2501961994 / Andrew Widjaya

Text Mining Mid Exam Explanation

1. Data Collection (Code = WebScraping_from_RawUrl.ipynb)

Data Collection dilakukan dengan menggunakan scraping website berita yang bersangkutan dengan menggunakan library BeautifulSoup. BeautifulSoup akan bekerja dengan membaca data url yang diberikan, kemudian akan diparse menjadi bentuk html.

```
#Parsing HTML using BeautifulSoup
soup = BeautifulSoup(request.text, 'html.parser')
```

Setelah di parse menjadi HTML , kemudian dilakukan pencarian class yang merepresentasikan isi teks berita, disini diperlukan inspect element di setiap media berita, sebab setiap media berita memiliki class yang berbeda – beda. Kemudian, setelah class didefinisikan, kita akan mencari semua isi berita dan memasukkannya ke dalam list dan kemudian akan dimasukkan menjadi sebuah dataFrame. Disini dipastikan juga bahwa label balance, dimana masing-masing memiliki 150 label (politik, hiburan, olahraga), menghasilkan total 450 data

```
#Cari class yang merepresentasikan isi berita keseluruhan
if media == "Detik.com":
  _class = "detail__body-text itp_bodycontent"
elif media == "CNN":
  _class = "detail-text text-cnn_black text-sm grow min-w-0"
elif media == "Kompas":
  _class = "read__content"
elif media == "Tempo":
  _class = "detail-konten"
elif media == "Liputan6":
  _class = "article-content-body__item-content"

div = soup.find('div',_class)

df = pd.DataFrame(columns=['Teks','Media','Label'])

#Cari semua isi berita, dimana berada pada tag <p> pada website berita
texts = div.find_all('p')
content = []

#Masukkan ke dalam list content
for text in texts:
  content.append(text.get_text())

#Masukkan ke dalam dataFrame dan direturn
if content:
  df = pd.concat([pd.DataFrame({'Teks':['\n'.join(content)],'Media':media,'Label':label},columns=df.columns),df])

return df
```

Setelah itu, semua data url yang sudah dicari secara manual, akan dimasukkan ke dalam dataFrame, kemudian akan dimasukkan ke dalam function scraping di atas, kemudian akan diconcat ke dalam 1 buah dataFrame yang dinamakan raw_data

```
raw_data = pd.DataFrame(columns=['Teks','Media','Label'])

Detik_Olahraga = pd.read_csv("Detik_Olahraga.csv")
Detik_Politik = pd.read_csv("Detik_Politik.csv")
Detik_Hiburan = pd.read_csv("Detik_Hiburan.csv")
Kompas_Olahraga = pd.read_csv("Kompas_Olahraga.csv")
Kompas_Politik = pd.read_csv("Kompas_Politik.csv")
Kompas_Hiburan = pd.read_csv("Kompas_Hiburan.csv")
Tempo_Politik = pd.read_csv("Tempo_Politik.csv")
Tempo_Olahraga = pd.read_csv("Tempo_Olahraga.csv")
Tempo_Hiburan = pd.read_csv("Tempo_Hiburan.csv")
CNN_Olahraga = pd.read_csv("CNN_Olahraga.csv")
CNN_Politik = pd.read_csv("CNN_Politik.csv")
CNN_Hiburan = pd.read_csv("CNN_Hiburan.csv")
Liputan6_Politik = pd.read_csv("Liputan6_Politik.csv")
Liputan6_Olahraga = pd.read_csv("Liputan6_Olahraga.csv")
Liputan6_Hiburan = pd.read_csv("Liputan6_Hiburan.csv")
```

```
for url in Detik_Olahraga.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Detik.com","Olahraga")])

for url in Detik_Politik.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Detik.com","Politik")])

for url in Detik_Hiburan.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Detik.com","Hiburan")])

for url in Kompas_Olahraga.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Kompas","Olahraga")])

for url in Kompas_Politik.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Kompas","Politik")])

for url in Kompas_Hiburan.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Kompas","Hiburan")])

for url in Tempo_Politik.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Tempo","Politik")])

for url in Tempo_Olahraga.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Tempo","Olahraga")])

for url in Tempo_Hiburan.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Tempo","Hiburan")])

for url in CNN_Olahraga.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "CNN","Olahraga")])
```

```
for url in CNN_Politik.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "CNN","Politik")])

for url in CNN_Hiburan.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "CNN","Hiburan")])

for url in Liputan6_Politik.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Liputan6","Politik")])

for url in Liputan6_Olahraga.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Liputan6","Olahraga")])

for url in Liputan6_Hiburan.Url:
  raw_data = pd.concat([raw_data,scrape_text(url, "Liputan6","Hiburan")])
```

