Dokumen Tugas Akhir

Natural Language Processing

Visualisasi Hasil Ekstraksi Kata Kunci

Ame Fedora Ignacia Ginting 6181801047

Rama Fauzi Setiawan 6181801051

I. Latar Belakang

Umumnya, kata kunci adalah komponen yang penting untuk merepresentasikan konten sebuah artikel dengan tepat. Tak hanya itu, kata kunci juga biasanya berperan penting pada saat menemukan artikel dari sebuah sistem pencarian informasi, basis data bibliografi, dan optimasi mesin pencari. Kata kunci juga membantu kita untuk dapat mengkategorikan artikel ke dalam subjek atau disiplin ilmu yang relevan.

Kata kunci yang biasa kita temukan dalam artikel merupakan hasil ekstraksi dari keseluruhan kata yang ada di dalam artikel tersebut. Proses ekstraksi kata dari artikel awalnya berdasarkan topik besar dari artikel tersebut atau opini dari penulis saja. Namun dengan kemajuan zaman, dengan teknik *Natural Language Processing*, dapat dihasilkan kata kunci yang lebih akurat dan waktu yang diperlukan untuk mendapatkan kata kunci pun lebih singkat.

Dalam eksperimen ini, kami akan menyelesaikan permasalahan untuk mengambil kata kunci dari gabungan beberapa *abstract* artikel yang kemudian akan divisualisasikan menggunakan *chart python*. Dalam prosesnya, kami akan mengikuti cara penyelesaian yang dilakukan oleh penulis dan menggunakan dataset dari *kaggle* yang disediakan oleh penulis.

Berdasarkan cara yang dijabarkan oleh penulis, tahap awal yang harus dikerjakan adalah *Text-Processing* yaitu berupa pembersihan *noise* dan normalisasi teks(*Lemmatization*). Tahap kedua, *Data Exploration* berupa pembuatan *word cloud* dan pembuatan *uni-gram, bi-gram,* dan *tri-gram.* Tahap ketiga, pembuatan vektor dari *n-gram* yang telah dibuat untuk selanjutnya divisualisasikan menjadi *chart*. Tahap terakhir yang dilakukan oleh penulis adalah membuat TF-IDF yang digunakan untuk menentukan kata kunci dari sebuah artikel saja.

Dari cara yang telah dijabarkan diatas, kami akan melakukan modifikasi proses pada bagian pembersihan *noise* dengan menghapus data yang memiliki "*Abstract Missing*" dan hanya mengambil kolom ID, Year, dan Abstract1(Title+Abstract). Modifikasi lain yang kami lakukan adalah membuat *stop-word* dari *Common Word* yang didapat melalui *Data Exploration* dan melakukan *FetchWord* tanpa fungsi *lambda*.

II. Studi Literatur

Dalam eksperimen pencarian kata kunci sebuah artikel, kami menggunakan beberapa metode dan algoritma sebagai berikut :

a. Lemmatization

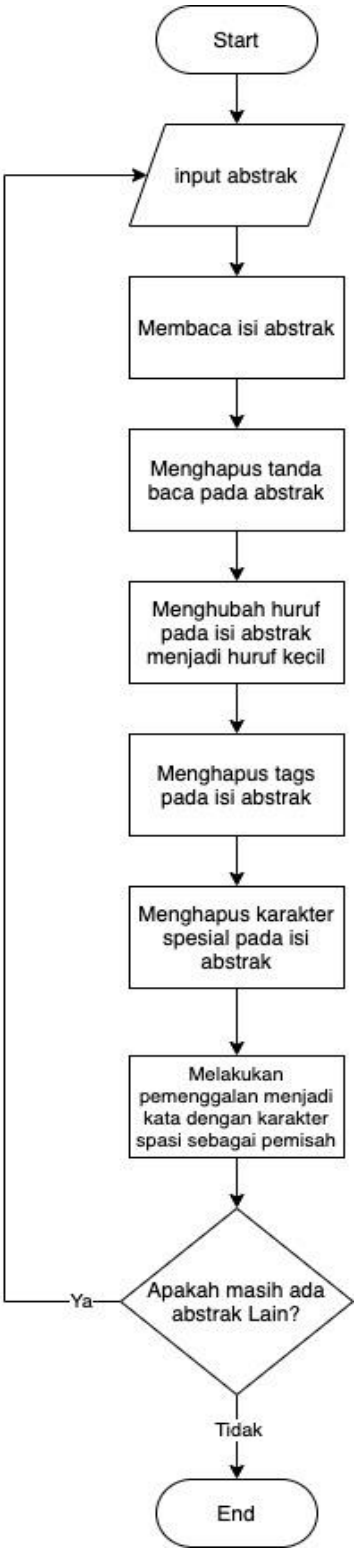
Lemmatization merupakan salah satu teknik pada pemrosesan Bahasa alami yang menggunakan kosakata dan morfologis kata-kata yang biasanya bertujuan untuk menghilangkan ujung inflektif saja dan mengembalikan kata kepada kata dasarnya atau bentuk kamus dari sebuah kata, yang dikenal sebagai lemma. *Lemmatization* digunakan pada kebutuhan yang berhubungan degan *text mining* seperti *information retrieval* yang dilakukan pada tahap *pre-processing*. Berikut adalah contoh kata dan hasil lemmanya dalam bahasa Inggris.

