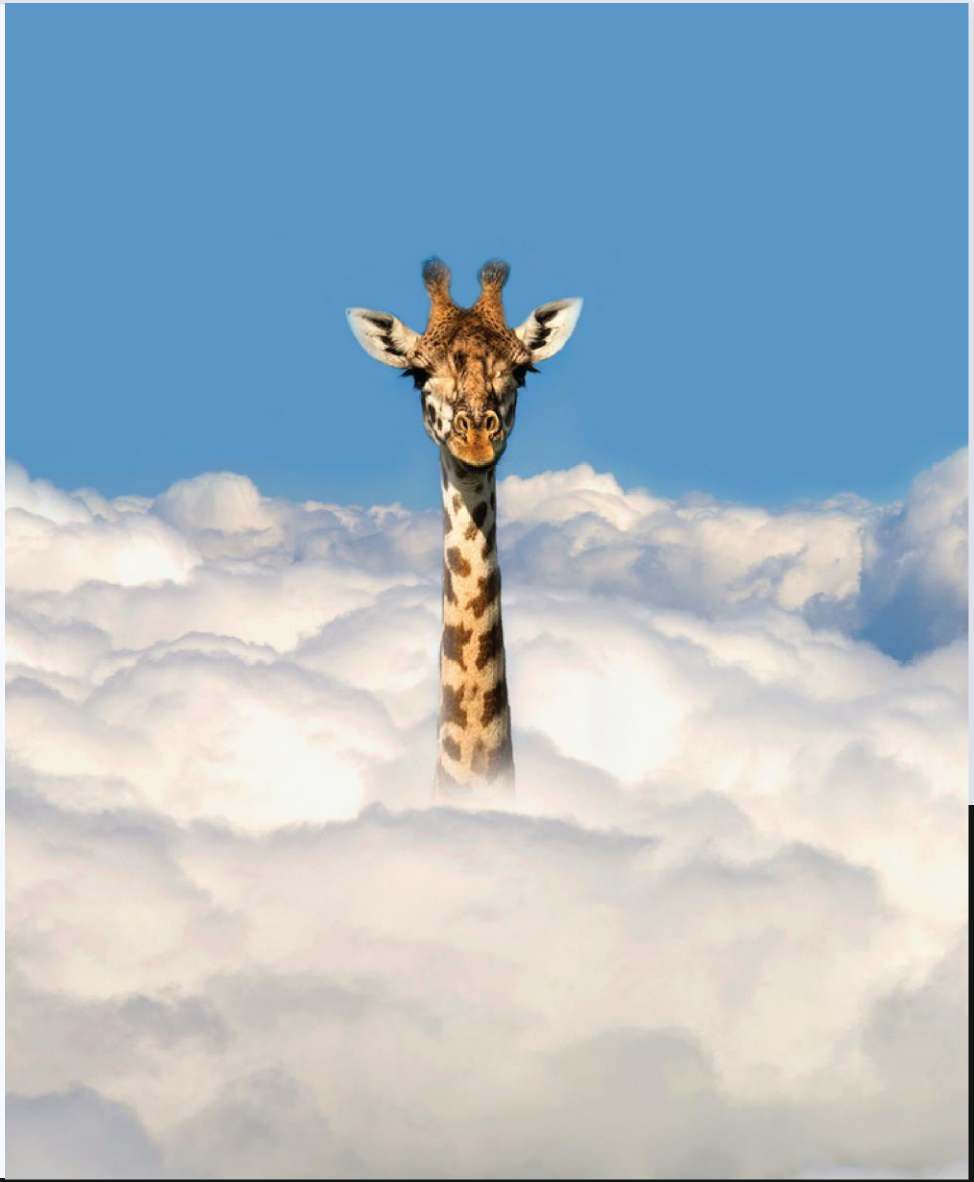
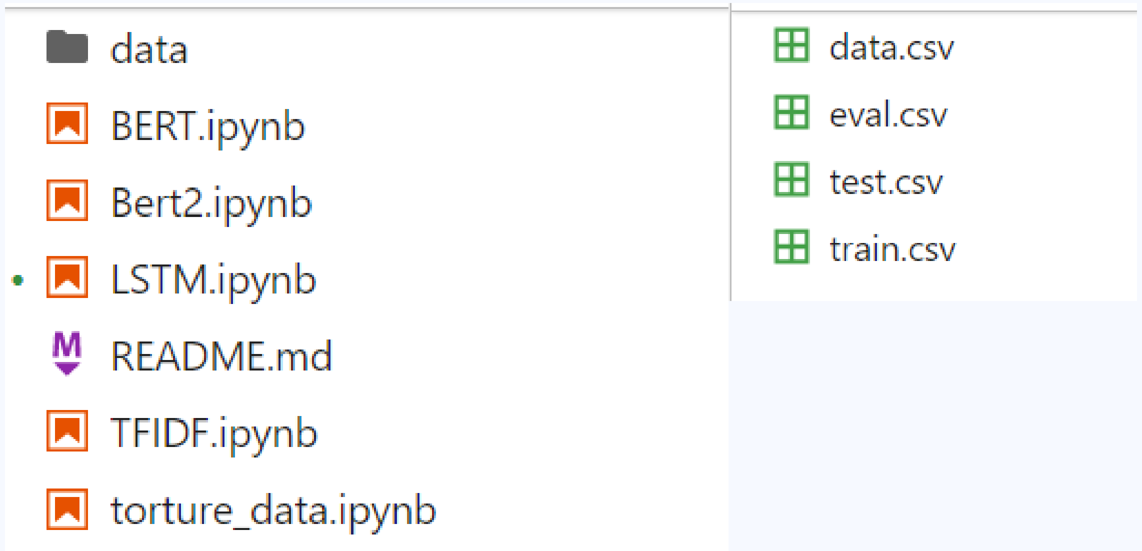
SENTIMENT ANALYSIS OVER DIGIKALA DATASET

ASH GROUP



PROJECT STRUCTURE



TF-IDF

“In This notebook used methods based on TFIDF and Logistic regression.

