

Aspect-based Sentiment Analysis menggunakan pendekatan Semisupervised: Multinomial Naive Bayes, Logistic Regression & Neural Network

Kelompok 14:

Helmi Alfarel - 180623586

Muhammad Afiful Amin - 1706039963

Bagian 1. Task Definition

Diberikan data yang berisi ulasan restoran di Indonesia, Lakukan sentiment analysis pada ulasan tersebut terhadap 4 aspect nya, yaitu FOOD, AMBIENCE, SERVICE, PRICE. Sebuah ulasan bisa jadi tidak mengandung ulasan mengenai suatu aspek, dan sentimen dari sebuah aspek bisa bernilai positif atau negatif.

Dataset untuk training yang berisi ulasan yang sudah berlabel terdiri dari 3865 ulasan. Dataset yang belum berlabel terdiri dari 159026 ulasan. Dataset untuk testing atau untuk submisi terdiri dari 14028 ulasan. Masing-masing dataset tersimpan dalam bentuk xml.

Bagian 2. Text preprocessing

Pada bagian ini, kami akan membahas mengenai pemrosesan dataset sehingga dapat digunakan oleh model machine learning. Terdapat beberapa tahap pada

bagian ini, yaitu mengubah bentuk dataset dari tree ke bentuk tabular, melakukan text cleaning, dan melakukan text vectorization.

Bagian 2.1 From Trees to Tables

Dataset yang diberikan berbentuk tree yang disimpan dalam file berformat XML.

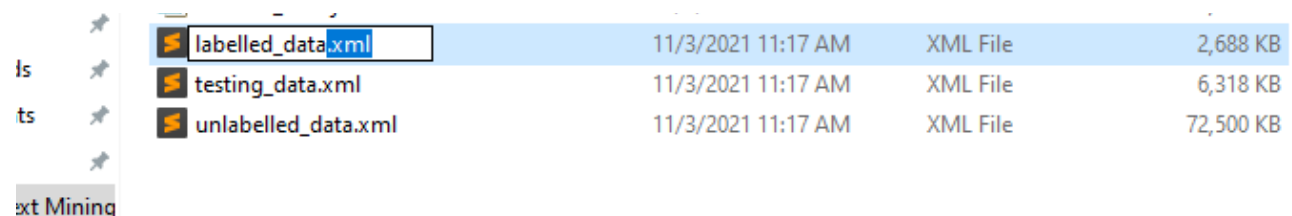
Terdapat banyak library pada python yang membantu kita mengolah file XML.

Namun, file labelled_data.xml, memiliki banyak sekali ketidaknormalan, yang menyebabkan library jadi error saat mengolah file tersebut. Setelah mencoba mengolah file xml tersebut berkali-kali tanpa hasil, saya akhirnya mencari cara lain.

Pertama, saya mengubah format file labelled_data.xml menjadi labelled_data.html.

Kedua, saya menggunakan tools online untuk mengubah file html menjadi file json.

Kemudian file json tersebut bisa saya olah dengan mudah.



labelled_data.xml	11/3/2021 11:17 AM	XML File	2,688 KB
testing_data.xml	11/3/2021 11:17 AM	XML File	6,318 KB
unlabelled_data.xml	11/3/2021 11:17 AM	XML File	72,500 KB

Untuk pemrosesan file unlabelled_data.xml dan testing_data.xml, saya menggunakan cara yang ditunjukkan pada Tutorial 1 -Text Mining.

Bagian 2.2 Text Cleaning

Text Cleaning adalah proses yang dilakukan untuk menyiapkan raw text sebelum diolah oleh machine learning model. Umumnya, terdapat beberapa tahapan pada proses ini [1], yaitu:

- Menghilangkan karakter yang tidak diinginkan, seperti extra whitespace dan extra line

- Mengubah kedalam lower case dan tokenisasi. Tokenisasi adalah proses yang merubah sebuah dokumen menjadi list of tokens.
- Menghilangkan stopwords
- Melepaskan kata-kata yang tergabung. Misalnya '#AwesomeDay', '#DataScientist', '#BloggingIsPassion'
- Lematisasi atau Stemming. Proses ini mengubah suatu kata menjadi bentuk dasarnya. Misal: Menulis = Tulis, Playing = Play, dsb.
- Melakukan pengoreksian ejaan dan grammar.