Form	Informasi Morfologi	Lemma
Studies	Kata ganti orang ketiga, bentuk *present tense* dari kata kerja study	Study
Studying	Bentuk *present continuous tense* dari kata kerja study	Study

Lemmitization menggunakan konteks dan pustaka leksikal untuk mendapatkan lemma sehingga lemma akan selalu menjadi kata yang valid.

b. Tokenization

Tokenization merupakan proses pemecahakan kalimat atau paragraf menjadi himpunan kata. Proses ini dilakukan dengan melakukan iterasi berdasarkan jumlah kata pada kalimat output dari proses sebelumnya. Proses *Tokenization* biasanya berisi beberapa proses seperti proses penghapusan tanda baca, proses pengubahan huruf pada himpunan kata menjadi huruf kecil, dan proses pengubahan atau penghapusan karakter khusus sesuai dengan kebutuhan. Output dari proses *Tokenization* terbagi menjadi 3 yaitu dapat berupa kata, karakter, dan subkata (n-gram karakter). Berikut adalah *flowchart* dari proses Tokenization pada eksperimen kami :


Start
input abstrak
Membaca isi abstrak
Menghapus tanda baca pada abstrak
Menghubah huruf pada isi abstrak menjadi huruf kecil
Menghapus tags pada isi abstrak
Menghapus karakter spesial pada isi abstrak
Melakukan pemenggalan menjadi kata dengan karakter spasi sebagai pemisah
Apakah masih ada abstrak Lain?
Ya
Tidak
End

c. Vektorisasi

Vektorisasi merupakan proses mengubah daftar kata menjadi matriks bilangan bilangan real yang sesuai. Vektorisasi juga biasa disebut sebagai fitur ekstraksi. Pada eksperimen ini kami menggunakan 2 jenis vektorisasi yaitu :

i. N-gram Model Vectorization

N-gram adalah suatu metode pengolahan dokumen yang melakukan proses pemecahan himpunan kata berdasarkan urutan n-word dimana n adalah nilai pasangan token. N-gram model biasanya digunakan dalam *spelling correction, word prediction* dan pengolahan teks lainnya.

Dalam eksperimen ini n-gram akan digunakan sebagai metode yang mencari karakteristik dari masing-masing dokumen yang akan menghasilkan suatu *language model. Language model* ini berisi data frekuensi penggunaan data pada suatu dokumen. Data tersebut terdiri dari n-gram per kata seperti unigram (satu karakter), bigram (dua karakter) dan trigram (tiga karakter). Berikut adalah contoh kalimat dan hasil n-gram modelnya.

Contoh Kalimat :
“Budi pergi ke pasar menemani ibunya”

Hasil unigram (1-gram) :
[“budi”, “pergi”, “ke”, “pasar”, “menemani”, “ibunya”]

Hasil bi-gram (2-gram) :
[“budi pergi”, “pergi ke”, “ke pasar”, “pasar menemani, “menemani ibunya”]

Hasil tri-gram (3-gram) :
[“budi pergi ke”, “pergi ke pasar”, “ke pasar menemani”, “pasar menemani ibunya”]

ii. Term Frequency Inverse Document Frequency (TF-IDF)

TF-IDF merupakan metode yang digunakan untuk menganalisa hubungan antara sebuah frase/kalimat dengan sekumpulan dokumen. Metode yang bekerja dengan menggabungkan dua konsep untuk perhitungan bobot, yaitu frekuensi kemunculan sebuah kata didalam sebuah dokumen tertentu (TF) dan inverse frekuensi dokumen (IDF) yang mengandung kata tersebut. Dalam hal ini dokumen adalah vektor, kata adalah dimensi, dan TF-IDF merupakan nilai dari setiap dimensi. TF digunakan untuk melihat kemungkinan kemunculan terms tertentu dalam sebuah dokumen, sedangkan IDF digunakan untuk mengetahui seberapa pentingnya sebuah kata. Semakin besar bobot TF-IDF dari sebuah kata, maka semakin penting juga sebuah kata tersebut.

Berikut adalah rumus dasar dari algoritma TF-IDF :

t = Term
d = Document
D = Corpus/Document Set

$$TF(t,d) = \frac{\text{count of t occurrence in d}}{\text{number of terms in d}}$$

$$IDF(t,D) = \log \frac{\text{total D}}{\text{number of document in D containing t}}$$

$$TF\text{-}IDF(t, d, D) = TF(t,d) \times IDF(t,D)$$

d. Visualisasi Data

Visualisasi data merupakan tampilan berupa grafis atau visual sebagai informasi dan data. Pada eksperimen ini kami menggunakan visualisasi data untuk menampilkan kumpulan data yang telah kami proses agar menjadi lebih sederhana untuk ditampilkan. Dalam hal ini, visualisasi yang kami gunakan dalah *barplot* dengan *chart python*.