TD-IDF

- In This notebook used methods based on TFIDF and Logistic regression.

- **RandomizedSearchCV** from the **sklearn** library is used to obtain the most appropriate parameters for the model.

- The mentioned pipeline was used to obtain the appropriate parameters.

- In the following, we will deal with the structure and more detailed explanation of this notebook.

Notebook structure

Table of contents

- Before You Run
  - Import Libraries
- Data
  - Load Data
  - Preprocess
- TFIDF and Logistic Regression
  - Base Line Accuracy
  - Pipeline
  - TFIDF
  - Logistic Regression
  - Evaluation
- Key Features
  - Most Important Features (Top 30)

TD-IDF

- In the Before You Start section, libraries that are not installed by default on Google Club are installed with the! Pip command, and all libraries used in the project are loaded.

- A data folder is also created in the path, which contains training, evaluation and test data.

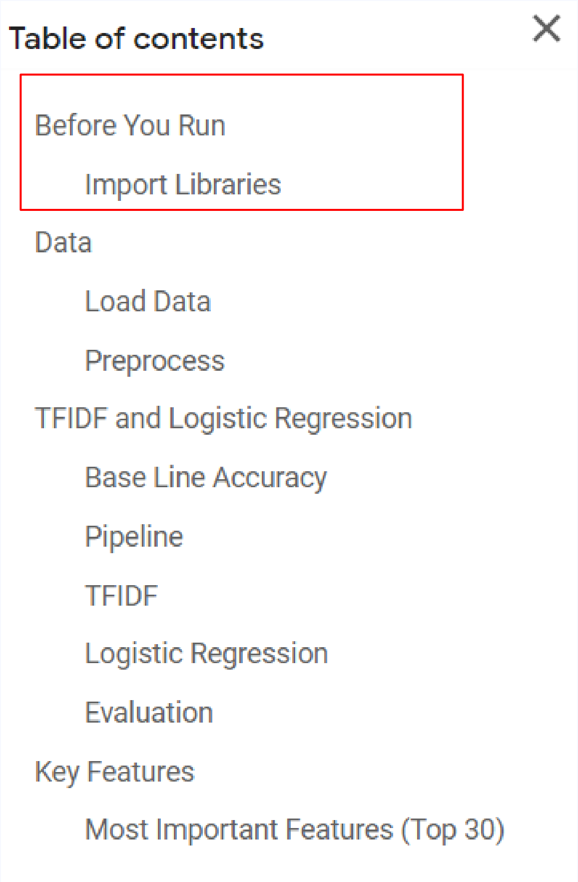


Table of contents

- Before You Run
  - Import Libraries
- Data
  - Load Data
  - Preprocess
- TFIDF and Logistic Regression
  - Base Line Accuracy
  - Pipeline
  - TFIDF
  - Logistic Regression
  - Evaluation
- Key Features
  - Most Important Features (Top 30)

- In this section, the data is read from the given path.
- The data are placed in three data frames related to training, evaluation and testing data.
- In this section, the labeled data is divided into two categories and converted to an integer so that the model can take it as input.

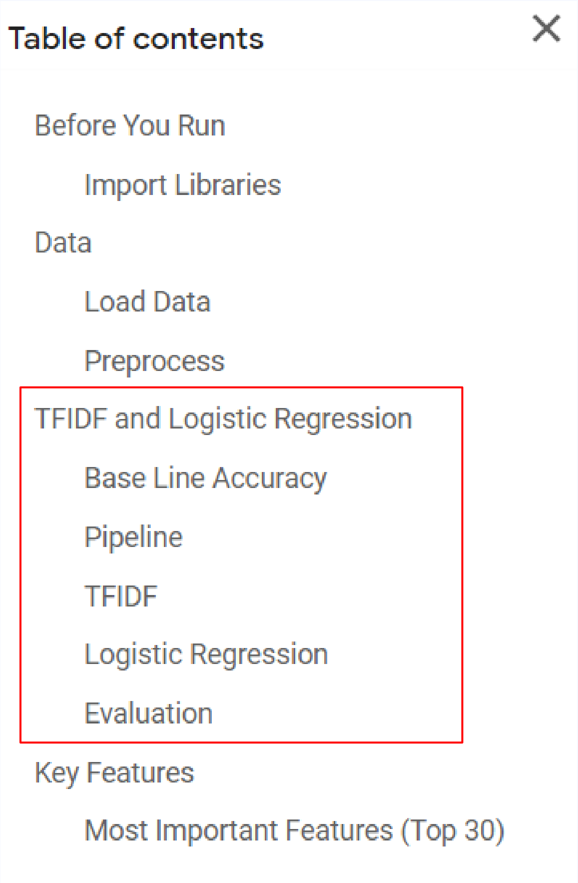
Load Data

```
PATH = 'data/'
PATH = PATH.rstrip('/')

# Train
df_train = pd.read_csv(PATH + '/train.csv')
df_train.columns = ['index', 'comment', 'rate']

# Evaluation
df_eval = pd.read_csv(PATH + '/eval.csv')
df_eval.columns = ['index', 'comment', 'rate']

# Test
df_test = pd.read_csv(PATH + '/test.csv')
df_test.columns = ['index', 'comment', 'rate']

# Create Lables
label_encoder = LabelEncoder()
# Y
train_y = label_encoder.fit_transform((df_train['rate'] >= 0).astype(int))
eval_y = label_encoder.fit_transform((df_eval['rate'] >= 0).astype(int))
test_y = label_encoder.fit_transform((df_test['rate'] >= 0).astype(int))
```

