LAPORAN TUGAS NLP

Muhammad Yasin Fajari

1301164666

1. Ubah format penyimpanan data ke CSV

```
df.to_csv("Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\News_dataset.csv", sep=';',index=False, encoding='utf-8')
```

```
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\News_dataset.csv', 'wb') as output:
    pickle.dump(df, output)
```

```
# X_train
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\X_train.csv', 'wb') as output:
    pickle.dump(X_train, output)

# X_test
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\X_test.csv', 'wb') as output:
    pickle.dump(X_test, output)

# y_train
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\y_train.csv', 'wb') as output:
    pickle.dump(y_train, output)

# y_test
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\y_test.csv', 'wb') as output:
    pickle.dump(y_test, output)

# df
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\df.csv', 'wb') as output:
    pickle.dump(df, output)

# features_train
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\features_train.csv', 'wb') as output:
    pickle.dump(features_train, output)

# labels_train
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\labels_train.csv', 'wb') as output:
    pickle.dump(labels_train, output)

# features_test
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\features_test.csv', 'wb') as output:
    pickle.dump(features_test, output)

# labels_test
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\labels_test.csv', 'wb') as output:
    pickle.dump(labels_test, output)

# TF-IDF object
with open('Y:\\Yasin Data\\Kuliah\\Natural Language Processing\\Tugas 4\\NLP-Course-TelU-master\\Text Classification\\Data\\answer\\tfidf.csv', 'wb') as output:
    pickle.dump(tfidf, output)
```

Pengubahan format simpan data dari *.pickle* menjadi *.csv* pada setiap proses simpan data.

2. Coba buatkan feature berikut (save dan upload feature), lalu laporkan pengaruhnya terhadap akurasi klasifikasi: a. Tanpa proses normalisation b. Tanpa proses lemmatisation c. Tanpa menghilangkan stopwords

a.

1. Text Cleaning and Preparation

Berikut adalah langkah-langkah text cleaning dan preprocessing:

1. Hapus karakter khusus '\r', '\n', "",
2. Ubah ke bentuk lowercase
3. Hapus sejumlah simbol
4. Hapus 's, misal student's name
5. Lematisasi, mengubah ke bentuk kata dasar dengan bantuan wordnet
6. Hapus stopwords

```
In [17]: # df['Content_Parsed_1'] = df['Content'].str.replace("\r", " ")
         # df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace("\n", " ")
         # df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace("    ", " ")

In [18]: # df['Content_Parsed_1'] = df['Content_Parsed_1'].str.replace('"', '')

In [19]: # df['Content_Parsed_2'] = df['Content_Parsed_1'].str.lower()

In [20]: # punctuation_signs = list("?:!.,;")
         # df['Content_Parsed_3'] = df['Content_Parsed_2']

         # for punct_sign in punctuation_signs:
         #     df['Content_Parsed_3'] = df['Content_Parsed_3'].str.replace(punct_sign, '')

In [21]: # # Remove possessive pronouns
         # df['Content_Parsed_4'] = df['Content_Parsed_3'].str.replace("'s", "")
```

```
In [18]: base_model = LogisticRegression(random_state = 8)
         base_model.fit(features_train, labels_train)
         accuracy_score(labels_test, base_model.predict(features_test))

Out[18]: 0.9281437125748503

In [19]: best_classifier.fit(features_train, labels_train)
         accuracy_score(labels_test, best_classifier.predict(features_test))

Out[19]: 0.9341317365269461
```

Pada proses tanpa normalisasi didapatkan akurasi seperti yang tertera pada gambar, hal tersebut sedikit mempengaruhi nilai akurasi. Pada proses normal menghasilkan akurasi sebesar 95% pada *base model* dan 94% pada *best classifier*, dimana dihasilkan nilai akurasi yang lebih rendah.