III. Rancangan

a. Dataset

Dataset asli berasal dari kaggle yang berisi *titles, authors, abstracts*, dan *extracted text* dari seluruh NIPS papers. Neural Information Processing Systems (NIPS) adalah salah satu konferensi Machine Learning terbaik di dunia. Topik yang diambil mulai dari Deep Learning dan Computer Vision hingga Cognitive Science dan Reinforcement Learning.

(Link Data Set : https://www.kaggle.com/benhamner/nips-papers)

Dataset yang digunakan pada eksperimen ini tidak menggunakan semua csv yang ada, melainkan hanya menggunakan papers.csv. Nantinya dataset papers.csv tersebut akan dipangkas lagi untuk mendapatkan dataset yang digunakan untuk eksperimen. Eksperimen ini hanya memanfaatkan kolom id, year, dan abstract1(Yang dibuat dengan menggabungkan judul dan abstract masing-masing pdf) dari papers.csv.

b. Library

Library yang digunakan dalam eksperimen ini terdiri dari:

- pandas
- re
- nltk
- sklearn
- PIL
- wordcloud
- matplotlib.pyplot
- seaborn
- scipy

c. Alur Program

Pada Tahap awal, kami melakukan pemangkasan pada dataset dengan memanfaatkan kode sebagai berikut:

```
dataset = pandas.read_csv('papers.csv')

#data = df.drop(labels='Abstract Missing', axis=1)
data = dataset.loc[dataset["abstract"] != "Abstract Missing"]

#Membuat dataframe yang digunakan berisi id, year, abstract1(title+abstract)
year = data['year']
abstract1 = data['title'] + ' ' + data['abstract']

df = abstract1.rename('abstract1').to_frame()

df.insert(0, "year", year, True)
df.insert(0, "id", data["id"], True)

df.reset_index(drop=True, inplace=True)
```

Hasil dari kode diatas ketika dijalankan akan menghasilkan dataframe seperti dibawah ini:

Index	id	year	abstract1
0	1861	2000	minimizes the generalized Kullback-Leibler div…
1	1975	2001	Characterizing Neural Gain Control using Spike…
2	3163	2007	Competition Adds Complexity It is known that d…
3	3164	2007	Efficient Principled Learning of Thin Junction…
4	3167	2007	Regularized Boost for Semi-Supervised Learning…
5	3168	2007	Simplified Rules and Theoretical Analysis for …
6	3169	2007	Predicting human gaze using low-level saliency…
7	3171	2007	Mining Internet-Scale Software Repositories La…
8	3172	2007	Continuous Time Particle Filtering for fMRI We…
9	3174	2007	An online Hebbian learning rule that performs …
10	3176	2007	Discriminative K-means for Clustering We prese…
11	3178	2007	The Epoch-Greedy Algorithm for Multi-armed Ban…

Setelah dataset yang akan kami gunakan sudah siap, tahap selanjutnya yang kami lakukan adalah mengubah text abstracts1 yang ada menjadi word count, common word, dan uncommon word. Tahapan mengubah text menjadi word count ini diperuntukan untuk menghitung jumlah seluruh kata dari masing-masing text abstract1 tersebut. Nantinya hasil dari word count tersebut akan dimasukkan ke dataframe yang sudah

kami buat sebelumnya. Kode untuk mengubah text abstract1 menjadi word count sebagai berikut:

```
#Fetch wordcount for each abstract
def countWord(text):
    count = len(str(text).split(" "))
    return count

data2 = []

for i in range(len(df)):
    data2.append(countWord(df['abstract1'][i]))

dataWordCount = pandas.DataFrame(data2, columns=['word_count'])

df.insert(3, "word_count", dataWordCount, True)
```

Hasil dari kode diatas ketika dijalankan akan menghasilkan dataframe baru seperti dibawah ini:

Index	id	year	abstract1	word_coun
0	1861	2000	shown to minimize the conventional least squar… minimizes the generalized Kullback-Leibler div… convergence of both algorithms can be proven …	112
1	1975	2001	Characterizing Neural Gain Control using Spike…	88
2	3163	2007	Competition Adds Complexity It is known that d…	70
3	3164	2007	Efficient Principled Learning of Thin Junction…	150
4	3167	2007	Regularized Boost for Semi-Supervised Learning…	124
5	3168	2007	Simplified Rules and Theoretical Analysis for …	169
6	3169	2007	Predicting human gaze using low-level saliency…	208
7	3171	2007	Mining Internet-Scale Software Repositories La…	194
8	3172	2007	Continuous Time Particle Filtering for fMRI We…	108
9	3174	2007	An online Hebbian learning rule that performs …	113
10	3176	2007	Discriminative K-means for Clustering We prese…	192
11	3178	2007	The Epoch-Greedy Algorithm for Multi-armed Ban…	77
12	3186	2007	Local Algorithms for Approximate Inference in …	176

Berdasarkan dari nilai word count yang sudah didapat, dibuat perhitungan mengenai basic statistical details dengan memanfaatkan fungsi .describe() yang dimiliki pandas, nilai-nilai ini berfungsi untuk melihat persebaran kata-katanya dan melihat ukuran dari data yang akan kami jalankan.

```
#Descriptive statisStics of word counts
df.word_count.describe()
```