- In this section, the data is read from the given path.
- The data are placed in three data frames related to training, evaluation and testing data.
- In this section, the labeled data is divided into two categories and converted to an integer so that the model can take it as input.

```python
def clean_comment(text, allspace=True, punc=True, sentence=True, only_persian=True):
    #remove halph space, new line ('\n') and '\r'
    text = text.replace('\u200c', ' ').replace('\n', '').replace('\r', '')
    # remove punctuations
    text = re.sub(symbols_complete_reg, "", text)
    # remove arabic letters
    text = remeove_arabic(text)
    # convert spaces to a one space and delete leading and trailing spaces
    text = re.sub("(\s)+", " ", text)
    text = text.strip()
    #lemmatize
    " ".join(clean_lemmatize(text.split(" ")))
    #stemming
    " ".join(clean_stem(text.split(" ")))
    # convert spaces to a one space and delete leading and trailing spaces
    text = re.sub("(\s)+", " ", text)
    text = text.strip()
    return text
```

- Pre-processing data has also been used on the **HAZM** library.
- Part of the code is in the picture opposite.



- In this section, a value is considered as the baseline.

- The pipeline was used to obtain the appropriate parameters in two models,
- **TF-IDF** and **logistic regression**.

- Finally, the model is evaluated.

TF-IDF

```python
# solvers= ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
# multi_classes = ['multinomial', 'ovr']

pipeline = Pipeline([
        ('tfidf', TfidfVectorizer(analyzer='word')),
        ('lr', LogisticRegression(random_state=0, solver='liblinear', max_iter=1000, multi_class='ovr'))
    ])
```

```python
parameters = {
    'lr__C': (0.01, 0.1, 2, 5, 10, 15, 20),
    'lr__penalty': ('l1', 'l2'),
    'tfidf__min_df': (0, 1, 3, 5),
    'tfidf__ngram_range': ((1, 1), (1, 2), (1, 3)),
    'tfidf__max_features': (None, 2000, 8000, 12000, 15000)
}
```

This pipeline shows the parameters from which the most suitable parameter should be found.

TF-IDF

```
Best accuracy in the pipline: 0.73
```

```
Best parameters in pipeline (accuracy):
{'lr__C': 15,
 'lr__penalty': 'l2',
 'tfidf__max_features': 12000,
 'tfidf__min_df': 1,
 'tfidf__ngram_range': (1, 1)}
```

- The most appropriate parameters and accuracy based on the obtained parameters.

TF-IDF

Related to TFIDF code

Show most important features

TFIDF

```
[ ]
vectorizer = TfidfVectorizer(min_df=1, ngram_range = (1,1), max_features=12000)
train_data_features = vectorizer.fit_transform(df_train['clean_comment'])
print(train_data_features.shape)
```

```
(800, 4273)
```

```
[ ]
## data snooping ALERT: we should transforom not fit again
eval_data_features = vectorizer.transform(df_eval['clean_comment'])
test_data_features = vectorizer.transform(df_test['clean_comment'])
```

```
[ ]
# show
vectorizer.get_feature_names()[200:210]
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning
  warnings.warn(msg, category=FutureWarning)
['آدمو',
 'آذرماه',
 'آذرگانم',
 'آرام',
 'آردن',
 'آرزو',
 'آروستو',
 'آری',
 'آز',
 'آزاد']
```

Logistic Regression

```
[ ]
# Load model

model = LogisticRegression(C=15, penalty='l2', random_state=0, solver='liblinear', max_iter=1000, multi_class='ovr')
# Train model
model.fit(train_data_features, train_y)

```

```
LogisticRegression(C=15, max_iter=1000, multi_class='ovr', random_state=0,
                   solver='liblinear')
```

TF-IDF

Related to logistic regression code

Model training

Evaluation

```
## evaluation on test data
y_test_pred = model.predict(test_data_features)
```

```
# On test data
print('----- Accuracy Score ----- ')
print(accuracy_score(test_y, y_test_pred))
print('----- Confusion Matrix ----- ')
print(confusion_matrix(test_y, y_test_pred))
print('----- Classification Report ----- ')
print(classification_report(test_y, y_test_pred))

```

```
----- Accuracy Score -----
0.7352941176470589
----- Confusion Matrix -----
[[ 18  34]
 [ 11 107]]
----- Classification Report -----
              precision    recall  f1-score   support

           0       0.62      0.35      0.44        52
           1       0.76      0.91      0.83       118

    accuracy                           0.74       170
   macro avg       0.69      0.63      0.64       170
weighted avg       0.72      0.74      0.71       170
```

Evaluation

- evaluation related to test data
- The accuracy obtained is relatively better than baseline.
- confusion matrix
- classification report

KEY FEATURES

Code for displaying markers, the two classes being more or less important.

