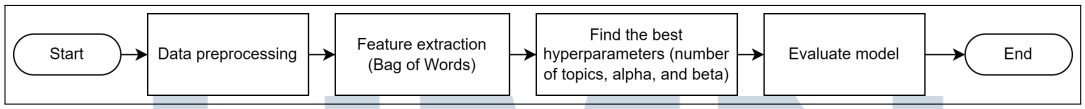
BAB 3
METODOLOGI PENELITIAN

3.1 Analisis Kebutuhan dan Studi Literatur

Analisis kebutuhan dilakukan dengan bertanya kepada pihak perusahaan mengenai permasalahan yang sedang dihadapi perusahaan dan sistem yang diperlukan perusahaan. Selain itu, dilakukan juga diskusi terkait rencana perancangan sistem secara detail, khususnya terkait sistem operasi, bahasa pemrograman, dan *framework* yang akan digunakan. Setelah itu, dilakukan studi literatur untuk mengumpulkan berbagai informasi dari penelitian terdahulu sebagai sumber referensi dan menerapkannya pada penelitian ini.

3.2 Perancangan Model LDA

Gambar 3.1 berikut merupakan *flowchart* yang menggambarkan alur perancangan model LDA. Dimulai dari *data preprocessing*, *feature extraction* dengan *Bag of Words* (BoW), mencari *hyperparameter* terbaik untuk model, dan evaluasi model.



Gambar 3.1. *LDA model development flow*

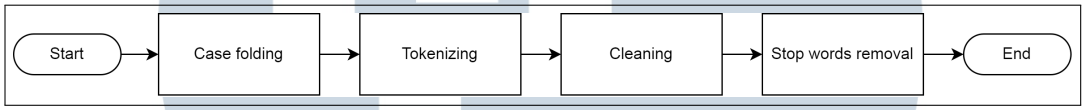
3.2.1 Import Dataset

Langkah pertama yang dilakukan adalah mencari dan meng-*import dataset*. *Dataset* yang digunakan terdiri dari *dataset* primer dan sekunder. *Dataset* sekunder yang digunakan berjumlah 1266 data yang diambil dari Kaggle, sedangkan *dataset* primer berjumlah 37 data *email* yang didapat dari perusahaan. Terdapat keterbatasan dalam pengumpulan *dataset* perusahaan karena sebagian besar *email* yang dikirimkan berisi data *confidential* (seperti data pribadi karyawan dan nominal gaji karyawan) sehingga proses perancangan model akan lebih banyak menggunakan Kaggle *dataset*. Kode 3.1 merupakan kode yang digunakan untuk meng-*import dataset*.

```
df = pd.read_csv('../../dataset.csv', usecols=['Pesan'])
emails = df.to_numpy()
mails = []
for mail in emails:
    mails.append(mail[0])
```

Kode 3.1: *Import dataset*

3.2.2 Data Preprocessing



Gambar 3.2. *Data preprocessing flow*

Alur *data preprocessing* dapat dilihat pada Gambar 3.2. Pada tahap ini, metode *stemming* atau *lemmatization* tidak digunakan karena daftar *stop words* yang digunakan tidak berisi kata dasar, melainkan kata berimbuhan lengkap. Berikut merupakan penjelasan untuk masing-masing proses.

1. *Case folding*
   Langkah pertama adalah melakukan *case folding*, yaitu mengubah semua huruf pada teks menjadi huruf kecil (*locwercase*).

2. *Tokenizing*
   Langkah selanjutnya adalah melakukan *tokenizing*, yaitu mengubah kalimat-kalimat pada teks menjadi token kata.

3. *Cleaning*
   Langkah berikutnya adalah *cleaning*, yaitu menghilangkan tanda baca dan angka dari teks sehingga hanya menyisakan alfabet (huruf a-z).

4. *Stop words removal*
   Langkah terakhir pada tahap *preprocessing* adalah *stop words removal*, yaitu menghilangkan kata-kata yang sering muncul dan tidak memiliki arti penting.