Pada awalnya, tahapan data cleaning yang saya buat adalah:

1. Mengubah teks menjadi lowercase
2. Menghilangkan semua angka menggunakan regex
3. Menghilangkan semua extra whitespace dan extra line
4. Melakukan stemming bahasa Indonesia menggunakan Sastrawi
5. Menghilangkan stopword bahasa Indonesia menggunakan Sastrawi
6. Menghilangkan stopword bahasa Inggris menggunakan NLTK
7. Melakukan word tokenisasi dengan NLTK

Masalah utama yang saya amati dari proses ini adalah, waktu untuk melakukan text cleaning sangat lama. Pada dataset `unlabelled_data`, waktu yang diperlukan dalam proses text cleaning melebihi 30 menit! Terdapat isu lain yang saya amati, yaitu dengan hilangnya stopword, hilang juga token-token seperti "Tidak", "Tak", "Not", dan kata negasi lainnya. Hal ini kurang baik menurut saya, karena saya berpendapat kata negasi diatas sangat berpengaruh terhadap polaritas sentimen review nya. Review yang berkonotasi positif biasanya menggunakan kata sifat positif, seperti enak, nyaman, murah dan sebagainya, dan ketika menuliskan review negatif,

terdapat beberapa yang menuliskan review dengan sopan dengan cara tidak menuliskan kata sifat yang berkonotasi negatif secara langsung (cth: Berisik, mahal, panas), namun menggunakan kata sifat positif yang dinegasikan seperti tidak enak, kurang nyaman, not cheap. Tentunya saya perlu membuktikan hal ini dengan menunjukkan buktinya secara statistik, namun dalam kasus ini (bahwa ini hanya tugas kelompok), saya merasa membuat asumsi diatas tidak terlalu buruk/berbahaya. Berikut beberapa contoh kejadian diatas:

```
<text>
Pertama kali mencoba cafe ini dan karena sebelumnya sudah makan, maka saya cuma memesan kopi. Saya memesan cappuccino, rasanya cukup enak. Selain itu saya juga memesan ice latte, nah untuk yang ini kopinya tidak berasa sama sekali.
</text>
<aspects id="0">
<aspect category="FOOD" polarity="NEGATIVE"/>
```

```
Lumayan suka sushi dan lagi ngidam sushi waktu mau nonton. Akhirnya milih ke sini karena penasaran. Pesennya sushi moriawase B. Overall... Biasa sih. Not so bad. Not so good. So-so lah rasanya. Harganya, hmm kalo aku sih beberapa menunya kurang worth ya sama harganya.
</text>
<aspects id="0">
<aspect category="FOOD" polarity="POSITIVE"/>
<aspect category="PRICE" polarity="NEGATIVE"/>
</aspects>
```

```
Pesan double pots kuah beef (recomend dari pelayan nya) side dish nya pilih yg squid goreng apa lah namanya itu dgn harga hemm lupa tepatnya tp sekitar 70rb an . Rasa kuahnya kurang enak tapi masih bisa untuk dinikmati kebantu campuran semacam bumbu2 gtu deh yg ada dimeja itu ngebuat gurih kuahnya gtu, squid gorengnya enakk tp ditepunya nya berasa amis amis gtu rasanya. Isinya termasuk pelit :( slice beefnya cuma dapet 2lembar mungkin karna harganya murah ya. Tempat di citos ini kecil dan biasa aja, cuma buat sekedar makan terus pergi bukan tempat yg terlalu nyaman untuk duduk2 berlama lama.
</text>
<aspects id="0">
<aspect category="FOOD" polarity="NEGATIVE"/>
<aspect category="PRICE" polarity="POSITIVE"/>
<aspect category="AMBIENCE" polarity="NEGATIVE"/>
```

Tahapan text cleaning yang selanjutnya saya buat adalah:

1. Mengubah teks menjadi lowercase
2. Menghilangkan semua angka menggunakan regex
3. Menghilangkan semua extra whitespace dan extra line
4. Menghilangkan karakter berulang lebih dari 2 kali dengan regex. Misalnya, enakkkkkkk = enak, makanaaaaaan = makanan, wwwoooooiiiiiii = woi. Tahapan ini dilakukan untuk menyerupai proses stemming dan pengoreksian ejaan.