Hasil yang didapat adalah sebagai berikut:

```
Out[4]:
count    3924.000000
mean      155.888124
std        46.001025
min        27.000000
25%       122.000000
50%       151.000000
75%       185.000000
max       325.000000
Name: word_count, dtype: float64
```

Setelah dibuat dataframe dan diketahui persebaran datasetnya, tahap berikutnya yang kami lakukan adalah membuat common word dan uncommon yang nantinya akan digunakan untuk membuat stop word pada proses text-preprocessing. Kode yang kami gunakan untuk membuat common word dan uncommon word dari dataframe sebelumnya adalah sebagai berikut:

```
#Identify common words
freq = pandas.Series(' '.join(df['abstract1']).split()).value_counts()[:20]

#freq to dict
newFreq = freq.to_dict()

#Identify uncommon words
freq1 =  pandas.Series(' '.join(df['abstract1']).split()).value_counts()[-20:]
```

Hasil dari common word dan uncommon word dari kode diatas adalah sebagai berikut:

Common word

```
In [34]: newFreq
Out[34]:
{'the': 30107,
 'of': 21685,
 'a': 16518,
 'and': 14218,
 'to': 13104,
 'in': 9454,
 'for': 8382,
 'that': 7841,
 'is': 7687,
 'We': 6239,
 'on': 5704,
 'we': 5167,
 'with': 5142,
 'as': 3686,
 'this': 3677,
 'are': 3546,
 'an': 3398,
 'by': 3302,
 'can': 2958,
 'learning': 2903}
```

Uncommon word

```
In [36]: freq1
Out[36]:
moreover                                                    1
subsystems.                                                 1
NGD                                                         1
$\operatorname{poly}(\frac{1}{\epsilon})$                   1
Multi-Response                                              1
LPboosting.                                                 1
$P_X$,                                                      1
gather,                                                     1
n}{\varepsilon^2}\}\right).$                                1
`estimate'                                                  1
${0}(ns)$                                                   1
compressing                                                 1
$|z|$                                                       1
inaccessibility                                             1
http://www.bioinf.jku.at/software/rfn.                      1
Hearts,                                                     1
Szlam,                                                      1
undercomplete                                               1
recommend,                                                  1
widely-applicable                                           1
dtype: int64
```

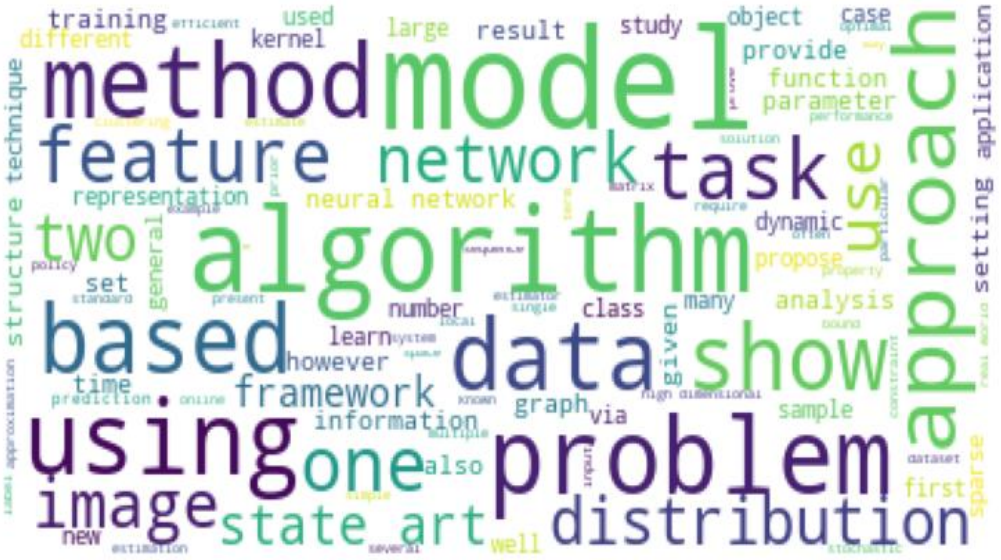
Selanjutnya kami mulai memasuki tahap text pre-processing. Tahap pertama yang kami lakukan adalah membuat stop word. Stop word didapatkan dengan memanfaatkan stopword bahasa inggris yang sudah ada dari library nltk, ditambah dengan common word yang sudah kami dapatkan dari tahap sebelumnya. Kode pembuatan stop word sebagai berikut:

```
#Creating a list of stop words and adding custom stopwords
stop_words = set(stopwords.words("english"))
#Creating a list of custom stopwords
new_words = newFreq.keys()
stop_words = stop_words.union(new_words)
```