Setelahnya, dataFrame raw_data akan diconvert menjadi csv untuk diproses lebih lanjut.

```
raw_data.to_csv("raw_data.csv")
```

2. Data Cleaning (Code = DataCleaning.ipynb)

Data Cleaning adalah pemrosesan teks agar teks sudah bersih dan tidak memiliki noise yang tidak diperlukan sebelum direpresentasikan dan dilakukan pemodelan. Data cleaning dilakukan menggunakan raw_data yang sudah dibuat sebelumnya. Kolom yang akan dicleaning adalah kolom “Teks” yang berisi isi berita

Beberapa step cleaning yang dilakukan pada kasus kali ini adalah

1. Case Folding

   Pada tahap ini, semua character akan diubah menjadi huruf kecil untuk disamaratakan dalam pemodelan.

   ```
   def cleaning(df,col_names):
     #Case Folding => Kecilkan semua kolom yang ada pada sebuah dataframe, nantinya untuk teks
     df[col_names] = df[col_names].str.lower()
     df[col_names] = df[col_names].apply(clean_text)
   ```

2. Text cleaning (function clean_text)

   Disini, function clean_text digunakan untuk membersihkan teks secara keseluruhan, dimana hal – hal yang dilakukan adalah:

   a. Membuang semua kata-kata yang mengandung website url, seperti http, https, dan .com

   b. Membuang semua kata-kata yang mengandung hashtag (#) dan tag (@)

c. Membuang semua tag html, ditakutkan terdapat tag html yang terambil ketika scraping
d. Membuang semua kata-kata yang non-alphabetic, sehingga teks akan tersisa character saja.
e. Mengganti newline(\n) dengan space, sebab tadi di web-scraping, newline digunakan untuk separate setiap tag <p> pada isi berita
f. Mengganti double space dengan single space, sebab sebelumnya di tahap e, jika newline diganti space, akan muncul konflik double space terjadi.
g. Stopword dan Custom Word Removal, yaitu membuang semua stopword(frequently used word, seperti dengan, ada, dll.) dan juga custom word yang didefinisikan.

    Custom word dicari dengan menggunakan trial and error, dimana setelah dicari, beberap website berita mengandung kata-kata dummy atau iklan yang perlu dihapus. Selain itu, disini juga dilakukan penghapusan kata-kata redaksi yang bersangkutan dari teks, seperti kompas, liputan, dll.

    Adapun untuk stopword , disini digunakan 2 buah corpus, yaitu pertama menggunakan corpus Data Science Indonesia (shoutout to Mr Satya from DSI) yang bernama kamus-stopword dari github, serta ditambahkan stopword_removal function dari Sastrawi library.
h. Stemming, yaitu proses untuk mengubah semua kata menjadi bentuk dasarnya, seperti “mengalahkan” menjadi “kalah”, “bermain” menjadi “main”, dll. Disini, library Sastrawi juga digunakan dengan menggunakan StemmerFactory.

Proses Code bisa dilihat di DataCleaning.ipynb, sebab untuk discreenshot akan menjadi jelek dan kecil, serta sulit dibaca. Hasil dari data cleaning akan dimasukkan ke dalam clean_data, dan akan dibuat csv baru yang akan digunakan dalam text representation dan modelling.