Kode 3.2 merupakan *function* yang digunakan untuk melakukan *preprocessing*. Pertama, semua huruf diubah menjadi huruf kecil menggunakan *built-in function* lower() dan mengubah kalimat menjadi token kata menggunakan

split(), seperti yang terlihat pada baris ke-9. Lalu, seluruh angka dan tanda baca dihilangkan dengan menggunakan *regex* (baris ke-13 dan 14). Langkah terakhir adalah menghilangkan *stop words* yang ada (baris ke-17 sampai 19). Daftar *stop words* yang digunakan berasal dari *library* NLTK (baris ke-2) dan Kaggle [43] (baris ke-3).

```
1  def preprocess(text):
2      stop_words = stopwords.words('indonesian')
3      stopwords2 = [line.rstrip('\n\r') for line in open(os.path.join(os.getcwd(), '
       stopwordbahasa.txt'))]
4      for w in stopwords2:
5          if w not in stop_words:
6              stop_words.append(w)
7
8      # Case Folding & Tokenizing
9      tokens = [word.lower() for word in text.split()]
10     filtered_tokens = []
11
12     # Cleaning
13     punctuation = re.compile(r'[-.?!,"':;\'@/()|0-9]')
14     tokens = [punctuation.sub("", token) for token in tokens]
15
16     # Stop Words Removal
17     for x in tokens:
18         if x not in stop_words and len(x) >= 3:
19             filtered_tokens.append(x)
20
21     return filtered_tokens
```

Kode 3.2: *Data preprocessing function*

3.2.3 Feature Extraction

Feature extraction dilakukan dengan membentuk objek *Bag of Words* (BoW) dari data hasil *preprocessing*. Selanjutnya, objek BoW ini akan digunakan sebagai *input* untuk model LDA. Dengan merepresentasikan data dalam bentuk BoW, setiap kata-kata dari hasil *preprocessing* akan diberikan bobot sesuai dengan frekuensi kemunculan kata tersebut sehingga akan terlihat kata apa saja yang sering muncul dan dianggap penting dalam setiap *email*. Kode 3.3 berikut merupakan proses konversi data hasil *preprocessing* menjadi *BoW corpus* menggunakan *library* Gensim.

```
dictionary = corpora.Dictionary(preprocessed)
corpus = [dictionary.doc2bow(text) for text in preprocessed]
```

Kode 3.3: *Creating BoW corpus*

BoW corpus yang dihasilkan dari kode di atas dapat dapat dilihat pada Gambar 3.3. Setiap baris berisi *id* kata dan frekuensi kemunculan kata yang ada pada setiap *email*. Contohnya pada data *email* pertama (baris pertama), kata dengan *id* = 0 muncul sebanyak 1 kali, kata dengan *id* = 1 muncul sebanyak 1 kali, dan seterusnya.

```
[(0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1), (13, 1), (14, 1),
(15, 1), (16, 1), (17, 1)]


[(0, 1), (1, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1), (14, 1), (15, 1), (17, 1), (18,
1), (19, 1), (20, 1)]


[(0, 1), (4, 1), (5, 1), (6, 1), (7, 1), (8, 1), (9, 1), (10, 1), (11, 1), (12, 1), (18, 1), (21, 1), (22, 1), (23, 1)]


[(0, 2), (3, 1), (7, 1), (8, 1), (9, 1), (11, 1), (16, 1), (17, 1), (18, 2), (23, 1), (24, 1), (25, 1), (26, 1)]


[(0, 2), (9, 1), (27, 1), (28, 1), (29, 2), (30, 1), (31, 1), (32, 1), (33, 1), (34, 1), (35, 1), (36, 1), (37, 1), (38, 1),
(39, 1), (40, 1)]


[(41, 1), (42, 1), (43, 1), (44, 1), (45, 1), (46, 1)]
```

Gambar 3.3. *BoW corpus*

3.2.4 Pengujian Model

Pada tahap ini, akan dilakukan percobaan menggunakan berbagai nilai parameter alpha, beta, dan jumlah topik pada model LDA untuk mencari nilai koherensi tertinggi. Jumlah topik yang digunakan adalah 1, 3, 5, 7, dan 10. Kode 3.4 merupakan kode proses *looping* untuk mencari nilai koherensi tertinggi dari setiap jumlah topik.

```
no_of_topics = [1, 3, 5, 7, 10]
for n in no_of_topics:
    lda_model = ldamodel.LdaModel(corpus=corpus,
                                  id2word=dictionary,
                                  num_topics=n,
                                  random_state=42,
                                  passes=100,
                                  alpha='auto',
                                  per_word_topics=True,
                                  eta='auto')

    coherence_model_lda = CoherenceModel(model=lda_model,
```

```
                                        texts=preprocessed,
                                        dictionary=dictionary,
                                        coherence='c_v')

    coherence_lda = coherence_model_lda.get_coherence()
    print(f"n: {n} | Score: {coherence_lda}")
```

