# Praktikum Kecerdasan Buatan

## Analisis Sentimen menggunakan RNN

1. Buat project baru pada google colaboratory. Beri nama `analisis_sentimen_rnn.ipynb`.
2. Unduh dataset yang akan digunakan dari halaman berikut: https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment
3. Lalu baca data menggunakan `pandas`, kemudian lakukan preprocessing teks terlebih dahulu. Gunakan library tweet-preprocessor (https://pypi.org/project/tweet-preprocessor/) untuk membersihkan tweet. Kemudian hilangkan tanda baca dan ubah menjadi huruf kecil. Anda dapat menambahkan preprocessing lain yang sekiranya dibutuhkan.

```
df['text_clean'] = ''


import preprocessor as p
#forming a separate feature for cleaned tweets
for i,v in enumerate(df['text']):
    df.loc[i,'text_clean'] = p.clean(v)


# converting all text to lower case
df['text_clean'] = df['text_clean'].apply(str.lower)

# using regex to remove punctuation
df['text_clean'] = df['text_clean'].apply(lambda x: re.sub(r'[^\w\s]', '',
x))
```

Setelah kode di atas dijalankan, output yang diharapkan yaitu:

```
df[['text', 'text_clean']]
```

| | text | text_clean |
|---|---|---|
| 0 | @VirginAmerica What @dhepburn said. | what said |
| 1 | @VirginAmerica plus you've added commercials t... | plus youve added commercials to the experience... |
| 2 | @VirginAmerica I didn't today... Must mean I n... | i didnt today must mean i need to take another... |
| 3 | @VirginAmerica it's really aggressive to blast... | its really aggressive to blast obnoxious enter... |
| 4 | @VirginAmerica and it's a really big bad thing... | and its a really big bad thing about it |
| ... | ... | ... |

Selanjutnya, gunakan hanya kolom `text_clean` dan `airline_sentiment` sebagai input dan target output pada model.

```
tweets = df[['text_clean','airline_sentiment']]
```

4. Kemudian split dataset menjadi 80% training set, 10% validation set, dan 10% test set. Jangan lupa mengatur seed = 43 seperti pada project sebelumnya.

```
import numpy as np
df_train, df_val, df_test = np.split(tweets.sample(frac=1,
random_state=seed), [int(.8*len(tweets)), int(.9*len(tweets))])
```

Anda dapat melihat hasil split di atas dengan kode berikut.

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11712 entries, 5747 to 10724
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   text_clean         11712 non-null  object
 1   airline_sentiment  11712 non-null  object
dtypes: object(2)
memory usage: 274.5+ KB
```

```
df_val.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1464 entries, 2676 to 9289
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   text_clean         1464 non-null   object
 1   airline_sentiment  1464 non-null   object
dtypes: object(2)
memory usage: 34.3+ KB
```

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1464 entries, 14343 to 14148
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   text_clean         1464 non-null   object
 1   airline_sentiment  1464 non-null   object
dtypes: object(2)
memory usage: 34.3+ KB
```

5. Kemudian buat class Vocabulary dengan kode berikut.

```
class Vocabulary:

    '''
    __init__ method is called by default as soon as an object of this
class is initiated
    we use this method to initiate our vocab dictionaries
    '''
    def __init__(self, freq_threshold, max_size):
        '''
        freq_threshold : the minimum times a word must occur in corpus to
be treated in vocab
        max_size : max source vocab size. Eg. if set to 10,000, we pick
the top 10,000 most frequent words and discard others
        '''
        #initiate the index to token dict
        ## <PAD> -> padding, used for padding the shorter sentences in a
batch to match the length of longest sentence in the batch
        ## <UNK> -> words which are not found in the vocab are replace by
this token
        self.itos = {0: '<PAD>', 1: '<UNK>'}
        #initiate the token to index dict
        self.stoi = {k:j for j,k in self.itos.items()}

        self.freq_threshold = freq_threshold
        self.max_size = max_size

    '''
    __len__ is used by dataloader later to create batches
    '''
    def __len__(self):
        return len(self.itos)

    '''
    a simple tokenizer to split on space and converts the sentence to list
of words
    '''
    @staticmethod
    def tokenizer(text):
        return [tok.lower().strip() for tok in text.split(' ')]

    '''
    build the vocab: create a dictionary mapping of index to string (itos)
and string to index (stoi)
    output ex. for stoi -> {'the':5, 'a':6, 'an':7}
    '''
```

```
    def build_vocabulary(self, sentence_list):
        #calculate the frequencies of each word first to remove the words
with freq < freq_threshold
        frequencies = {}  #init the freq dict
        idx = 4 #index from which we want our dict to start. We already
used 4 indexes for pad, start, end, unk

        #calculate freq of words
        for sentence in sentence_list:
            for word in self.tokenizer(sentence):
                if word not in frequencies.keys():
                    frequencies[word]=1
                else:
                    frequencies[word]+=1


        #limit vocab by removing low freq words
        frequencies = {k:v for k,v in frequencies.items() if
v>self.freq_threshold}

        #limit vocab to the max_size specified
        frequencies = dict(sorted(frequencies.items(), key = lambda x: -
x[1])[:self.max_size-idx]) # idx =4 for pad, start, end , unk

        #create vocab
        for word in frequencies.keys():
            self.stoi[word] = idx
            self.itos[idx] = word
            idx+=1

    '''
    convert the list of words to a list of corresponding indexes
    '''
    def numericalize(self, text):
        #tokenize text
        tokenized_text = self.tokenizer(text)
        numericalized_text = []
        for token in tokenized_text:
            if token in self.stoi.keys():
                numericalized_text.append(self.stoi[token])
            else: #out-of-vocab words are represented by UNK token index
                numericalized_text.append(self.stoi['<UNK>'])

        return numericalized_text
```