Displayed in the next slide.

Key Features

```
def top_key_features(vectorizer, model, n_top=30):
    weights = model.coef_
    feature_names = vectorizer.get_feature_names()
    sorted_features = weights[0].argsort()[::-1]
    most_important = sorted_features[:n_top]
    least_important = sorted_features[-n_top:]

    print('Most important words in the class 1: \n')
    for i in most_important:
        print(f"{feature_names[i]}: {weights[0, i]}")

    print('Most important words in the class 2: \n')
    for i in least_important:
        print(f"{feature_names[i]}: {weights[0, i]}")

```

MOST IMPORTANT WORDS

Marker with the most and the least importance

```
Most important words in the class 1:

دوربین: 3.23638261564184
نیز: 3.2268381428187576
digikala: 2.5309529271587725
تو: 2.479308604417539
اینکه: 2.4694945869725706
واقعا: 2.4462823368506523
وجود: 2.3987426101027483
همین: 2.3792914670190344
می: 2.326414357430155
هست: 2.282688498825936
نسبت: 2.259165864957199
قشنگ: 2.233477102291063
دستگاه: 2.218611980912459
کفش: 2.1748800026603132
رینگ: 2.0940869189893143
نصب: 2.092673687316176
نبود: 2.086794772131762
شده: 2.0393511467273053
پیش: 2.0336723271515424
نظر: 2.013301657967153
تقریبا: 2.0045345996962887
طعمش: 1.976633655219301
گوشفیل: 1.9611449450543894
کاملا: 1.9409312724655396
شارژ: 1.9182286786984186
موجود: 1.8853972547772857
کنه: 1.8686890072866533
جعبه: 1.8662110840563089
مونده: 1.862958366737985
کار: 1.8509578934361548
```

```
Most important words in the class 2:

ممنون: -2.585981917605968
پیشنهاد: -2.601150890635267
اصن: -2.6590737895816665
ارزه: -2.6865214236215222
میتونست: -2.700873505236188
زود: -2.7349909580529834
قیمتی: -2.736748778734155
داشتم: -2.742322547359291
ایزو: -2.74815022072744
گران: -2.783841857805191
نبودم: -2.807553418767238
خارجی: -2.861727566263604
میدان: -2.8896879792089
جنساش: -3.0736316644023147
هزینه: -3.12127795430147
چرت: -3.1470532380685987
نخرید: -3.1710191621410657
خوشمزه: -3.1895338170461547
خوش: -3.240596869286545
موقع: -3.256814634687412
اصلا: -3.3689189682318483
سریع: -3.371894314393841
همیشه: -3.3760117987597438
لک: -3.3776581342706042
معمولی: -3.4630835296163816
چندان: -3.5087714683679874
کیفیت: -3.594619493143974
ارسال: -3.6211621559710023
دقیقه: -3.9098751181828355
سنگینه: -4.043723145905925
```

LSTM + CNN

“Two different LSTM and CNN architecture have been implemented in LSTM.ipynb

NOTEBOOK STRUCTURE

LSTM + CNN

- Before You Run
  - Import Libraries
  - Load Data
  - Preprocess
- FastText Embedding
  - Download Skipgram Model
  - Load FastText Model
- LSTM Model Architecture
  - Fit LSTM Model
- CNN Model Architecture
  - Fit CNN Model

Before You Run

Import Libraries

Load Data

Preprocess

```
[ ] import matplotlib.pyplot as plt
    import seaborn as sns
    import tensorflow as tf

    from tensorflow.keras.preprocessing.sequence import pad_sequences

    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Embedding, LSTM, Bidirectional, Dense, Dropout, SpatialDropout1D
    from tensorflow.keras.layers import GlobalMaxPool1D, MaxPooling1D, GlobalMaxPooling1D, Conv1D
    from sklearn.metrics import classification_report, confusion_matrix


    from tensorflow.keras.callbacks import EarlyStopping
    import fasttext

    from hazm import word_tokenize, Normalizer
    from sklearn.preprocessing import LabelEncoder
    import pandas as pd
    import re
    import numpy as np
```

Before You Run

Import Libraries

Load Data

Preprocess

```python
PATH = 'data/'
PATH = PATH.rstrip('/')

# Train
df_train = pd.read_csv(PATH + '/train.csv')
df_train.columns = ['index', 'comment', 'rate']

# Evaluation
df_eval = pd.read_csv(PATH + '/eval.csv')
df_eval.columns = ['index', 'comment', 'rate']

# Test
df_test = pd.read_csv(PATH + '/test.csv')
df_test.columns = ['index', 'comment', 'rate']

# Create Lables
label_encoder = LabelEncoder()

train_y = label_encoder.fit_transform((df_train['rate'] >= 0).astype(int))
eval_y = label_encoder.fit_transform((df_eval['rate'] >= 0).astype(int))
test_y = label_encoder.fit_transform((df_test['rate'] >= 0).astype(int))
```

Before You Run

Import Libraries

Load Data

Preprocess

```python
# clean_text function
def clean_comment(text, allspace=True, punc=True, sentence=True, only_persian=True):
    #remove halph space, new line ('\n') and '\r'
    text = text.replace('\u200c', ' ').replace('\n', '').replace('\r', '')
    # remove punctuations
    text = re.sub(symbols_complete_reg, "", text)
    # remove arabic letters
    text = remeove_arabic(text)
    # convert spaces to a one space and delete leading and trailing spaces
    text = re.sub("(\s)+", " ", text)
    text = text.strip()
    return text
```