3. Text Representation & Modelling (TextRepresentation+Modelling.ipynb)

Dataset yang akan dipakai adalah clean_data.csv, yang dihasilkan dari proses DataCleaning sebelumnya. Sebelum membahas mengenai Text Representation yang digunakan, disini dilakukan sedikit feature transformation, yaitu Label sebagai target variable akan diubah menjadi integer dengan mapping. Kemudian dilakukkan train_test_split dengan rasio 80:20 berdasarkan value label yang sudah ada.

```
#Apply Mapping to Integer for Label column
df['Label'] = df['Label'].map({'Olahraga':0,'Politik':1,'Hiburan':2})

df.head()
```

	Unnamed: 0	Teks	Media	Label
0	0	san antonio spurs kalah tuan rumah phoenix sun...	Detik.com	0
1	1	hilang main tottenham hotspur nekat main tahan...	Detik.com	0
2	2	nicolas jackson cetak hat trick menang chelsea...	Detik.com	0
3	3	lionel me i menang ballon d or unggul cukup ja...	Detik.com	0
4	4	liga basket ajar indonesia dbl buat kejut part...	Detik.com	0

```
#Split the dataset into Train and Test
X_train, X_test,Y_train, Y_test = train_test_split(df['Teks'],df['Label'],random_state = 42 , test_size=0.2, stratify = df['Label'])
```

Pada kasus ini, metode yang digunakan adalah Word2Vec dan juga FastText dengan keduanya menggunakan mode skip-gram. Mode skip-gram digunakan sebab kasus soal menginginkan set vektor yang berdasarkan hasil input training kata dan target output konteks.

1. Word2Vec

Word2Vec merupakan metode Text Representation yang menggunakan hubungan antara kata (word embeddings) dalam bentuk vektor numerik, sehingga hubungan antara kata satu dengan kata lain dapat ditangkap dari sebuah teks. Mode Skip-Gram akan berusaha memprediksi context kata dari input word yang diberikan. Disini, parameter yang wajib diberikan adalah vector_size = 50 dan min_count >=3(sesuai permintaan soal)

Sebagai tambahan, disini word2Vec harus dilatih dari korpus atau kata-kata yang ada dalam data Training saja, sebab disitulah bank data yang dimiliki oleh kita.

```
#Ambil setiap kata dalam X_train, disini tidak usah tokenization, sebab dalam data cleaning sudah dipastikan bahwa setiap kata terpaut satu spasi
sentences = [sentence.split() for sentence in X_train]

#Train model Word2vec sesuai dengan keinginan soal
#Penjelasan detail ada di pdf
word2vec_model = gensim.models.Word2Vec(sentences,
                                        vector_size=50,
                                        window=10,
                                        min_count=3,
                                        sg = 1)
```

Disini,

- vector_size menjelaskan mengenai banyaknya vektor dimensi yang akan terbuat dalam proses training nantinya. Disini vector_size = 50, maka nanti setiap data akan memiliki 50 dimensi
- window mengontrol jumlah kata sebelum dan sesudah kata target yang diambil sebagai konteks untuk pelatihan model.Jika window = 10, maka di setiap kata X, akan ada 10 kata sebelum dan sesudah X yang diambil sebagai konteks representative kata tersebut
- min_count menyatakan jumlah kata minimum yang akan diperhitungkan agar kata diperhitungkan dalam model. Jika min_count = 3, maka kata-kata yang muncul kurang dari 3 kali dalam korpus training word2Vec akan diabaikan(valued 0).
- sg = 1 menyatakan mode skip-gram, jika sg = 0 akan menyatakan mode CBOW

