Logistic Regression*, dan *Gradient Boosting Classifier*.

Pada perbandingan ini, model dibuat dengan menggunakan vektor embedding dari kedua algoritma graph embedding. Tabel 6.17 berikut merupakan hasil pengukuran performa dari masing-masing link prediction yang dibuat:

Tabel 6.17 Perbandingan performa algoritma machine learning

Model	Graph Embedder	F1-Score	Accuracy	Precision	Recall
Random Forest	Node2Vec	83.45%	83.96%	85.34%	81.65%
Random Forest	RDF2Vec	80.99%	81.03%	80.46%	81.52%
Gaussian Naïve Bayes	Node2Vec	68.75%	70.84%	73.294%	64.74%
Gaussian Naïve Bayes	RDF2Vec	58.94%	55.13%	52.2%	65%
Logistic Regression	Node2Vec	60.81%	60.26%	59.44%	62.25%
Logistic Regression	RDF2Vec	55.5%	52.2%	51.51%	60.15%
Gradient Boosting Classifier	Node2Vec	76.96%	75.77%	72.78%	81.65%
Gradient Bossting Classifier	RDF2Vec	73.26%	72.27%	70.14%	76.67%

Berdasarkan hasil perbandingan nilai performa pada Tabel 6.17 di atas, didapatkan bahwa model *link prediction* menggunakan algoritma *Random Forest* dengan vektor embedding Node2Vec secara keseluruhan memiliki performa yang terbaik dibandingkan dengan algoritma *machine learning* lainnya yang menggunakan vektor embedding Node2Vec. Berdasarkan nilai *f1-score* dan *accuracy* yang diperoleh, algoritma yang terbaik secara berurutan adalah *Random Forest*, *Gradient Boosting*

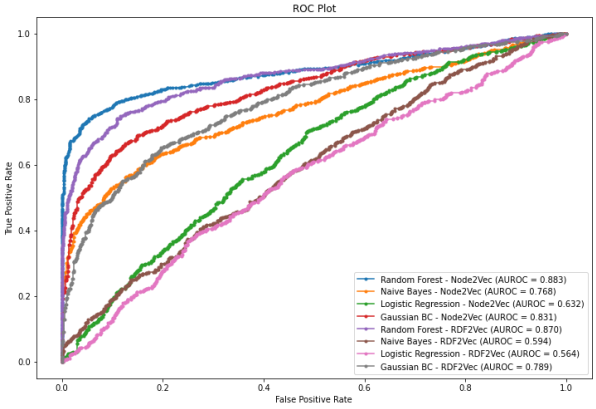
Classifier, *Gaussian Naïve Bayes*, dan terakhir *Logistic Regression*.

Selain itu, didapatkan bahwa model *link prediction* menggunakan algoritma *Random Forest* dengan vektor embedding RDF2Vec secara keseluruhan juga memiliki performa yang terbaik dibandingkan dengan algoritma *machine learning* lainnya yang menggunakan vektor embedding Node2Vec. Berdasarkan nilai *f1-score* dan *accuracy* yang diperoleh, algoritma yang terbaik secara berurutan adalah *Random Forest*, *Gradient Boosting Classifier*, *Gaussian Naïve Bayes*, dan terakhir *Logistic Regression*.

Berdasarkan perbandingan model antar graph embeddernya, model yang menggunakan embedding Node2Vec secara umum memiliki nilai *f1-score*, *accuracy*, *precision*, dan *recall* yang lebih baik dibandingkan model yang menggunakan embedding RDF2Vec. Namun, untuk perbedaannya ada pada nilai precision dan recall yang dimiliki. Model dengan Node2Vec cenderung mendapatkan nilai precision yang lebih besar dibandingkan recall, sedangkan model dengan RDF2Vec mendapatkan nilai recall yang lebih besar.