Setelah stop word dibuat, tahap selanjutnya pembuatan corpus. Corpus ini yang nantinya akan digunakan untuk mengambil seluruh text yang sudah dibersihkan dari masing-masing abstract. Corpus yang dibuat pada eksperimen akan melewati beberapa tahap yaitu menghapus tanda baca, lowercase, menghapus tags, menghapus special character dan digit, mengubah string ke string, Stemming, dan Lemmatization. Kode yang digunakan untuk membuat corpus tersebut adalah sebagai berikut:

```
corpus = []
for i in range(0, 3924):
    #Remove punctuations
    text = re.sub('[^a-zA-Z]', ' ', df['abstract1'][i])

    #Convert to lowercase
    text = text.lower()

    #remove tags
    text=re.sub("&lt;/?.*?&gt;"," &lt;&gt; ",text)

    # remove special characters and digits
    text=re.sub("(\\d|\\W)+"," ",text)

    ##Convert to list from string
    text = text.split()

    ##Stemming
    ps=PorterStemmer()
    #Lemmatisation
    lem = WordNetLemmatizer()
    text = [lem.lemmatize(word) for word in text if not word in
            stop_words]
    text = " ".join(text)
    corpus.append(text)
```

Corpus yang sudah dibuat sebelumnya, akan dibuatkan visualisasi menggunakan Library wordcloud untuk melihat perbandingan jumlah masing-masing kata. Semakin besar ukuran kata tersebut, semakin banyak jumlah kata tersebut.



Kode yang digunakan untuk membuat wordcloud diatas sebagai berikut:

```
#Word cloud
wordcloud = WordCloud(
                          background_color='white',
                          stopwords=stop_words,
                          max_words=100,
                          max_font_size=50,
                          random_state=42
                          ).generate(str(corpus))
print(wordcloud)
fig = plt.figure(1)
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
fig.savefig("word1.png", dpi=900)
```

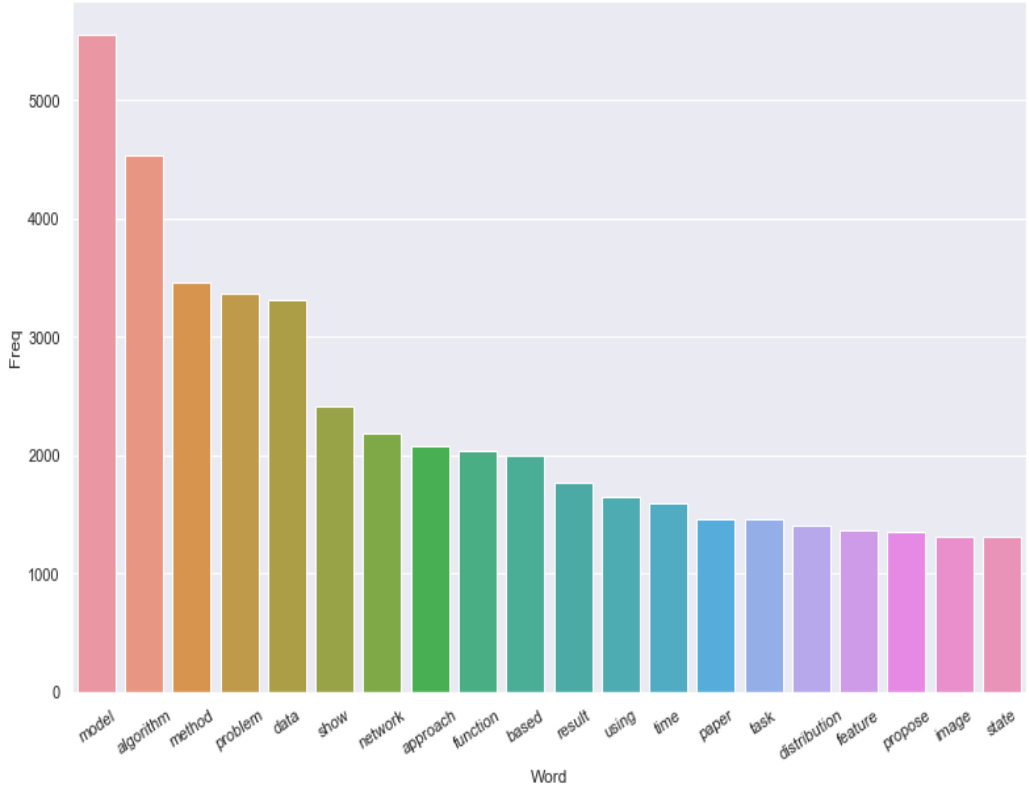
Tahap selanjutnya adalah pembuatan Vector word count. Pembuatan vector ini memanfaatkan Library Sklearn. Hasil dari Vector ini nantinya akan diubah menjadi visualisasi berupa uni-gram, bi-gram, dan tri-gram. Kode untuk pembuatan vektor ini sebagai berikut:

```
cv=CountVectorizer(max_df=0.8,stop_words=stop_words, max_features=10000, ngram_range=(1,3))
X=cv.fit_transform(corpus)

list(cv.vocabulary_.keys())[:10]
```

Untuk pembuatan visualisasi ini, kami memanfaatkan Library Seaborn. Nantinya visualisasi akan terbagi menjadi tiga gambar, unigram, bigram, dan trigram. Masing-masing dari pembuatan gambar tersebut menggunakan kode yang hampir sama, hanya mengubah parameter n-gram nya.

```
#Most frequently occuring words
def get_top_n_words(corpus, n=None):
    vec = CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in
                  vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1],
                       reverse=True)
    return words_freq[:n]

#Convert most freq words to dataframe for plotting bar plot
top_words = get_top_n_words(corpus, n=20)
top_df = pandas.DataFrame(top_words)
top_df.columns=["Word", "Freq"]
#print(top_df)

#Barplot of most freq words
sns.set(rc={'figure.figsize':(13,8)})
g = sns.barplot(x="Word", y="Freq", data=top_df)
g.set_xticklabels(g.get_xticklabels(), rotation=30)
plt.show()
```