FastText Embedding

Download Skipgram Model

Load FastText Model

There exist two different **Fastext** pre-trained model which can be used.

```
# Model 1: Dimension: 100 from # https://github.com/taesiri/PersianWordVectors
# SKIPGRAM_MODEL_FILE_ID_1 = '1wPnMG9_GNUVdSgbznQziQc5nMWI3QKNz'
# !gdown --id $SKIPGRAM_MODEL_FILE_ID

# Model 2: Dimension: 300 from https://fasttext.cc/docs/en/pretrained-vectors.html
!wget https://dl.fbaipublicfiles.com/fasttext/vectors-wiki/wiki.fa.zip
! unzip wiki.fa.zip
! rm -rf wiki.fa.zip
! rm -rf wiki.fa.vec
```

```
EMBEDDING_LEN = 300 # 100 for Model 1 and 300 for Model 2
```

FastText Embedding

Download Skipgram Model

Load FastText Model

```
[ ]  # Fit Keras Tokenizer on comments
     comments = df_train['clean_comment'].values
     tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=3000)
     tokenizer.fit_on_texts(comments)

     vocab_size = len(tokenizer.word_index) + 1
     print('Vocabulary Size : {}'.format(vocab_size))
```

FastText Embedding

Download Skipgram Model

Load FastText Model

Create Word Embedding Matrix for all used words.

```
[ ] # initial embedding matrix
    embedding_matrix = np.zeros((vocab_size, EMBEDDING_LEN))

    for word, i in tokenizer.word_index.items():
      embedding_vector = model_skipgram.get_word_vector(word)
      # words that cannot be found will be set to 0
      if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

    print(f"Embedding Matrix Shape is: {embedding_matrix.shape}")
```

LSTM Model Architecture

Fit LSTM Model

Here is implemented deep LSTM network architecture.

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 616, 300)          1296900

bidirectional (Bidirectiona  (None, 616, 600)          1442400
l)

dropout (Dropout)            (None, 616, 600)          0

bidirectional_1 (Bidirectio  (None, 64)                162048
nal)

dropout_1 (Dropout)          (None, 64)                0

dense (Dense)                (None, 300)               19500

dropout_2 (Dropout)          (None, 300)               0

dense_1 (Dense)              (None, 1)                 301

=================================================================
Total params: 2,921,149
Trainable params: 2,921,149
Non-trainable params: 0
_________________________________________________________________
```

Classification Report

```
pred_1 = model_1.predict(test_padded_sequence)
y_pred_1 = np.array((pred_1 > 0.5).astype(int)[:,0])
print(confusion_matrix(y_true=test_y, y_pred=y_pred_1))
print(classification_report(y_true=test_y, y_pred=y_pred_1))
```

```
[[ 21  31]
 [ 14 104]]
              precision    recall  f1-score   support

           0       0.60      0.40      0.48        52
           1       0.77      0.88      0.82       118

    accuracy                           0.74       170
   macro avg       0.69      0.64      0.65       170
weighted avg       0.72      0.74      0.72       170
```

CNN Model Architecture

Fit CNN Model

Here is implemented deep CNN network architecture.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 300)         1296900

 conv1d (Conv1D)             (None, None, 256)         230656

 global_max_pooling1d (Globa  (None, 256)              0
 lMaxPooling1D)

 dense_2 (Dense)             (None, 256)               65792

 dense_3 (Dense)             (None, 1)                 257

=================================================================
Total params: 1,593,605
Trainable params: 296,705
Non-trainable params: 1,296,900
_________________________________________________________________
```

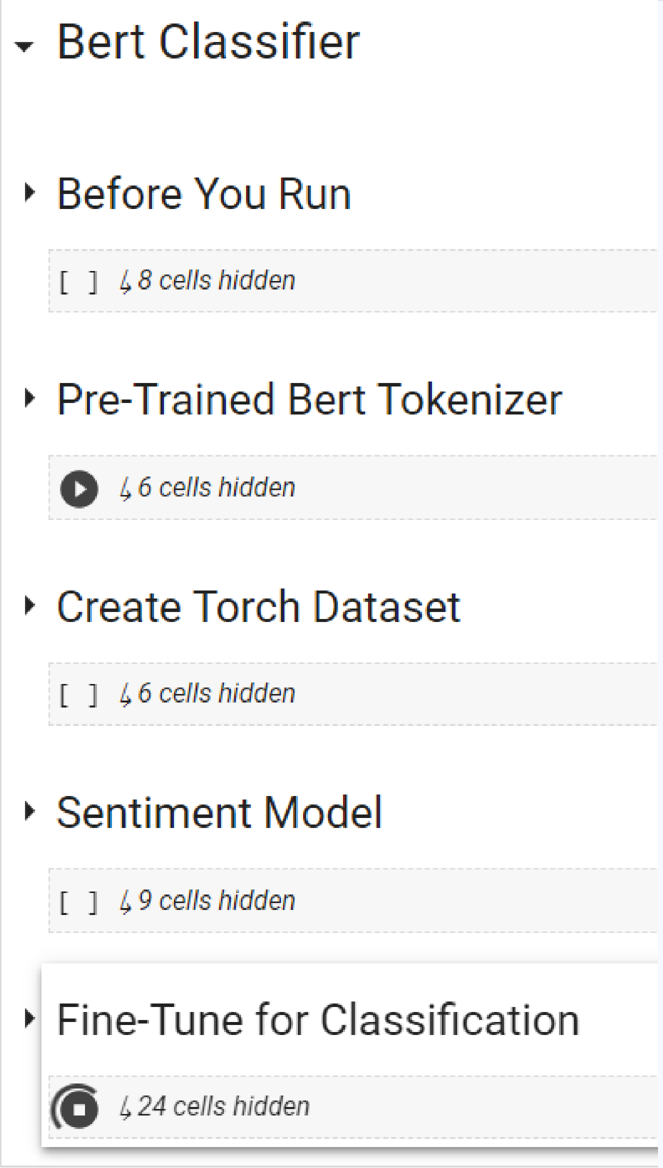
Classification Report

```
[ ] pred_2 = model_2.predict(test_padded_sequence)
    y_pred_2 = np.array((pred_2 > 0.5).astype(int)[:,0])
    print(confusion_matrix(y_true=test_y, y_pred=y_pred_2))
    print(classification_report(y_true=test_y, y_pred=y_pred_2))
```

```
[[ 18  34]
 [  4 114]]
              precision    recall  f1-score   support

           0       0.82      0.35      0.49        52
           1       0.77      0.97      0.86       118

    accuracy                           0.78       170
   macro avg       0.79      0.66      0.67       170
weighted avg       0.78      0.78      0.74       170
```