Pada

Gambar *6.1* berikut, ditampilkan hasil grafik ROC dari masing-masing model link prediction yang dibuat:



Gambar 6.1 Perbandingan grafik ROC tiap model

Dari masing-masing model tersebut, model Random Forest dengan menggunakan embedding Node2Vec memiliki nilai ROC tertinggi sebesar 0.883, sedangkan model Naïve Bayes dengan embedding RDF2Vec memiliki nilai ROC terendah sebesar 0.594. Untuk perbedaan nilai ROC yang paling besar dari algoritma machine learning yang sama dimiliki oleh Naïve Bayes, dimana model Naïve Bayes dengan embedding Node2Vec memiliki nilai ROC 0.768 dan model Naïve Bayes dengan embedding RDF2Vec memiliki nilai 0.594. Perbedaan yang didapatkan dari kedua model tersebut adalah 0.174.

Gambar 6.2 merupakan tampilan hasil file RDF yang telah ditambahkan properti *halalf:similarTo* untuk tiap pasangan produk yang diprediksi saling berhubungan menggunakan model link prediction:

```
halalf:Abc_Kopi_Instant_Kopi_Gula_Susu a foodlirmm:FoodProduct;
            halalf:similarTo halalf:Luwak_White_Koffie_Tarik_Malaka .


halalf:Abc_Kopi_Susu_Coklat_Mocca a foodlirmm:FoodProduct;
            halalf:similarTo halalf:Abc_Kopi_Instant_Kopi_Gula_Susu .


halalf:Adem_Sari_Ching_Ku_Herbal_Tea a foodlirmm:FoodProduct;
            halalf:similarTo halalf:Kraft_Sandwich_Oreo_Coklat_Vanila .


halalf:Adem_Sari_Minuman_Penyejuk a foodlirmm:FoodProduct;
            halalf:similarTo halalf:Monde_Cookies_Serena_Egg_Roll_Original .


halalf:Ades_Air_Mineral a foodlirmm:FoodProduct;
            halalf:similarTo halalf:Kayarasa_Jasela_Instant .


halalf:Aik_Cheong_Cafe_Art_Matcha a foodlirmm:FoodProduct;
            halalf:similarTo halalf:Aik_Cheong_White_Coffee_Original .


halalf:Aik_Cheong_Coffeemix_Reguler a foodlirmm:FoodProduct;
            halalf:similarTo halalf:Aik_Cheong_White_Coffee_Original .
```

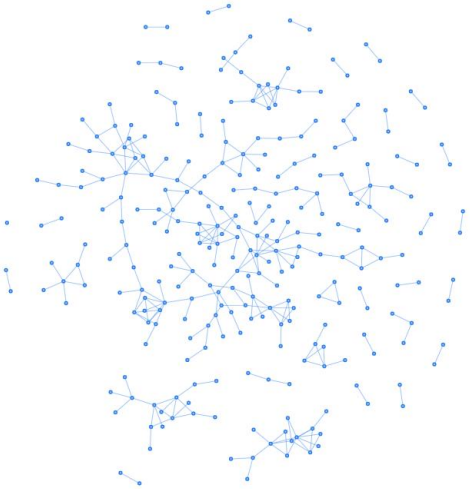
Gambar 6.2 Penambahan property halalf:similarTo

6.5 Visualisasi Hasil Link Prediction

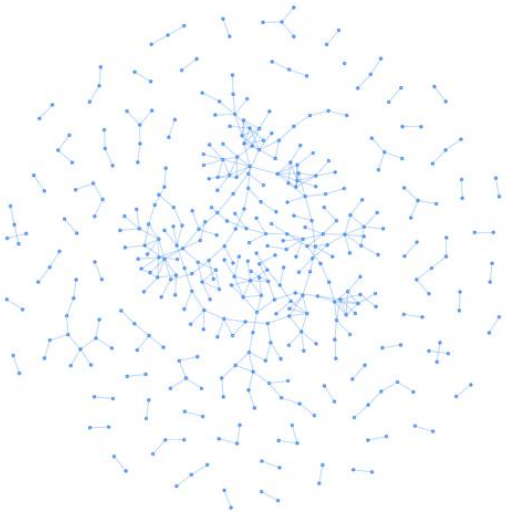
Hasil prediksi dari model *link prediction* yang digunakan selanjutnya diubah kembali bentuknya dari vektor *embedding* ke uri produk makanannya masing-masing. Selanjutnya untuk tiap pasangan produk makanan yang diprediksi berlabel 1

dibuatkan visualisasinya. Visualisasi dibuat menggunakan python dengan *library pyvis*.