Hasil dari kode pembuatan visualisasi unigram diatas sebagai berikut:



Setelah pembuatan untuk visualisasi unigram, bigram, dan trigram selesai, tahap selanjutnya adalah pembuatan tf-idf dari vector yang ada untuk menjalankan tahap berikutnya yaitu mengeluarkan keyword dari masing-masing abstract1 dari dataframe. Kode untuk pembuatan tf-idf sebagai berikut:

```
tfidf_transformer=TfidfTransformer(smooth_idf=True,use_idf=True)
tfidf_transformer.fit(X)
# get feature names
feature_names=cv.get_feature_names()

# fetch document for which keywords needs to be extracted
doc=corpus[532]

#generate tf-idf for the given document
tf_idf_vector=tfidf_transformer.transform(cv.transform([doc]))
```

Tf-idf yang sudah dibuat nantinya akan diurutkan berdasarkan dari yang paling besar nilainya hingga paling kecil. Hal diatas kami lakukan dengan membuat fungsi yang nantinya akan dipanggil saat pembuatan keywords.

```
#Function for sorting tf_idf in descending order
def sort_coo(coo_matrix):
    tuples = zip(coo_matrix.col, coo_matrix.data)
    return sorted(tuples, key=lambda x: (x[1], x[0]), reverse=True)
```

Fungsi selanjutnya yang digunakan untuk melengkapi kode diatas adalah dengan membuat fungsi untuk mengeluarkan hasil top ke-n dari tf-idf yang sudah terurut menurun sebelumnya.

```
def extract_topn_from_vector(feature_names, sorted_items, topn=10):
    """get the feature names and tf-idf score of top n items"""

    #use only topn items from vector
    sorted_items = sorted_items[:topn]

    score_vals = []
    feature_vals = []

    # word index and corresponding tf-idf score
    for idx, score in sorted_items:

        #keep track of feature name and its corresponding score
        score_vals.append(round(score, 3))
        feature_vals.append(feature_names[idx])

    #create a tuples of feature,score
    #results = zip(feature_vals,score_vals)
    results= {}
    for idx in range(len(feature_vals)):
        results[feature_vals[idx]]=score_vals[idx]

    return results
```

Pada tahap terakhir, program akan mengambil text dari corpus yang dipilih. Text yang diambil dari corpus tersebut akan dibentuk vector tf-idf nya, kemudian dijalankan fungsi sort_coo dan extract_topn_from_vector. Untuk lebih jelasnya penggunaan dari kedua fungsi diatas, dapat dilihat pada kode berikut:

```
#sort the tf-idf vectors by descending order of scores
sorted_items=sort_coo(tf_idf_vector.tocoo())
#extract only the top n; n here is 10
keywords=extract_topn_from_vector(feature_names,sorted_items,5)
```

Hasil akhir dari program ini adalah abstract yang dipilih dan keywords yang dihasilkan. Contoh menggunakan corpus[532]

```
Abstract:
compositionality optimal control law present theory compositionality stochastic optimal
control showing task optimal controller constructed certain primitive primitive feedback
controller pursuing agenda mixed proportion much progress making towards agenda compatible
agenda present task resulting composite control law provably optimal problem belongs certain
class class rather general yet number unique property one bellman equation made linear even
non linear discrete dynamic give rise compositionality developed special case linear dynamic
gaussian noise framework yield analytical solution e non linear mixture linear quadratic
regulator without requiring final cost quadratic generally natural set control primitive
constructed applying svd green function bellman equation illustrate theory context human arm
movement idea optimality compositionality prominent field motor control yet hard reconcile
work make possible

Keywords:
compositionality 0.397
control 0.293
primitive 0.27
linear 0.195
optimal control 0.193
```