5. Melakukan word tokenisasi dengan NLTK, yang menghilangkan punctuation dan emoji.

```
def clean(text):
    clean_text = text.lower()
    clean_text = re.sub(r'[\d]', '', clean_text)
    clean_text = re.sub(r'^\w\s', '', clean_text)
    clean_text = re.sub(r'\s+', ' ', clean_text)
    clean_text = re.sub(r'(\w)\1+', r'\1', clean_text) # ganti karakter berulang >= 2 kali jadi 1 karakter
    clean_text = clean_text.strip()

    clean_text_tokens = nltk.word_tokenize(clean_text)
    clean_text = " ".join(clean_text_tokens)

    return clean_text
```

Keuntungan yang langsung dirasakan adalah durasi text cleaning berkurang secara drastis. Untuk melakukan text cleaning pada unlabelled_data, waktu yang diperlukan hanya sekitar 3 menit.

Walaupun begitu, terdapat beberapa isu yang saya hadapi pada bagian ini:

- Terdapat data yang hanya terdiri dari titik-titik (.....) dan hanya terdiri dari emoji.
- Terdapat data dengan bahasa yang tidak menggunakan alfabet latin.

Berikut adalah beberapa contohnya.

```
df_unlabelled.loc[df_unlabelled["clean_text"] == "", :]
```

	text	clean_text
4203		0
4285	
5198	...	
6853	... 🤔	
9642	:{(
...
152826		0
153931!	
154556	🤔	
157836	
158844		0

```
print(row[0], " ", row[1]['text'], " ", row[1]['clean_text'])
```

499 这个餐馆除了羊肉串还可以, 其他菜都非常普通, 但价格超过五星级饭店的水平, 税费服务费还要另外增加至少15个点, 这样一算, 一串羊肉串的价格快达到16千了, 食物品质远远不能匹配价格, 可能他是国内一些旅行团的合作餐厅的原因吧, 老板人很热情, 但谈到团体折扣就不松口, 让人感觉他的意思是你爱来不来的样子, 我们单位其实属于大客户, 一次活动就10条之外, 但现在我们一般不会去了, 性价比太低了, 这不是我自己这样认为, 我们很多朋友都是这个评价.....要解馋, 已经有替代 这个餐馆除了羊肉串还可以其他菜都非常普通但价格超过五星级饭店的水平税费服务费还要另外增加至少个点这样一算一串羊肉串的价格快达到千了食物品质远不能匹配价格可能他是国内一些旅行团的合作餐厅的原因吧老板人很热情但谈到团体折扣就不松口让人感觉他的意思是你爱来不来的样子我们单位其实属于大客户一次活动就条之外但现在我们一般不会去了性价比太低了这不是我自己这样认为我们很多朋友都是这个评价要解馋已经有替代

4178 . Hahahaha hahahaha

5877 . Hahahaha hahahaha

6162 Sashimi sashimi

11573 일루아한국레스토랑. 현재소프트오프닝중으로입니다. 6월에가서음식을먹어봤는데, 특히육개장, 물냉면맛있더라고요. 한국사람입맛에두아주좋더라고요. 참고로홍콩에서너무맛있게먹었던육개장보다훨씬더좋더라고요. 사리나쇼핑센타근처이던데. 인테리어두깔끔하구. 한국분매니저님두친절하시구. 대박나세요 일루아한국레스토랑현재소프트오프닝중으로입니다6월에가서음식을먹어봤는데특히육개장을냉면맛있더라고요한국사람입맛에두아주좋더라고요참고로홍콩에서너무맛있게먹었던육개장보다훨씬더좋더라고요사리나쇼핑센타근처이던데인테리어두깔끔하구한국분매니저님두친절하시구대박나세요

43364 " "

Solusi yang saya ambil adalah dengan menghapus data yang memenuhi kategori diatas, jika data tersebut berada pada labelled_data atau unlabelled_data. Jika data testing tentu saya tidak bisa hapus datanya dan saya hanya bisa *berharap yang terbaik*.