Kode 3.4: *Finding the best number of topics*

Selanjutnya, dilakukan juga percobaan untuk mencari nilai α dan β terbaik. α merupakan parameter yang mengontrol distribusi topik terhadap dokumen (*email*) dan β adalah parameter yang mengontrol distribusi kata terhadap topik. Interval nilai α dan β bisa bervariasi, mulai dari 0 hingga 100. Namun, nilai α dan β yang digunakan pada penelitian ini adalah 0.1, 0.25, 0.75, 1, dan 'auto'. Hal ini dikarenakan penelitian terdahulu [36] telah membuktikan bahwa untuk mendapatkan hasil pengelompokan dokumen yang lebih baik dengan topik yang koheren, sebaiknya menghindari penggunaan nilai di atas 1. Kode 3.5 merupakan proses *looping* yang dilakukan.

```
alpha = [0.1, 0.25, 0.5, 0.75, 1, 'auto']
beta = [0.1, 0.25, 0.5, 0.75, 1, 'auto']

for a in alpha:
    for b in beta:
        lda_model = ldamodel.LdaModel(corpus=corpus,
                                      id2word=dictionary,
                                      num_topics=1,
                                      random_state=42,
                                      passes=10,
                                      alpha=a,
                                      per_word_topics=True,
                                      eta=b)

        coherence_model_lda = CoherenceModel(model=lda_model,
                                             texts=preprocessed,
                                             dictionary=dictionary,
                                             coherence='c_v')

        coherence_lda = coherence_model_lda.get_coherence()
        print(f"a: {a} | b: {b} | Score: {coherence_lda}")
```

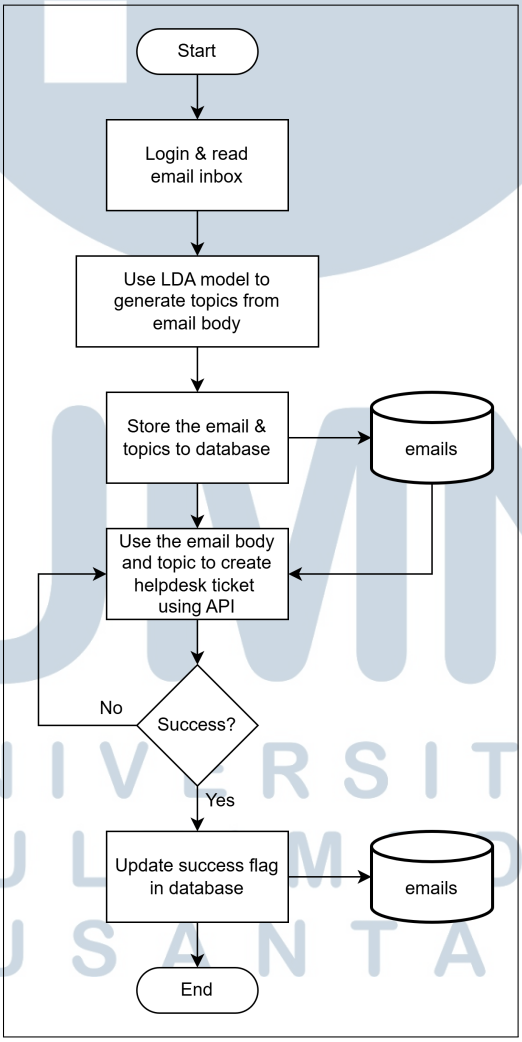
Kode 3.5: *Finding the best hyperparameter (alpha & beta)*

3.2.5 Evaluasi

Selanjutnya, evaluasi akan dilakukan menggunakan metode *topic coherence* C_v untuk mengukur koherensi topik yang dihasilkan dalam setiap pengujian. Parameter yang menghasilkan nilai koherensi tertinggi akan dipilih untuk diimplementasikan pada RPA.