BERT

“Two different **BERT** approach have implemented one using **Pytorch** from scratch and other using **Ktrain** library…



Bert1

- **Before You Run** block covers tasks related to required libraries and Reading Data and General Model Config.
- **Pre-Trained Bert Tokenizer** block covers loading BERT Tokenizer model and demonstrate encoding with mentioned model.
- **Create Torch Dataset** block is devoted to create Torch compatible Dataset for batching and training purposes.
- **Sentiment Model** block loads data using data loader implemented in last step and run task on data without any fine-tuning as an example.
- **Fine-Tune for Classification** block fine-tune BERT for task over available data.

Before You Run

- Install Required Libraries
- Import Required Libraries
- Read Data
- Model General Config

```
[ ]  PATH = 'data/'
     PATH = PATH.rstrip('/')

     # Train
     df_train = pd.read_csv(PATH + '/train.csv')
     df_train.columns = ['index', 'comment', 'rate']

     # Evaluation
     df_eval = pd.read_csv(PATH + '/eval.csv')
     df_eval.columns = ['index', 'comment', 'rate']

     # Test
     df_test = pd.read_csv(PATH + '/test.csv')
     df_test.columns = ['index', 'comment', 'rate']

     # Create Lables
     label_encoder = LabelEncoder()

     train_y = label_encoder.fit_transform((df_train['rate'] >= 0).astype(int))
     eval_y = label_encoder.fit_transform((df_eval['rate'] >= 0).astype(int))
     test_y = label_encoder.fit_transform((df_test['rate'] >= 0).astype(int))
```

Before You Run

- Install Required Libraries
- Import Required Libraries
- Read Data
- Model General Config

```python
# Model Config
MAX_LEN = 128
TRAIN_BATCH_SIZE = 32
VALID_BATCH_SIZE = 32
TEST_BATCH_SIZE = 16

EPOCHS = 10
# Every EEVERY_EPOCH print status
EEVERY_EPOCH = 10
LEARNING_RATE = 2e-5
CLIP = 0.0

MODEL_NAME_OR_PATH = 'HooshvareLab/bert-fa-base-uncased'
```

Pre-Trained Bert Tokenizer

BERT Tokenizer

Sample

```
encoding = tokenizer.encode_plus(
    sample,
    max_length=32,
    truncation=True,
    add_special_tokens=True,
    return_token_type_ids=True,
    return_attention_mask=True,
    padding='max_length',
    return_tensors='pt'
)

print(f'Keys: {encoding.keys()}\n')
for k in encoding.keys():
    print(f'{k}:\n{encoding[k]}')
```

```
[6] sample = 'از این محصول بدم اومده!'
```

```
[7] tokens = tokenizer.tokenize(sample)
    token_ids = tokenizer.convert_tokens_to_ids(tokens)

    print(f'Tokens: {tokenizer.convert_tokens_to_string(tokens)}')
    print(f'Token IDs: {token_ids}')
```

```
Tokens: از این محصول بدم اومده !
Token IDs: [2791, 2802, 3573, 19910, 36711, 1001]
```

```python
class SentimentDataset(torch.utils.data.Dataset):
    """ Create a PyTorch dataset for Digikala SentimentDataset. """

    def __init__(self, tokenizer, comments, targets, is_predict=False, max_len=128):
        self.comments = comments
        self.targets = targets
        self.tokenizer = tokenizer
        self.max_len = max_len
        self.is_predict = is_predict
```