Berikut adalah cara saya membuang data yang kurang baik tersebut. Pada saat melakukan filter data yang not-alphanumeric, entah kenapa terdapat text yang menurut saya merupakan alphanumeric juga ditampilkan, sehingga saya menuliskan index yang ingin dihapus secara manual.

```
# Buang data yang jelek, ngga semuanya
drop_list = np.array(df_unlabelled.loc[df_unlabelled["clean_text"] == "", :].index).tolist() #dari yg clean_text nya == ""
not_alphanumeric = [7407, 36651, 43742, 61740, 63852, 69086, 93026, 96568, 110142, 108759, 130316]

drop_list = drop_list + not_alphanumeric

df_unlabelled = df_unlabelled.drop(index=drop_list)
```

Bagian 2.3 Text Transformation / Feature Engineering

Text Transformation adalah cara untuk mengubah/merepresentasikan teks menjadi bentuk yang dapat digunakan oleh machine learning model secara langsung.

Terdapat banyak cara untuk merepresentasikan teks, yang biasanya dibagi menjadi dua kelompok:

1. Feature Vectors

Feature Vectors adalah teknik untuk merepresentasikan teks menjadi *sparse vector* yang (biasanya) berdimensi besar atau memiliki feature yang banyak, dimana

sebuah feature merefleksikan sebuah konteks dalam teks. Misalnya konteksnya bisa berupa jumlah frekuensi suatu kata dalam teks, atau frekuensi sebuah kata dilanjutkan oleh kata lain (N-gram). Beberapa teknik yang termasuk Feature Vectors adalah Bag-of-Words, N-Gram, TF-IDF.

2. Word Embedding

Word Embedding adalah teknik untuk merepresentasikan teks menjadi vektor yang berdimensi kecil. Feature dalam vektor ini biasanya tidak memiliki arti/semantic yang intrinsic, melainkan memiliki sifat *latent*, dan biasanya didapatkan menggunakan teknik lain seperti *distributional semantics*. Beberapa contoh teknik ini adalah Word2Vec (Google), GloVe (Stanford), fastText (Facebook), BERT (Google).

Berdasarkan deskripsi diatas, jelas teknik Feature Vectors akan memiliki interpretabilitas yang lebih baik daripada Word Embedding. Sehingga untuk mempermudah pengerjaan, saya menggunakan teknik Bag-of-Words, dengan menggunakan bantuan CountVectorizer dari library Scikit-learn.

Pada teknik Bag-of-Words, ada hal yang disebut Vocabulary, yaitu himpunan token yang dijadikan fitur dalam vektor representasi teks. Sangat penting untuk memperhatikan token apa yang ada di vocabulary karena sangat mempengaruhi vektor yang dibuat. Token yang sebaiknya ada pada Vocabulary merupakan token yang memiliki pengaruh terhadap sentimen. Seperti Enak, Murah, Makanan, Sepi, Harga, Staff, Service, Mahal, Murah, Pahit, Berisik, Ramah, dan sebagainya. Token yang tidak berpengaruh kepada sentimen adalah token stopwords.

Stopwords adalah kata umum (common words) yang biasanya muncul dalam jumlah besar dan dianggap tidak memiliki makna. Contoh beberapa stopwords adalah Yang, Ada, Di, Sini, Sana, Of, The, Ke, Tidak, Not, Adalah dll. Seperti yang saya bahas di bagian Text Cleaning beberapa stopwords seperti Tidak, Not, Tak, merupakan token yang berpengaruh saat melakukan sentiment analysis, jadi sebaiknya beberapa token-token stopwords diatas itu dimasukkan ke dalam vocabulary.

Token lain yang sebaiknya tidak dimasukkan ke dalam vocabulary adalah token yang memiliki frekuensi yang sangat rendah. Berikut adalah implementasi saya menciptakan Vocabulary dari seluruh teks di dataset dan membuang token yang tidak diinginkan.

```
vectorizer = CountVectorizer()  
transformed = vectorizer.fit_transform(df_train['clean_text'].tolist())  
transformed = transformed.toarray()
```

```
X = pd.DataFrame(data = transformed, columns = vectorizer.get_feature_names())
```

CountVectorizer adalah module yang mengubah sebuah teks menjadi vektor representasi bag-of-words.