2. FastText

   FastText adalah kemajuan dari Word2Vec yang dikembangkan oleh Facebook, dimana selain representasi kata, FastText juga merepresentasikan sub-kata dari setiap kata yang ada. Sehingga, kita dapat juga mencari kata-kata yang sulit diraih dari teks, serta lebih kompleks dalam perhitungan dan memakan waktu lebih lama.

```
sentences = [sentence.split() for sentence in X_train]

fastText_model = gensim.models.FastText(sentences,
                                        vector_size=50,
                                        window=10,
                                        min_count=3,
                                        sg=1
                                       )
```

Setelah melakukan model text representation, setiap model akan melakukan vektorisasi terhadap setiap kata yang ada di X_train maupun X_test secara terpisah.

Untuk Word2Vec:

```
#Lakukan vektorisasi untuk setiap kata
def vectorize_word2vec(sentence): #Parameter = kalimat/teks
    words = sentence.split() #Pecah teks menjadi kata
     #Lakukan word vectorizing untuk setiap kata dari model word2vec yang ada
    words_vecs = [word2vec_model.wv[word] for word in words if word in word2vec_model.wv]
    if len(words_vecs) == 0: #Kalau ternyata tidak ada / tidak memenuhi min_count, maka nilainya adalah 0
        return np.zeros(50) #return nilai 0
    words_vecs = np.array(words_vecs) #Kalau tidak 0, jadikan array lalu keluarkan rata-ratanya
    return words_vecs.mean(axis=0)
```

Apply ke X_train dan X_test , kemudian store dalam numpy array.

```
#Apply word vectorize ke setiap kata yang ada pada X_train dan X_test
X_train_word2vec= np.array([vectorize_word2vec(sentence) for sentence in X_train])
X_test_word2vec = np.array([vectorize_word2vec(sentence) for sentence in X_test])
```

Hal yang sama juga dilakukan pada model FastText

```
#Lakukan vektorisasi untuk setiap kata
def vectorize_fastText(sentence): #Parameter = kalimat/teks
    words = sentence.split() #Pecah teks menjadi kata
    #Lakukan word vectorizing untuk setiap kata dari model word2vec yang ada
    words_vecs = [fastText_model.wv[word] for word in words if word in fastText_model.wv]
    if len(words_vecs) == 0:#Kalau ternyata tidak ada / tidak memenuhi min_count, maka nilainya adalah 0
        return np.zeros(50) #return nilai 0
    words_vecs = np.array(words_vecs) #Kalau tidak 0, jadikan array lalu keluarkan rata-ratanya
    return words_vecs.mean(axis=0)
```

Apply ke X_train dan X_test , kemudian store dalam numpy array.

```
#Apply word vectorize ke setiap kata yang ada pada X_train dan X_test
X_train_fastText = np.array([vectorize_fastText(sentence) for sentence in X_train])
X_test_fastText = np.array([vectorize_fastText(sentence) for sentence in X_test])
```

Modelling

Modelling dilakukan dengan menggunakan SVM dan Random Forest tanpa adanya parameter tuning. Kedua model menggunakan default parameter agar memastikan tidak ada biased modelling dalam komparasi kedua model.

Model SVM

```
from sklearn.svm import SVC
svm = SVC()
```

Kemudian, model dilatih dengan kedua text representation method

Word2Vec embedding

```
1 model_svm_word2vec = svm.fit(X_train_word2vec,Y_train)
2
3 prediction_word2vec_svm = model_svm_word2vec.predict(X_test_word2vec)
```

```
1 print(confusion_matrix(prediction_word2vec_svm, Y_test))
```