Create Torch Dataset

Load Torch Dataset

Sample Batch

Dataset batches contain these data.

```
'comment': ['ی مجازی دیگر هستند\u200cنیز ذاتا دو هسته ای با توان ساختن دو هسته Core i3 و Core i5 پردازنده های.',
 'سلام به دوستای عزیزم \r\nعزاداری هاتون قبول باشه',
 'کلا پولتون رو دور نریزید',
 'از صمیم قلب امیدوارم دایانا با کارن بمونه و پوریا رو فراموش کنه',
 'آنطور که اپل ادعا می کند آیپاد شافل دارای طراحی فوق العاده است، که البته ادعایی غیر واقعی نیست.',
 'در کل کفش بدی نیست ولی من خودم دادشتم دور دوخت ولی با این قیمت این کفش ارزش خرید نداره',
 'بر روی این صفحه نمایش، عملکرد آن در زوایای مختلف نیز بسیار مناسب و قابل قبول است IPS به دلیل وجود پنل.',
```

```
'attention_mask': tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]]),
```

```
'input_ids': tensor([[    2,  7639,  6343,  ...,      0,      0,      0],
        [    2,  4285,  2789,  ...,      0,      0,      0],
        [    2,  5569, 84778,  ...,      0,      0,      0],
        ...,
        [    2, 40291, 14341,  ...,      0,      0,      0],
        [    2,  3805,  3805,  ...,      0,      0,      0],
        [    2,  2831,  5824,  ...,      0,      0,      0]]),
```

```
'targets': tensor([1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0,
        1, 1, 1, 1, 1, 0, 1, 1]),
```

Sentiment Model

Empty GPU VRAM

Load Sentiment Model

Make Prediction Based on Sample Batch

Network consist of:

- BERT model
- Dropout layer
- Classifier layer

```
class SentimentModel(nn.Module):

    def __init__(self, config):
        super(SentimentModel, self).__init__()

        self.bert = BertModel.from_pretrained(MODEL_NAME_OR_PATH, return_dict=False)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)

    def forward(self, input_ids, attention_mask, token_type_ids):
        _, pooled_output = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids)

        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        return logits
```

```python
pt_model = SentimentModel(config=config)
pt_model = pt_model.to(device)
```

Sentiment Model

Empty GPU VRAM

Load Sentiment Model

Make Prediction Based on Sample Batch

```python
result = pt_model(sample_batch['input_ids'].to(device),
                  sample_batch['attention_mask'].to(device),
                  sample_batch['token_type_ids'].to(device))

# Make Prediction
_, predictions = torch.max(result, dim=1)

for i, comment in enumerate(sample_batch['comment']):
    print(str(comment) + " : " + str(predictions[i]))
```

```
پردازنده های Core i5 و Core i3 نیز ذاتا دو هسته ای با توان ساختن دو هسته‌ی مجازی دیگر هستند. : tensor(1, device='cuda:0')
سلام به دوستای عزیزم
عزاداری هاتون قبول باشه : tensor(1, device='cuda:0')
کلا پولتون رو دور نریزید : tensor(0, device='cuda:0')
از صمیم قلب امیدوارم دایانا با کارن بمونه و پوریا رو فراموش کنه : tensor(0, device='cuda:0')
آنطور که اپل ادعا می کند آیپاد شافل دارای طراحی فوق العاده است، که البته ادعایی غیر واقعی نیست. : tensor(1, device='cuda:0')
در کل کفش بدی نیست ولی من خودم دادشتم دور دوخت ولی با این قیمت این کفش ارزش خرید نداره : tensor(0, device='cuda:0')
به دلیل وجود پنل IPS بر روی این صفحه نمایش، عملکرد آن در زوایای مختلف نیز بسیار مناسب و قابل قبول است. : tensor(1, device='cud
10 روز استفاده میگفت که بسیار دانگل خوبی است. لگ ندارد. افت کیفیت ندارد. برد خوبی دارد در حد 5 متر، خیلی راحت متصل میشود و داغ نمیکند
در شگفت انگیز ارزش خرید بالایی دارد.
دیجی جان شگفت انگیزش کن من و چندتا دوستان میخوایم خرید کنیم. : tensor(0, device='cuda:0')
چقد فاصله داریم!!! : tensor(0, device='cuda:0')
خیلی زود شارژ خالی میکنه : tensor(1, device='cuda:0')
گوشت استفاده شده در غذاها بوی نامطبوع داشت و سفت و نایخته بود : tensor(1, device='cuda:0')
```

Based on define **`EEVERY_EPOCH`** configuration, we run **Evaluation** method and then print training status

```
Evaluation Process: 100% 7/7 [00:03<00:00, 2.50it/s]
Train Loss: 0.137851...
Train Acc: 0.957...
Valid Loss: 1.009251...
Valid Acc: 0.755...
```

We Save the model while training to keep track of training and reduce the risk of losing trained parameters and afterward we load the model again and do the predictions.