d. Modifikasi

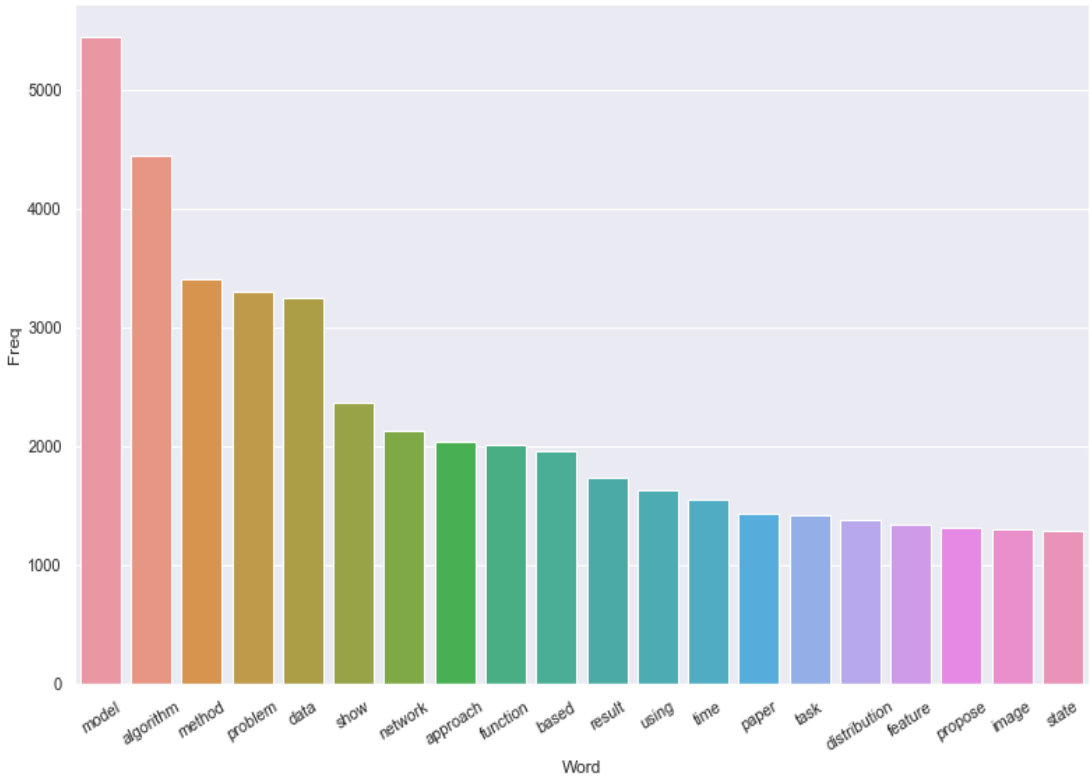
Pada eksperimen ini, beberapa hal yang menjadi modifikasi kami pada kode asli dari referensi tugas adalah sebagai berikut:

- Pengambilan dan pembersihan dataset dari dataset asli
- Pembuatan stop word menggunakan custom stop word
- Pembuatan word count secara manual tanpa Library

2. Hasil Analisis dan Kesimpulan

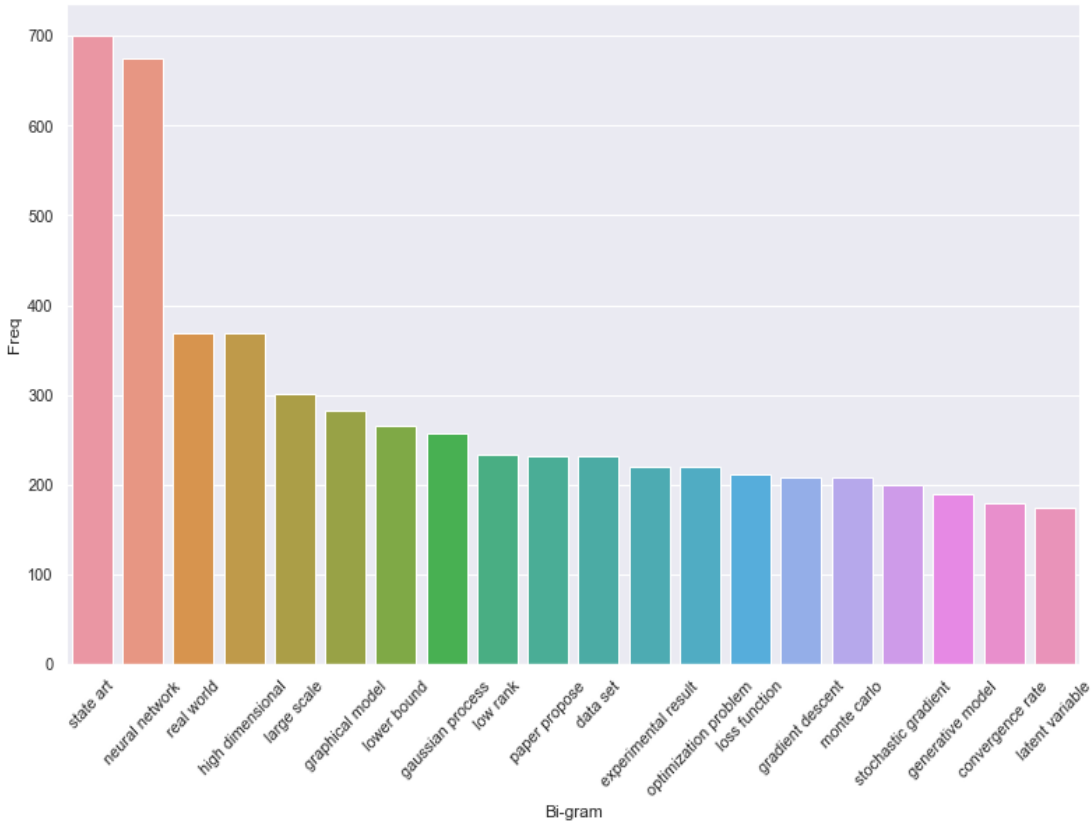
Dari hasil eksperimen yang sudah kami lakukan di atas, kami mendapatkan hasil berupa visualisasi unigram, bigram, dan trigram terhadap seluruh dataset yang digunakan serta hasil generate keywords menggunakan tf-idf sesuai dengan text yang dipilih.