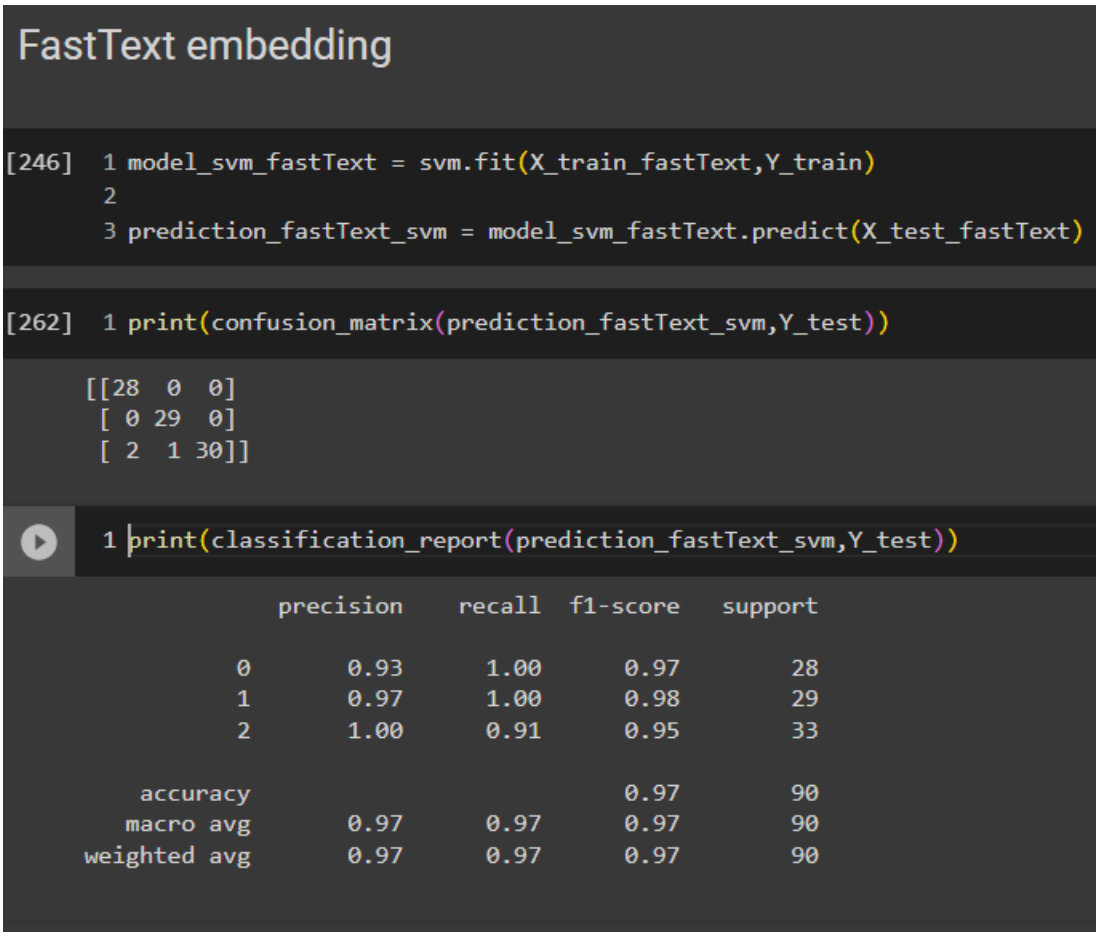
```
[[30  0  0]
 [ 0 29  1]
 [ 0  1 29]]
```

```
1 print(classification_report(prediction_word2vec_svm, Y_test))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	30
1	0.97	0.97	0.97	30
2	0.97	0.97	0.97	30
accuracy			0.98	90
macro avg	0.98	0.98	0.98	90
weighted avg	0.98	0.98	0.98	90

Disini, terlihat bahwa model SVM bekerja dengan hampir sempurna pada model word2Vec, dimana terdapat

- 1 misklasifikasi kelas 1(Politik) ke kelas 2(Hiburan) dan sebaliknya, mengakibatkan keduanya memiliki precision 97% (29/30 *100%) dan recall 97% juga, sebab hanya 29 prediksi yang benar dari total 30 prediksi label 2(29/30 * 100%).
- Model juga memberikan akurasi akhir 98%



Disini, terlihat bahwa model SVM juga bekerja dengan hampir sempurna pada model fastText, dimana terdapat

- 2 misklasifikasi kelas 0(Olahraga) ke kelas 1(Politik) dan 1 misklasifikasi dari label 1 ke label 2 , mengakibatkan keduanya memiliki precision 93% (28/30 *100%) dan 97%(29/30 * 100%)
- Label 2 memiliki nilai recall 91%, sebab ada 33 prediksi di kelas 2 dan hanya 30 yang benar-benar label 2 (30/33 * 100%).
- Model juga memberikan akurasi akhir 97%

Model Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()
```

```
1 model_rf_word2vec = rf.fit(X_train_word2vec,Y_train)
2
3 prediction_word2vec_rf = model_rf_word2vec.predict(X_test_word2vec)
```

```
1 print(confusion_matrix(prediction_word2vec_rf, Y_test))
```

```
[[30  0  2]
 [ 0 30  1]
 [ 0  0 27]]
```

```
1 print(classification_report(prediction_word2vec_rf, Y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        32
           1       1.00      0.97      0.98        31
           2       0.90      1.00      0.95        27

    accuracy                           0.97        90
   macro avg       0.97      0.97      0.97        90
weighted avg       0.97      0.97      0.97        90
```