Gambar *6.3* dan Gambar 6.4 berikut merupakan hasil visualisasi dari masing-masing algoritma *graph embedding* yang digunakan:



Gambar 6.3 Link prediction hasil embedding Node2Vec

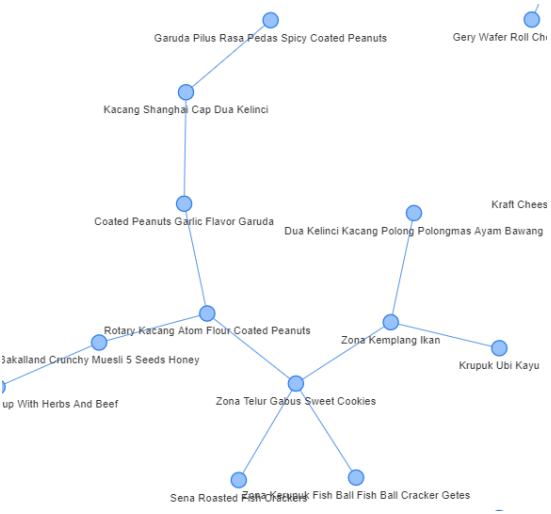


Gambar 6.4 Link prediction hasil embedding RDF2Vec

Berdasarkan hasil visualisasi dari *link prediction* kedua model pada

Gambar 6.3 dan Gambar 6.4 di atas, dapat dilihat bahwa terdapat lebih banyak node yang berhubungan pada visualisasi *link prediction* menggunakan RDF2Vec. Hal ini dikarenakan model dengan RDF2Vec memprediksi hubungan yang berlabel 1 lebih banyak dibandingkan model dengan Node2Vec. Akibatnya nilai recall yang dimiliki lebih tinggi, namun karena tidak semua hasil prediksi yang dihasilkan benar, maka mempengaruhi performa nilai *precision* yang jauh lebih kecil dibandingan dengan model *embedding* Node2Vec.

Selanjutnya, Gambar 6.5 menunjukan node produk makanan dengan kategori kacang yang saling terhubung satu sama lainnya.



Gambar 6.5 Contoh hubungan antar produk makanan hasil link prediction Node2Vec

Sedangkan Gambar 6.6 merupakan node produk makanan dengan kategori kacang yang saling terhubung satu sama lainnya. Produk ‘Rotary Kacang Atom Flour Coated Peanuts’ memiliki hasil prediksi hubungan yang benar dengan produk ‘Coated Peanuts Garlic Flavor Garuda’, namun tidak dengan produk ‘Zona Telur Gabut Sweet Cookies’.

Gambar 6.6 Contoh hubungan antar produk makanan hasil link prediction RDF2Vec

Berdasarkan visualisasi pada Gambar 6.5 dan Gambar 6.6 yang menunjukkan kategori produk makanan kacang-kacangan, dapat terlihat bahwa link yang dihasilkan dari embedding RDF2Vec lebih banyak, namun masih terdapat beberapa produk yang sebenarnya tidak berhubungan. Sedangkan pada link yang dihasilkan dari embedding Node2Vec, produk-produk yang saling berhubungan memiliki kemiripan yang lebih baik.

BAB VII
KESIMPULAN DAN SARAN

Dalam bab ini dijelaskan tentang kesimpulan dan saran yang didapatkan dari proses pelaksanaan tugas akhir ini

7.1 Kesimpulan

Kesimpulan yang didapatkan dari pengerjaan tugas akhir yang telah dilakukan adalah sebagai berikut:

1. Model link prediction dengan menggunakan algoritma Random Forest mampu memberikan hasil prediksi yang cukup baik. Model link prediction yang menggunakan embedding Node2Vec mampu mendapatkan akurasi sebesar 83.4% dan f1-score sebesar 82.7%, sedangkan model link prediction yang menggunakan embedding RDF2Vec mampu mendapatkan akurasi sebesar 80% dan f1-score sebesar 79.7%. Hal ini dikarenakan algoritma Random Forest mampu memberikan hasil klasifikasi yang baik dengan menggunakan data berbentuk vektor.
2. Model link prediction dengan menggunakan embedding Node2Vec mampu menghasilkan nilai rata-rata accuracy, precision, dan f1-score yang lebih baik dibandingkan model link prediction dengan menggunakan embedding RDF2Vec. Namun, model link prediction dengan menggunakan RDF2Vec dapat menghasilkan nilai rata-rata *recall* yang lebih tinggi dibandingkan dengan nilai rata-rata *precision*-nya. Hal ini dikarenakan model link prediction RDF2Vec menghasilkan prediksi label mirip lebih banyak dibandingkan model dengan Node2Vec.
3. Model link prediction yang menggunakan parameter hasil tuning memiliki performa lebih tinggi dibandingkan model link prediction yang menggunakan parameter default. Namun, perbedaan performa antar model tersebut tidak signifikan. Untuk model yang menggunakan embedding Node2Vec, antara model dengan parameter hasil tuning dan parameter default hanya memiliki perbedaan nilai f1-score sebesar 0.9%. Sedangkan model yang menggunakan

embedding RDF2Vec, antara model dengan parameter hasil tuning dan parameter default hanya memiliki perbedaan nilai f1-score sebesar 0.7%.

4. Algoritma Random Forest secara umum memiliki performa yang paling tinggi dalam melakukan link prediction antar produk makanan dari data yang digunakan dibandingkan beberapa algoritma machine learning lainnya seperti Gaussian Naïve Bayes, Logistic Regression, dan Gradient Boosting Classifier.
5. Semakin jauh perbandingan antara jumlah data pada label tidak mirip dan label mirip, maka nilai accuracy dan precision akan semakin meningkat, namun nilai f1-score dan recall akan semakin menurun dikarenakan imbalance data dimana jumlah data berlabel tidak mirip jauh lebih banyak. Sehingga, model akan memprediksi data berlabel tidak mirip lebih banyak guna meningkatkan accuracy, namun jumlah label mirip yang diprediksi akan semakin sedikit.

7.2 Saran

Seperti dijelaskan dalam kesimpulan bahwa model link prediction Random Forest yang diimplementasikan pada tugas akhir ini memberikan hasil prediksi data tes yang cukup memuaskan, namun hasil tersebut didapatkan dengan melakukan oversampling dan undersampling yang cukup banyak. Berikut ini diberikan beberapa saran yang dapat dilakukan untuk penyempurnaan implementasi link prediction dengan menggunakan model Random Forest:

1. Memperbanyak kombinasi skenario pengaturan parameter model link prediction untuk menemukan kombinasi parameter yang lebih baik untuk meningkatkan kinerja prediksi dari model link prediction.
2. Menambahkan atau mengurangi jumlah vektor embedding untuk menemukan pengaruh jumlah vektor hasil embedding terhadap hasil model link prediction.
3. Membuat model link prediction dengan algoritma machine learning lain dengan parameter hasil tuningnya untuk

mengetahui pengaruh penggunaan parameter untuk tiap model link prediction.

4. Menambahkan data produk makanan, khususnya untuk produk makanan yang saling berhubungan sehingga tidak dibutuhkan dilakukannya oversampling untuk model yang dibuat.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] "Muslim Population by Country 2020," *Review, World Population*, 2020. https://worldpopulationreview.com/country-rankings/muslim-population-by-country.

[2] R. Sham, R. Z. Rasi, N. Abdamia, S. Mohamed, and T. K. M. Thahira Bibi, "Halal Logistics Implementation in Malaysia: A Practical View," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 226, no. 1, 2017, doi: 10.1088/1757-899X/226/1/012040.

[3] WFHC, "World Halal Food Council." https://www.whfc-halal.com/about-us/history.

[4] N. A. Rakhmawati, J. Fatawi, A. C. Najib, and A. A. Firmansyah, "Linked open data for halal food products," *J. King Saud Univ. - Comput. Inf. Sci.*, no. xxxx, 2019, doi: 10.1016/j.jksuci.2019.04.004.