Pelajari kode di atas.

6. Selanjutnya buat class Dataset dan DataLoader berikut.

```
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
```

```
class TweetSentimentDataset(Dataset):
    # Static constant variable
    LABEL2INDEX = {'positive': 0, 'neutral': 1, 'negative': 2}
    INDEX2LABEL = {0: 'positive', 1: 'neutral', 2: 'negative'}
    NUM_LABELS = 3

    def load_dataset(self, df):
        df.columns = ['text_clean','airline_sentiment']
        #print(df)
        df['airline_sentiment'] = df['airline_sentiment'].apply(lambda
lab: self.LABEL2INDEX[lab])
        return df

    def __init__(self, df, freq_threshold = 3, vocab_max_size = 10000,
*args, **kwargs):
        self.data = self.load_dataset(df)
        self.source_texts = self.data['text_clean'].tolist()
        self.freq_threshold = freq_threshold
        self.vocab_max_size = vocab_max_size
        self.vocab = Vocabulary(freq_threshold, vocab_max_size)
        self.vocab.build_vocabulary(self.source_texts)

    def __getitem__(self, index):
        data = self.data.iloc[index,:]
        text, sentiment = data['text_clean'], data['airline_sentiment']
        token_ids = self.vocab.numericalize(text)
        return torch.tensor(token_ids), torch.tensor(sentiment)

    def __len__(self):
        return len(self.data)
```

```
class MyCollate:
    def __init__(self, pad_idx):
        self.pad_idx = pad_idx

    def __call__(self, batch):
        input_tensors = []
        labels = []
        lengths = []
```

```
        for x, y in batch:
            input_tensors.append(x)
            labels.append(y)
            lengths.append(x.shape[0]) #Assume shape is (T, *)
        longest = max(lengths)
        if len(input_tensors[0].shape) == 1:
            x_padded = torch.nn.utils.rnn.pad_sequence(input_tensors,
batch_first=True, padding_value = self.pad_idx)
        else:
            raise Exception('Current implementation only supports (T)
shaped data')
        y_batched = torch.as_tensor(labels, dtype=torch.long)
        return x_padded, y_batched
```

```
# create Tensor datasets
train_data = TweetSentimentDataset(df_train)
valid_data = TweetSentimentDataset(df_val)
test_data = TweetSentimentDataset(df_test)
```

```
# dataloaders
batch_size = 5

# make sure to SHUFFLE your data
pad_idx = 0
train_loader = DataLoader(train_data, shuffle=True, batch_size=batch_size,
collate_fn = MyCollate(pad_idx=pad_idx), worker_init_fn=seed_worker,
generator=g)
valid_loader = DataLoader(valid_data, shuffle=True, batch_size=batch_size,
collate_fn = MyCollate(pad_idx=pad_idx), worker_init_fn=seed_worker,
generator=g)
test_loader = DataLoader(test_data, shuffle=True, batch_size=batch_size,
collate_fn = MyCollate(pad_idx=pad_idx), worker_init_fn=seed_worker,
generator=g)
```