Pada model random Forest dengan metode word2Vec representation, terlihat bahwa:

- Terdapat 3 misklasifikasi label 2 , 2 data masuk ke dalam Label 0 dan 1 data masuk ke dalam label 1, menghasilkan precision 90% saja (27 / 30 * 100%), sebab hanya 27 data dari 30 data yang terprediksi benar
- Recall pada label 0 dan 1 berakibat menurun menjadi 94%(30/32 * 100%) dan 97%(30/31 *100%) , sebagai akibat dari misklasifikasi label 2
- Akurasi keseluruhan model mencapai 97%

FastText embedding

```
1 model_rf_fastText = rf.fit(X_train_fastText,Y_train)
2
3
4 prediction_fastText_rf = model_rf_fastText.predict(X_test_fastText)
```

```
[255]  1 print(confusion_matrix(prediction_fastText_rf, Y_test))
```

```
[[30  0  1]
 [ 0 28  1]
 [ 0  2 28]]
```

```
[256]  1 print(classification_report(prediction_fastText_rf, Y_test))
```

	precision	recall	f1-score	support
0	1.00	0.97	0.98	31
1	0.93	0.97	0.95	29
2	0.93	0.93	0.93	30
accuracy			0.96	90
macro avg	0.96	0.96	0.96	90
weighted avg	0.96	0.96	0.96	90

Pada model Random Forest dengan fastText representation, terlihat bahwa

- Label 1 mengalami 2 misklasifikasi ke label 2 , sedangkan label 2 mengalami misklasifikasi ke label 0 dan label 1 sebanyak 1 data masing-masing.
- Berakibat dengan menurunnya precision dan recall masing-masing label
    1. Label 0 memiliki recall menjadi 97% (30/31 * 100%)
    2. Label 1 memiliki precision 93% (28/30*100%)
    3. Label 1 memiliki recall 97% (28/29 * 100%)
    4. Label 2 memiliki precision 93% (28/30 * 100%)
    5. Label 2 memiliki recall 93% (28/30 * 100%)
- Total akurasi keseluruhan mencapai 96%

Final Conclusion

Setelah dilakukan scrapping, labelling, kemudian cleaning, text representation dan juga modelling, terlihat bahwa model yang dihasilkan dari setiap text representation sudah sangat baik, dimana total akurasi mencapai >95%. Untuk model terbaik diberikan kepada model SVM dengan text representation word2Vec.

Masih banyak hal yang dapat dikulik dari kasus ini, seperti hyperparameter tuning pada kedua model dan juga memodifikasi text representation parameter, terutama min_count. Namun, dikarenakan kendala waktu, hal tersebut masih belum dapat dilakukan