[5] M. Hasan, "Social Network Data Analytics," *Soc. Netw. Data Anal.*, no. April, 2011, doi: 10.1007/978-1-4419-8462-3.

[6] L. M. Aiello, A. Barrat, R. Schifanella, C. Cattuto, B. Markines, and F. Menczer, "Friendship prediction and homophily in social media," *ACM Trans. Web*, vol. 6, no. 2, 2012, doi: 10.1145/2180861.2180866.

[7] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *J. Am. Soc. Inf. Sci. Technol.*, vol. 58, no. 7, pp. 1019–1031, 2007, doi: 10.1002/asi.v58:7.

[8] P. Wang, B. W. Xu, Y. R. Wu, and X. Y. Zhou, "Link prediction in social networks: the state-of-the-art," *Sci. China Inf. Sci.*, vol. 58, no. 1, pp. 1–38, 2014, doi: 10.1007/s11432-014-5237-y.

[9] M. Rezaul Karim, M. Cochez, J. B. Jares, M. Uddin, O. Beyan, and S. Decker, "Drug-drug interaction

prediction based on knowledge graph embeddings and convolutional-LSTM network," *ACM-BCB 2019 - Proc. 10th ACM Int. Conf. Bioinformatics, Comput. Biol. Heal. Informatics*, pp. 113–123, 2019, doi: 10.1145/3307339.3342161.

[10] G. Crichton, Y. Guo, S. Pyysalo, and A. Korhonen, "Neural networks for link prediction in realistic biomedical graphs: A multi-dimensional evaluation of graph embedding-based approaches," *BMC Bioinformatics*, vol. 19, no. 1, pp. 1–11, 2018, doi: 10.1186/s12859-018-2163-9.

[11] H. Chen, Y. Hu, B. Perozzi, and S. Skiena, "HARP: Hierarchical representation learning for networks," *32nd AAAI Conf. Artif. Intell. AAAI 2018*, no. December, pp. 2127–2134, 2018.

[12] W3C, "LinkedData," 2016. https://www.w3.org/wiki/LinkedData.

[13] A. C. Najib, "Perangkingan Entitas Linked Open Data Menggunakan Pendekatan Resolusi Entitas Untuk Peningkatan Relevansi," 2020.

[14] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data - The story so far," *Int. J. Semant. Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, 2009, doi: 10.4018/jswis.2009081901.

[15] B. Guus Scheriber, VU University Amsterdam, Yves Raimond, "RDF 1.1 Primer," 2014. https://www.w3.org/TR/rdf11-primer/.

[16] H. Cai, V. W. Zheng, and K. C. C. Chang, "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1616–1637, 2018, doi: 10.1109/TKDE.2018.2807452.

[17] A. Grover and J. Leskovec, "*Node2Vec*," *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. data Min.*,

pp. 855–864, 2016, doi: 10.1145/2939672.2939754.

[18] P. R. B and H. Paulheim, “The Semantic Web – ISWC 2016,” vol. 9981, pp. 498–514, 2016, doi: 10.1007/978-3-319-46523-4.

[19] E. S. Team, “What is Machine Learning? A Definition,” *expert.ai*, 2020. https://www.expert.ai/blog/machine-learning-definition/.

[20] D. Volkmann, “Gravitationsbiologie— Kompensation oder Reduktion der Schwerkraft: Experimente auf Klinostaten oder unter Schwerelosigkeit?,” *Biol. unserer Zeit*, vol. 22, no. 6, pp. 323–329, 1992, doi: 10.1002/biuz.19920220617.

[21] M. DHAWANGKHARA, “Surabaya Menggunakan Teknik Random Forest Dan Cart (Studi Kasus Kota Surabaya),” 2016.

[22] M. S. Zahher, “Classification and Regression by Random Forest,” 2018. https://medium.com/@msz991/classification-and-regression-by-random-forest-2ee16cdc7b63.

[23] M. Bressan, F. Chierichetti, R. Kumar, S. Leucci, and A. Panconesi, “Counting graphlets: Space vs time,” *WSDM 2017 - Proc. 10th ACM Int. Conf. Web Search Data Min.*, pp. 557–566, 2017, doi: 10.1145/3018661.3018732.