7. Tambahkan kode berikut untuk mengubah device ke GPU jika tersedia.

```
import torch
is_cuda = torch.cuda.is_available()

# If we have a GPU available, we'll set our device to GPU. We'll use this
device variable later in our code.
if is_cuda:
    device = torch.device("cuda")
    print("GPU is available")
else:
    device = torch.device("cpu")
    print("GPU not available, CPU used")
```

8. Buat class untuk model RNN dengan kode berikut.

```
import torch.nn as nn
import torch.nn.functional as F
class SentimentRNN(nn.Module):
    def
__init__(self,no_layers,vocab_size,hidden_dim,embedding_dim,output_dim,dro
p_prob=0.5):
        super(SentimentRNN,self).__init__()

        self.output_dim = output_dim
        self.hidden_dim = hidden_dim

        self.no_layers = no_layers
        self.vocab_size = vocab_size
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn =
nn.RNN(input_size=embedding_dim,hidden_size=self.hidden_dim,
                            num_layers=no_layers, batch_first=True)
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(self.hidden_dim, output_dim)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self,x):
        batch_size = x.size(0)
        embeds = self.embedding(x)
        rnn_out, hidden = self.rnn(embeds)
        out = self.dropout(hidden.squeeze(0))
        out = self.fc(out)
        output = self.softmax(out)
        return output
```

Kemudian buat model dengan kode berikut.

```
no_layers = 1
vocab_size = len(train_data.vocab) + 2 #extra 2 for padding and unknown
embedding_dim = 64
output_dim = 3
hidden_dim = 256


model =
SentimentRNN(no_layers,vocab_size,hidden_dim,embedding_dim,output_dim,drop
_prob=0.5)

#moving to gpu
model.to(device)
```

Ilustrasikan arsitektur RNN yang digunakan dengan kode di atas.

9. Lakukan training dengan kode berikut.

```
import time
start_time = time.time()
epochs = 10
val_accuracies = []

for epoch in range(epochs):
    running_loss = 0.0
    model.train()
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data[0].to(device), data[1].to(device)
        model.zero_grad()
        logits = model(inputs)
        loss = criterion(logits, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 500 == 499:    # print every 500 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss /
500:.3f}')
            running_loss = 0.0
    with torch.set_grad_enabled(False):
        val_accuracies.append(compute_accuracy(model, valid_loader,
device))
        print(f'val accuracy:'
        f'{val_accuracies[-1]:.2f}%'
        )

    print(f'time elapsed: {(time.time() - start_time)/60:.2f} min')
```

10. Kemudian tambahkan kode berikut untuk menghitung akurasi validasi pada setiap epoch.

```
def compute_accuracy(model, data_loader, device):
  with torch.no_grad():
    correct_pred, num_examples = 0, 0
    for i, (features, targets) in enumerate(data_loader):
      features = features.to(device)
      targets = targets.to(device)

      logits = model(features)
      _, predicted_labels = torch.max(logits, 1)
      num_examples += targets.size(0)
      correct_pred += (predicted_labels == targets).sum()
    return correct_pred.float()/num_examples * 100
```

```
with torch.set_grad_enabled(False):
        val_accuracies.append(compute_accuracy(model, valid_loader,
device))
        print(f'val accuracy:'
        f'{val_accuracies[-1]:.2f}%'
        )
```

11. Buat line chart untuk menggambarkan akurasi pada validation set untuk setiap epoch menggunakan `matplot` library.
12. Perhatikan hasil akurasi pada validation set untuk setiap epoch. Kemudian gunakan model dengan epoch yang memberikan akurasi terbaik pada validation set, untuk kemudian digunakan untuk mengukur akurasi pada testing set. Sesuaikan kode pada nomor 10 untuk mengukur akurasi pada testing set.

    Jangan lupa menyimpan output dari proses training, validation, dan testing sebagai laporan praktikum.
13. Modifikasi dataset menjadi hanya menggunakan sentiment positif dan negatif saja. Lalu buat kembali model RNN yang sesuai.

## Responsi

Buat laporan praktikum yang terdiri dari arsitektur RNN, output proses training pada setiap epoch (loss training, akurasi validasi, line chart akurasi validasi) serta akurasi pada testing set untuk model RNN untuk analisis sentiment dengan 3 label (positif, negatif, netral) dan 2 label (positif, negatif). Jangan lupa tuliskan model pada epoch berapa yang digunakan sebagai model terbaik.