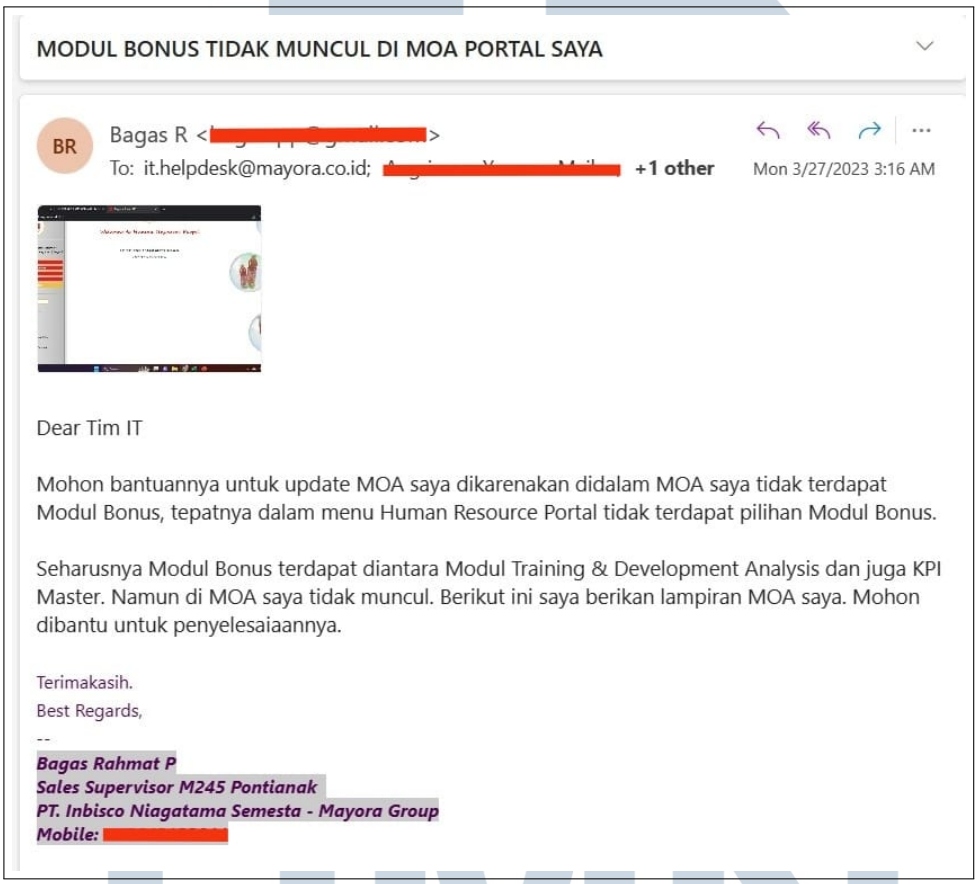
3.3 Implementasi Model LDA pada RPA

Setelah menemukan parameter yang sesuai pada model LDA, maka model tersebut akan digabungkan dalam rangkaian RPA. Berikut merupakan gambaran mengenai alur RPA yang akan dibuat secara umum.



Gambar 3.4. *RPA flow*

Proses pertama yang dilakukan adalah *login* ke akun *email* pada Zimbra dan membaca *inbox*. Selanjutnya, setiap *email body* yang dibaca akan digunakan sebagai input untuk model LDA setelah dilakukan *preprocessing*. Gambar 3.5 merupakan contoh data *email* permasalahan karyawan.



Gambar 3.5. *Employee problem email*

Model LDA tersebut akan dijalankan sehingga menghasilkan *output* berupa topik yang mewakili isi *email*. Kemudian *email* tersebut akan disimpan ke dalam *database* beserta dengan topik yang didapat. Topik yang dihasilkan akan diletakkan pada bagian awal deskripsi tiket dan dilanjutkan dengan penjelasan masalah (*body email*). Proses selanjutnya adalah melakukan *create ticket*. Pembuatan tiket tersebut akan dilakukan melalui API agar proses dapat berjalan tanpa menggunakan *interface* (sebagai *background job*). Data tersebut merupakan data yang diambil dari *email* yang telah diproses dan tersimpan di *database*. Jika tiket sudah berhasil terbentuk, maka akan dilakukan *update* status *email* pada *database* untuk menandakan bahwa sudah ada tiket untuk *email* tersebut. Jika belum berhasil, maka proses pembuatan tiket akan diulangi hingga berhasil.