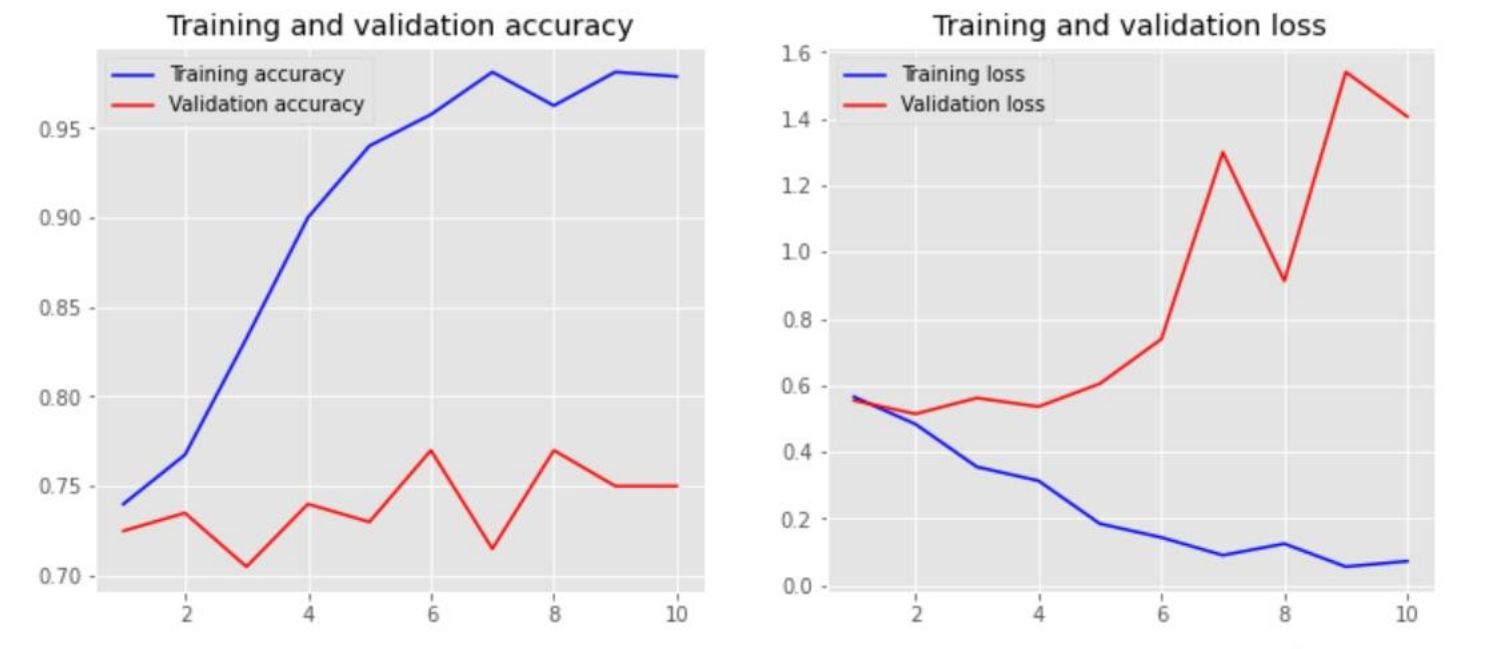
```
pt_model.load_state_dict(torch.load('best-model.bin'))
```

```
test_comments = df_test['comment'].to_numpy()
predictions = predict(pt_model, test_comments, tokenizer, max_len=128)
```

Fine-Tune for Classification

- Import Required Libraries
- Util Method
- Define Train Method
- Define Evaluation Method
- Training Model
- Define Predict Method
- Predict Test Data
- Reports
  - Accuracy
  - F1-Score
  - Other Reports
  - Model History Plot

As you already know, since dataset labels have randomly been assigned, even though the training model can converge but it does not guarantee test results.



Classification Report

```
print(classification_report(test_y, predictions, target_names=['positive','negative']))
```

```
              precision    recall  f1-score   support

    positive       0.68      0.52      0.59        52
    negative       0.81      0.89      0.85       118

    accuracy                           0.78       170
   macro avg       0.74      0.70      0.72       170
weighted avg       0.77      0.78      0.77       170
```

```python
# build, train, and validate model (Transformer is wrapper around transformers library)

MODEL_NAME = 'HooshvareLab/distilbert-fa-zwnj-base'  # replace this with model of choice
transformer_model = text.Transformer(MODEL_NAME, maxlen=500, class_names=class_names)
trn = transformer_model.preprocess_train(x_train, y_train)
val = transformer_model.preprocess_test(x_eval, y_eval)
classifier_model = transformer_model.get_classifier()
learner = ktrain.get_learner(classifier_model, train_data=trn, val_data=val, batch_size=6)
learner.fit_onecycle(5e-5, 4)
```

Bert2

- Training using Ktrain is straightforward. We only need to feed training and evaluation data and specify the count of epochs and learning rate to fine-tune BERT for the specified task.

```python
# build, train, and validate model (Transformer is wrapper around transformers library)

MODEL_NAME = 'HooshvareLab/distilbert-fa-zwnj-base'  # replace this with model of choice
transformer_model = text.Transformer(MODEL_NAME, maxlen=500, class_names=class_names)
trn = transformer_model.preprocess_train(x_train, y_train)
val = transformer_model.preprocess_test(x_eval, y_eval)
classifier_model = transformer_model.get_classifier()
learner = ktrain.get_learner(classifier_model, train_data=trn, val_data=val, batch_size=6)
learner.fit_onecycle(5e-5, 4)
```

We load the pre-trained Pars BERT model with a random initialized final Dense layer. The weights of all the layers of the model, including the dense layer, will be updated during backpropagation since we have not frozen any layers. Additionally, **get_learner** creates a learner object with train and validation data, which can be used to fine-tune the classifier.

Classification Report

```
preprocessing test...
language: fa
test sequence lengths:
        mean : 24
        95percentile : 55
        99percentile : 121

              precision    recall  f1-score   support

    Positive       0.74      0.48      0.58        52
    Negative       0.80      0.92      0.86       118

    accuracy                           0.79       170
   macro avg       0.77      0.70      0.72       170
weighted avg       0.78      0.79      0.77       170


array([[ 25,  27],
       [  9, 109]])
```

AT END

There exists another file named **torture_data.Ipynb** which we got more familiar with problem data in that file.