Transformed berisi vektor-vektor representasi teks. Contoh nya:

contoh_text = ['Saya makan nasi padang', 'Padang terkenal oleh rendang', 'Di padang, makan nasi padang']

vectorizer.fit_transform(contoh_text)

Hasil =

1	1	1	1	0	0	0
0	0	0	1	1	1	0

0	1	1	2	0	0	1
---	---	---	---	---	---	---

Vectorizer.get_feature_names() mengembalikan list of token yang menjadi vocabulary. Dalam contoh diatas, Vectorizer.get_feature_names() berisi

Saya	makan	Nasi	Padang	Terkenal	Rendang	Di
------	-------	------	--------	----------	---------	----

Sehingga X berisi:

Saya	makan	Nasi	Padang	Terkenal	Rendang	Di
1	1	1	1	0	0	0
0	0	0	1	1	1	0
0	1	1	2	0	0	1

Dengan menggunakan X, kita bisa mencari token stopwords (yang punya frekuensi tinggi), dan token yang memiliki frekuensi rendah. Pertama, mencari Stopwords

```
np.array(X[X.columns[X.sum()>800]].columns).tolist() # Mencari stopwords yang harus dibuang.
# Stopword biasanya jumlahnya jauh lebih banyak drpd token yang penting
# Bisa dilihat pada threshold sum > 800, column2 nya sudah tidak membantu
# Dalam menentukan sentimen
# kecuali 'ga', 'enak', 'place', 'not', 'tempat', 'tempatnya', 'rasanya'
```

```
['ada',
 'and',
 'banget',
 'buat',
 'but',
 'dan',
 'di',
 'enak',
 'fod',
 'for',
 'ga',
 'god',
 'in',
 'ini',
 'is',
 'it',
 'juga',
 'makan',
 'my',
 'not',
 'nya',
 'of',
 'pas',
 'place',
 'rasanya',
 'sama',
 'saya',
 'sih',
 'so',
 'tapi',
 'tempat',
 'tempatnya',
 'the',
 'this',
 'to',
```

Dengan data training yang terdiri dari 3865 teks, kata yang memiliki frekuensi lebih dari 800 kali biasanya merupakan kata stopwords, namun harus diperhatikan bahwa masih ada kata yang berpengaruh pada sentimen, seperti 'ga', 'enak', 'place', 'not', 'tempat', 'tempatnya', 'rasanya'.

Coba bandingkan jika saya menetapkan frekuensi 400 sebagai threshold nya,

```
a = np.array(X[X.columns[X.sum()>800]].columns).tolist()
b = np.array(X[X.columns[X.sum()>400]].columns).tolist()

c = [i for i in b if i not in a]

print(c)

['agak', 'aja', 'are', 'as', 'ayam', 'banyak', 'bgt', 'biasa', 'bisa', 'chicken', 'cofe', 'cukup', 'cuma', 'dari', 'dengan', 'd
isini', 'gak', 'goreng', 'gue', 'gw', 'harga', 'have', 'here', 'ice', 'its', 'itu', 'jadi', 'just', 'kalo', 'karena', 'ke', 'ke
sini', 'kurang', 'lagi', 'lebih', 'like', 'love', 'lumayan', 'makanan', 'makananya', 'mau', 'menu', 'nasi', 'nice', 'on', 'on
e', 'ordered', 'pesen', 'rasa', 'service', 'suka', 'taste', 'terlalu', 'that', 'their', 'they', 'tp', 'very', 'we', 'ya', 'yo
u']
```

Terdapat banyak token penting yang memiliki frekuensi lebih dari 400, seperti like, lumayan nice, suka, taste,

Untuk menentukan token yang frekuensinya sangat kecil sehingga tidak berguna, berikut implementasinya. Saya menggunakan threshold total frekuensi token > 10 untuk dikatakan merupakan token yang berguna. Dengan dataset yang berisi 3865 teks dan threshold minimal frekuensi token = 10, saya mencari token yang dalam rata2, muncul minimal satu kali setiap 386 teks. Bisa dilihat, token yang memiliki frekuensi lebih dari 10 hanya sekitar 2457 token, sehingga vektor representasi kita akan memiliki fitur berjumlah 2457 dikurangi jumlah token yang saya tetapkan sebagai stopwords diatas.

```
# Cari token yang terlalu sedikit, jadi ngga berguna
print('Original- vocab size')
print(len(X.columns))
print('remove lowest tokens - vocab size')
print(len(X[X.columns[X.sum()>10]].columns))
```

```
Original- vocab size
18342
remove lowest tokens - vocab size
2457
```