Bentuk visualisasi pertama yang kami dapat adalah bentuk unigram dari vector kata yang didapat menggunakan countVectorizer.



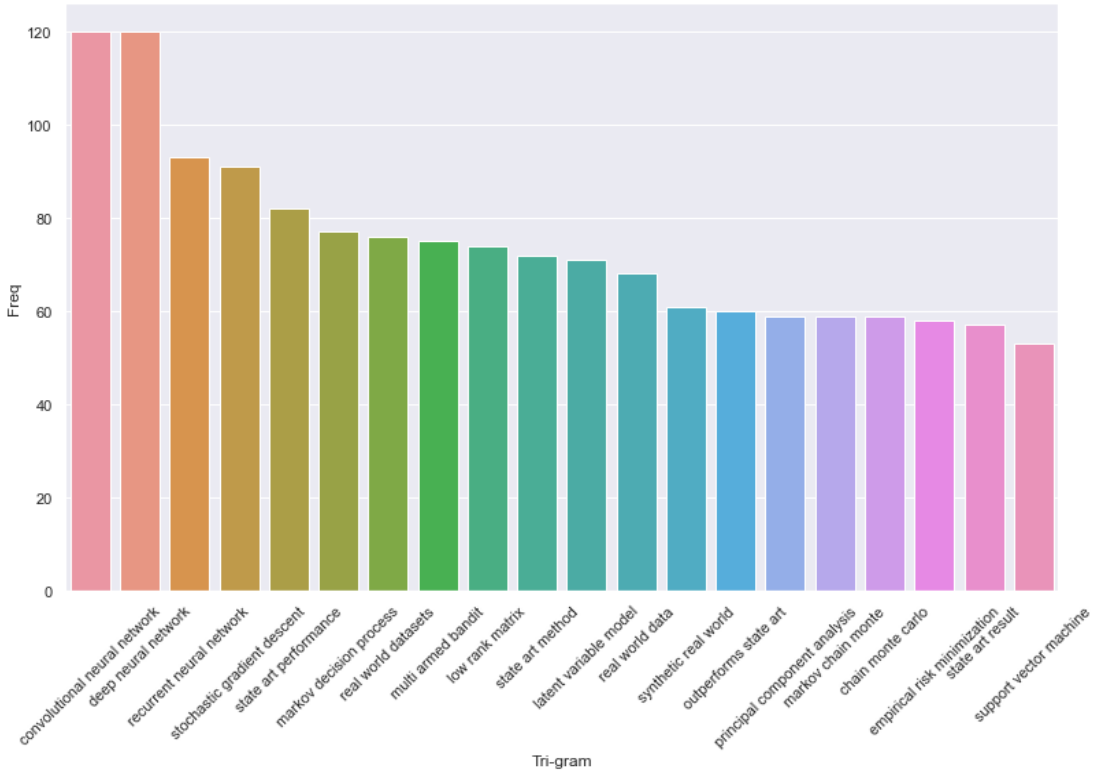
Dari hasil visualisasi di atas dapat disimpulkan bahwa dari dataset text NISP Papers (Judul dan Abstract), kata “modul” menjadi kata yang paling sering muncul dalam keseluruhan text. Menandakan bahwa terdapat banyak pembahasan modul dalam setiap text nya.

Bentuk visualisasi kedua yang kami dapat adalah bentuk bigram dari vector kata yang didapat menggunakan countVectorizer.



Berdasarkan hasil visualisasi di atas dapat disimpulkan bahwa dari dataset text NISP Papers (Judul dan Abstract), bigram “state art” menjadi yang paling banyak muncul di dalam keseluruhan text. Namun, disamping itu, terdapat bigram “neural network” yang jumlahnya hampir sama dengan bigram “state art”. Melihat hasil antara unigram dan bigram, kami menyimpulkan bahwa kata “modul” kemungkinan terdapat dalam satu text yang sering menyebutkan kata “modul” sehingga barplot yang dihasilkan tinggi dan tidak seimbang dengan proporsi kata lainnya di setiap text nya.

Bentuk visualisasi ketiga yang kami dapat adalah bentuk tri.gram dari vector kata yang didapat menggunakan countVectorizer.



Berdasarkan hasil visualisasi di atas dapat disimpulkan bahwa dari dataset text NISP Papers (Judul dan Abstract), trigram “Convolutional neural network” dan “deep neural network” menjadi yang paling banyak muncul dalam keseluruhan text.

Dari hasil unigram, bigram, dan trigram dari ketiga visualisasi diatas, dapat dilihat bahwa ketiga nilai teratas dari masing-masing n-gram berbeda-beda dan sama sekali tidak mengandung kata-kata yang sama. Namun jika diperhatikan kembali, pada visualisasi bigram, “neural network” menjadi bigram yang paling banyak disebut setelah “state art”. Pada visualisasi trigram, bigram “neural network” terdapat di dalam kedua trigram teratasnya, “Convolutional neural network” dan “deep neural network”. Sehingga kami menarik kesimpulan bahwa dataset text(Title dan Abstract) NIPS Papers, kebanyakan membahas tentang “neural network”.