[24] P. Pezeshkpour, Y. Tian, and S. Singh, “Investigating robustness and interpretability of link prediction via adversarial modifications,” *NAACL HLT 2019 - 2019 Conf. North Am. Chapter Assoc. Comput. Linguist. Hum. Lang. Technol. - Proc. Conf.*, vol. 1, pp. 3336–3347, 2019, doi: 10.18653/v1/n19-1337.

[25] M. Junker, R. Hoch, and A. Dengel, “On the evaluation of document analysis components by recall, precision,

and accuracy," *Proc. Int. Conf. Doc. Anal. Recognition, ICDAR*, pp. 717–720, 1999, doi: 10.1109/ICDAR.1999.791887.

[26] R. Arthana, "Mengenal Accuracy, Precision, Recall, dan Specificity serta yang diprioritaskan dalam Machine Learning," 2019. https://rey1024.medium.com/mengenal-accuracy-precission-recall-dan-specificity-serta-yang-diprioritaskan-b79ff4d77de8.

[27] G. Karyo Utomo, "Prediksi Kemiripan Produk Makanan berdasarkan Tingkat Persamaan Bahan Baku Dengan Menggunakan Algoritma Similarity Graph" 2021.

[28] Durajati, C., and Gumelar, A.B., "Pemanfaatan Teknik Supervised Untuk Klasifikasi Teks Bahasa Indonesia," *Jurnal Link Vol 16/No. 1, 1-8,* 2012.

[29] Feldman, R., and Sanger, J., "The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data," *New York: Cambridge University Press*, 2007.

[30] Hosmer, D.W. and Lemeshow, S., "Applied Logistic Regression," *New York: John Wiley & Sons, Inc.*, 2000.

[31] Agresi. A., "Categorical Data Analysis," *New York: John Wiley & Sons, Inc.*, 1990.

[32] Hasanah, Rini Nur, Purnomo Budi Santosa, Dodit Suprianto, "Sistem Pengenalan Wajah Secara Real-Time," Jurnal EECCIS, vol. 7, no. 2, p. 179, 2013.

[33] Spark, "Ensembles - spark.mllib," Spark, [Online]. Available: http://spark.apache.org/docs/latest/mllib-ensembles.html. [Accessed 4 Juni 2016].

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Penulis Bernama Irfan Rifqi Susetyo, lahir di Jakarta pada tanggal 23 Juni 1999, yang merupakan anak pertama dari tiga bersaudara. Penulis telah menempuh jenjang Pendidikan formal di beberapa sekolah yaitu: SD Muhammadiyah 08+ Duren Sawit (2005-2011), SMP Labschool Jakarta (2011-2014), SMAN 12 Jakarta (2014-2017). Penulis melanjutnya Pendidikan sarjana di Departemen Sistem Informasi Fakultas Teknologi Elektro dan Informatika Cerdas (FTEIC) Institut Teknologi Sepuluh Nopember (ITS) Surabaya pada tahun 2017 yang terdaftar sebagai mahasiswa dengan NRP 05211740000064.

Selama menjadi mahasiswa, penulis aktif mengikuti beberapa organisasi dan kepanitiaan seperti Himpunan Mahasiswa Sistem Informasi, UKM Flag Football, Information Systems Expo, dan Gerigi ITS 2019. Di bidang akademik, penulis aktif menjadi asisten dosen pada mata kuliah Sistem Basis Data, Manajamen Proyek TI, dan Kecerdasan Bisnis.Penulis juga sempat magang sebagai Data Visualization Intern di Enterprise Data Management – Bank Mandiri dan Software Tester Intern di Digiasia Bios.

Untuk mendapatkan gelar S.Kom (Sarjana Komputer), penulis mengambil topik penelitian tugas akhir *knowledge graph* pada laboratorium Akuisisi Data dan Diseminasi Informasi (ADDI). Untuk kepentingan penelitian, Penulis dapat dihubungi melalui email irfnrifqi@gmail.com

Halaman ini sengaja dikosongkan