Bagian 3. Multinomial Naive Bayes

Algoritma Multinomial Naive Bayes memprediksi class sebuah input d dengan cara menghitung:

$$c_{map} = \arg \max_{c \in C} \hat{P}(c | d) = \arg \max_{c \in C} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k | c)$$

Yaitu Class c yang memiliki Probabilitas conditional $P(c | d)$ paling tinggi, dengan $P(c|d)$ adalah probabilitas prior kelas c dikali dengan total likelihood setiap token t dalam dokumen d memiliki kelas c $P(t | c)$. Dalam multinomial $P(t | c)$

merepresentasikan jumlah frekuensi token t yang berada dalam dokumen d yang berlabel c . Sedangkan pada bernoulli $P(t | c)$ merepresentasikan jumlah dokumen d yang memiliki token t dan juga berlabel c .

Untuk menjalankan model Naive menggunakan pendekatan semi supervised, saya mencoba untuk mengikuti prosedur yang dibuat pada sumber [2]. Yaitu:

- **Inputs:** Collections X_l of labeled documents and X_u of unlabeled documents.
- Build an initial naive Bayes classifier, $\hat{\theta}$, from the labeled documents, X_l , only. Use maximum a posteriori parameter estimation to find $\hat{\theta} = \arg \max_{\theta} P(X_l | \theta) P(\theta)$ (see Equations 3.5 and 3.6).
- Loop while classifier parameters improve, as measured by the change in $l(\theta | X, Y)$ (the log probability of the labeled and unlabeled data, and the prior) (see Equation 3.8):
 - **(E-step)** Use the current classifier, $\hat{\theta}$, to estimate component membership of each unlabeled document, *i.e.*, the probability that each mixture component (and class) generated each document, $P(c_j | x_i; \hat{\theta})$ (see Equation 3.7).
 - **(M-step)** Re-estimate the classifier, $\hat{\theta}$, given the estimated component membership of each document. Use maximum a posteriori parameter estimation to find $\hat{\theta} = \arg \max_{\theta} P(X, Y | \theta) P(\theta)$ (see Equations 3.5 and 3.6).
- **Output:** A classifier, $\hat{\theta}$, that takes an unlabeled document and predicts a class label.

Namun, ada hal yang saya tidak ikuti, yaitu looping dilakukan dengan memperhatikan improvement dari modelnya, dihitung dengan

$$l(\theta | X, Y) = \log(P(\theta)) + \sum_{x_i \in X_u} \log \sum_{j \in [M]} P(c_j | \theta) P(x_i | c_j; \theta) + \sum_{x_i \in X_l} \log (P(y_i = c_j | \theta) P(x_i | y_i = c_j; \theta)). \quad (3.8)$$

Karena saya kurang paham fungsi nya pada bagian $\sum_{x_i \in X_u} \log \sum_{j \in [M]} P(c_j | \theta) P(x_i | c_j; \theta)$

```
def run_semisupervisedNB(X_train, y_train, X_unlabelled, X_testing):
    model = MultinomialNB()
    model.fit(X_train, y_train)

    new_X = pd.concat([X_train, X_unlabelled])
    new_y = np.concatenate((y_train, model.predict(X_unlabelled)))

    model = MultinomialNB()
    model.fit(new_X, new_y)

    return model.predict(X_testing)

food_pred = run_semisupervisedNB(X_train, y_food, X_unlabelled, X_testing)
```

Bagian 4. Logistic Regression

Algoritma Logistic Regression yang dipakai disini adalah Multinomial Logistic Regression yang memiliki formula sebagai berikut untuk parameter berjumlah tiga, yaitu positif, negatif, dan null:

$$P = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$$

P merupakan probabilitas dari $Y = 1$ (Bernoulli variable). β_0 , β_1 , dan β_2 adalah parameter dari model. x_1 dan x_2 merupakan *predictor*.

Pada implementasinya sendiri, saya menggunakan Logistic Regression yang diimport dari *module* sklearn. Sedangkan untuk memprediksi probabilitas saya menggunakan fungsi `predict_proba()` yang dimiliki oleh Logistic Regression ini.

Langkah-langkah yang diterapkan secara garis besar pada model ini adalah sebagai berikut:

1. Melakukan training secara supervised terhadap labeled data.
2. Menyiapkan `X_train` dan `y_train` yang diperoleh dari labeled data serta unlabeled data.