b.

```
In [10]: # # Downloading punkt and wordnet from NLTK
         # nltk.download('punkt')
         # print("------------------------------------------------------------")
         # nltk.download('wordnet')

In [11]: # # Saving the lemmatizer into an object
         # wordnet_lemmatizer = WordNetLemmatizer()

In [12]: # nrows = len(df)
         # lemmatized_text_list = []

         # for row in range(0, nrows):

         #     # Create an empty list containing lemmatized words
         #     lemmatized_list = []

         #     # Save the text and its words into an object
         #     text = df.loc[row]['Content_Parsed_4']
         #     text_words = text.split(" ")

         #     # Iterate through every word to lemmatize
         #     for word in text_words:
         #         lemmatized_list.append(wordnet_lemmatizer.lemmatize(word, pos="v"))

         #     # Join the list
         #     lemmatized_text = " ".join(lemmatized_list)

         #     # Append to the list containing the texts
         #     lemmatized_text_list.append(lemmatized_text)

In [13]: # df['Content_Parsed_5'] = lemmatized_text_list
```

```
In [18]: base_model = LogisticRegression(random_state = 8)
         base_model.fit(features_train, labels_train)
         accuracy_score(labels_test, base_model.predict(features_test))
Out[18]: 0.9431137724550899

In [19]: best_classifier.fit(features_train, labels_train)
         accuracy_score(labels_test, best_classifier.predict(features_test))
Out[19]: 0.9461077844311377
```

Pada proses tanpa lemmatisasi didapatkan akurasi seperti yang tertera pada gambar, hal tersebut tidak berpengaruh pada nilai akurasi. Pada proses normal menghasilkan akurasi sebesar 95% pada *base model* dan 94% pada *best classifier*. Pada *best classifier* didapatkan nilai akurasi lebih tinggi dari pada proses normal.

c.

```
In [14]: # # Downloading the stop words list
         # nltk.download('stopwords')

In [15]: # # Loading the stop words in english
         # stop_words = list(stopwords.words('english'))

In [16]: # stop_words[0:10]

In [17]: # df['Content_Parsed_6'] = df['Content_Parsed_5']

         # for stop_word in stop_words:

         #     regex_stopword = r"\b" + stop_word + r"\b"
         #     df['Content_Parsed_6'] = df['Content_Parsed_6'].str.replace(regex_stopword, '')

In [18]: base_model = LogisticRegression(random_state = 8)
         base_model.fit(features_train, labels_train)
         accuracy_score(labels_test, base_model.predict(features_test))
Out[18]: 0.9221556886227545

In [19]: best_classifier.fit(features_train, labels_train)
         accuracy_score(labels_test, best_classifier.predict(features_test))
Out[19]: 0.9311377245508982
```

Pada proses tanpa *stopword* didapatkan akurasi seperti yang tertera pada gambar, hal tersebut sedikit berpengaruh pada nilai akurasi. Pada proses normal menghasilkan akurasi sebesar 95% pada *base model* dan 94% pada *best classifier*, dimana didapatkan hasil akurasi yang lebih rendah.

3. Coba buat tfidf dengan nilai "max_features" yang berbeda-beda (lebih besar dan lebih kecil dari 300), lalu laporkan pengaruhnya terhadap akurasi klasifikasi.
   a. Lebih tinggi dari 300 (max_features = 550)

```
In [33]: # Parameter election
         ngram_range = (1,2)
         min_df = 10
         max_df = 1.
         max_features = 550

In [18]: base_model = LogisticRegression(random_state = 8)
         base_model.fit(features_train, labels_train)
         accuracy_score(labels_test, base_model.predict(features_test))
Out[18]: 0.9640718562874252

In [19]: best_classifier.fit(features_train, labels_train)
         accuracy_score(labels_test, best_classifier.predict(features_test))
Out[19]: 0.9700598802395209
```

Semakin kecil *max_features*, maka semakin kecil akurasi dikarenakan pembobotan yang dilakukan kecil sehingga akurasi tidak terlalu baik.