3. Membuat variabel untuk menyimpan prediksi dan probabilitas dari unlabeled data.
4. Melakukan iterasi selama masih terdapat unlabeled data dengan probabilitas dari prediksi yang besar (disini saya menggunakan *threshold* 99%.
5. Menggabungkan unlabeled data dengan probabilitas melebihi *threshold* ke dalam labeled data.
6. Setelah tidak ada lagi unlabeled data yang memiliki probabilitas tinggi, dengan kata lain *f1_score* tidak dapat di-*improve* lebih lanjut maka iterasi dihentikan.

Bagian 5. Neural Network

Arsitektur Neural Network yang saya buat untuk tugas ini adalah sebagai berikut:

Input Layer = 2593 node

Hidden Layer = 512 node

Output layer = 3 node

Output layer berdimensi 3 karena bencana menggunakan layer terakhir sebagai softmax layer dan menggunakan categorical cross entropy sebagai fungsi loss nya.

Oleh karena itu, data label y juga harus disesuaikan.

```
1 def baseline_model():
2     # Create model here
3     model = Sequential()
4     model.add(Dense(512, input_dim = input_dim, activation = 'relu'))
5     model.add(Dense(output_dim, activation = 'softmax'))
6     # Compile model here
7     model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
8     return model

1 model = baseline_model()
2 model.summary()

Model: "sequential"

```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	1328128
dense_1 (Dense)	(None, 3)	1539

```

Total params: 1,329,667
Trainable params: 1,329,667
Non-trainable params: 0

```

	price_pol	Pos	Net	Neg
0	0	0	1	0
1	0	0	1	0
2	0	0	1	0
3	0	0	1	0
4	-1	0	0	1
...
3858	0	0	1	0
3859	1	1	0	0
3860	0	0	1	0
3861	0	0	1	0
3862	0	0	1	0

Untuk implementasi semi supervised, saya melakukan training model dengan data berlabel, kemudian model itu digunakan untuk memprediksi data unlabelled. Hasil

prediksi tersebut berbentuk probabilitas relatif (softmax) masing-masing class terhadap kelas lain, yang dapat digunakan untuk melatih kembali modelnya. Saya tidak mengimplementasikan looping karena waktu training data gabungan labelled + unlabelled cukup lama, menghabiskan waktu sekitar 20 menit dengan jumlah epochs = 20 dan validation 5 kali. Dan jika saya melakukan training lagi terhadap data gabungan, terjadi error karena memory yang digunakan melebihi batas yang ada di google collab.

```
def run_semisupervised(X_train, y_train, X_unlabelled, X_testing):
    model = KerasClassifier(build_fn = baseline_model, epochs = 20, batch_size = 5, verbose = 0)
    model.fit(X_train.to_numpy(), y_train)

    new_X = pd.concat([X_train, X_unlabelled])
    new_y = np.concatenate((y_train, model.predict_proba(X_unlabelled.to_numpy())))

    model = KerasClassifier(build_fn = baseline_model, epochs = 20, batch_size = 5, verbose = 0)
    model.fit(new_X.to_numpy(), new_y)

    return model.predict(X_testing.to_numpy())
```

Bagian 6. Kesimpulan

Berikut adalah tabel skor masing-masing model.

Tabel Skor Aspect Only

	Precision	Recall	F1-score
Multinomial NB	80.41	93.25	83.77
Logistic Regression			
Neural Network	92.25	88.41	88.09

Tabel Skor Aspect-Sentiment Pair

	Precision	Recall	F1-score
--	-----------	--------	----------

Multinomial NB	60.37	69.58	62.75
Logistic Regression			
Neural Network	74.66	71.12	71.00

Bisa dilihat bahwa pada Aspek-Only, Naive Bayes lebih baik dari Neural Network, namun sebaliknya pada Recall, dan akhirnya F1-Score Neural Network lebih tinggi.

Pada Aspect-sentiment pair, Neural Network mengungguli Naive Bayes dari setiap metrik scoring.

Referensi:

- [1] <https://towardsdatascience.com/effectively-pre-processing-the-text-data-part-1-text-cleaning-9ecae119cb3e>
- [2] Chapelle, Scholkopf & Zien: Semi-Supervised Learning
- [3] <https://towardsdatascience.com/a-gentle-introduction-to-self-training-and-semi-supervised-learning-ceee73178b38>