b. Lebih rendah dari 300 (max_features = 100)

```
In [71]: # Parameter election
         ngram_range = (1,2)
         min_df = 10
         max_df = 1.
         max_features = 100
```

```
In [37]: base_model = LogisticRegression(random_state = 8)
         base_model.fit(features_train, labels_train)
         accuracy_score(labels_test, base_model.predict(features_test))
Out[37]: 0.8802395209580839
```

```
In [38]: best_classifier.fit(features_train, labels_train)
         accuracy_score(labels_test, best_classifier.predict(features_test))
Out[38]: 0.8922155688622755
```

Semakin besar *max_features*, maka semakin besar akurasi dikarenakan pembobotan yang dilakukan besar sehingga akurasi bisa sangat baik.

4. Coba dengan beberapa algoritma klasifikasi yang berbeda (minimal 2 algoritma), carilah parameter terbaik (jelaskan nilai2 parameter yang telah dicoba untuk tiap jenis algoritma).

   a. Decision Tree

```
In [9]: # Create the parameter grid based on the results of random search
        param_grid = {'criterion': ['gini', 'entropy'],
                      'max_depth': [1, 2, 3, 4, 5, 6, 7, 8],
                      'min_samples_split': [2, 3]}

        # Create a base model
        classifier = DecisionTreeClassifier()
```

```
In [99]: base_model = svm.SVC(random_state = 8)
         base_model.fit(features_train, labels_train)
         accuracy_score(labels_test, base_model.predict(features_test))
Out[99]: 0.9550898203592815
```

```
In [100]: best_classifier.fit(features_train, labels_train)
          accuracy_score(labels_test, best_classifier.predict(features_test))
Out[100]: 0.7275449101796407
```

Pada decTree parameter yang digunakan ada 3 yaitu kriteria yang digunakan gini index atau entropy, lalu max_depth untuk iterasi sebanyak 8 dan sample_split sebanyak 2 – 3.

   b. Naïve Bayes

```
In [9]: # Create the parameter grid based on the results of random search
        param_grid = {'var_smoothing': np.logspace(0,-9, num=100)}

        # Create a base model
        classifier = GaussianNB()
```

```
In [18]: base_model = DecisionTreeClassifier()
         base_model.fit(features_train, labels_train)
         accuracy_score(labels_test, base_model.predict(features_test))
Out[18]: 0.8053892215568862
```

```
In [19]: best_classifier.fit(features_train, labels_train)
         accuracy_score(labels_test, best_classifier.predict(features_test))
Out[19]: 0.7574850299401198
```

Pada NB parameter yang digunakan hanya var_smoothing yang digunakan untuk memilih varian nilai terbesar dari setiap fitur.

5. Jika anda ingin menggunakan teks bahasa Indonesia, bagian mana saja yang perlu dilakukan penyesuaian?

   Yang perlu dilakukan penyesuain yaitu pada proses *preprocessing* data, sebagai berikut:

   a. Case Folding

      Dalam proses ini yang perlu dilakukan yaitu:

      a. Mengubah menjadi *lowercase*
      b. Menghapus simbol-simbol, karakter *whitespace*, tanda seperti "\r, \n, \t, dsb."
      c. Tidak perlu melakukan penghapusan pada tanda baca seperti "'**s**" karena itu hanya berlaku pada teks Bahasa Inggris

   b. Tokenizing

      Dapat dilakukan untuk menghitung banyak jumlah perkata yang muncul, dapat dilakukan juga pada 1 kalimat langsung. Jadi misal ada 1 paragraf yang memiliki 2 atau lebih kalimat, digunakan *tokenizing.sent_tokenize()* untuk memecah perkalimat pada paragraf tersebut.

   c. Filtering

      Pada proses ini dilakukan *filter stopword* untuk Bahasa Indonesia, bisa menggunakan librari Sastrawi untuk prosesnya.

   d. Stemming

      Proses ini dilakukan untuk mengubah setiap kata kedalam kata dasar. Pada prosesnya bisa digunakan juga librari Sastrawi.

6. Opsional: Gunakan word embedding (e.g word2vec, GloVe).