

بسم الله الرحمن الرحيم

## پروژه پردازش متن

# برای محصولات دیجیکالا

نسخه شماره ۴

تیم ارائه دهنده: داده کاوان فونیکس

ناظر و سرپرست تیم: دکتر علیرضا وفایی صدر

درخواست دهنده: ستاد علوم شناختی IPM

بهار ۱۳۹۹

## فهرست مطالب

۱	-----	تعریف پروژه
۲	-----	تولید متن با معماری pre-trained
۳	-----	برگرداندن داده های تولید شده به جملات
۴	-----	میزان شباهت بین هر کلاس
۵	-----	تولید لغات ابری به شکل درختی
۶	-----	جمع بندی و تحقیقات آتی

## تعریف پروژه

- در ادامه فاز دوم و بحث های صورت گرفته، به ادامه کار تولید متن با لایک های بالا پرداختیم.
- تولید متن با همان مدل قبلی classifier ولی با نگهداری وزن ها و استفاده از مدل pre-trained
  - معیاری برای میزان شباهت بین هر کلاس
  - تولید لغات ابری با اشکال مختلف (درختی)

## تولید متن با معماری pre-trained قبلی

برای اینکه کامنت های تولید کنیم classifier آن ها در کلاس با لایک بالا قرار دهد، از همان مدل قبلی که دقت ۷۵٪ برای چهار کلاسه به ما داد استفاده کردیم. ولی این بار به جای چهار کلاس، دو کلاس verylike-dislike و like-dislike را انتخاب کرده و به عنوان کلاس با برچسب ۰ و ۱ به مدل دادیم. داده های این دو کلاس را روی این مدل دو کلاسه از اول train کردیم.

مدل bi-classification را که ترین کردیم روی داده های ترین دقت ۹۵٪ و روی داده ولیدیشن دقتی بین ۷۴٪ تا ۸۰٪ داد. سپس برای تولید نویز، لایه embedding همین معماری را برداشتیم بعد از آن یکسری لایه trainable قرار دادیم که کار generator برای تولید داده های نویز را انجام دهد و تمامی لایه های معماری classifier قبلی را Freeze کرده، و وزن های لایه های classifier را save کردیم و لایه های آن را از حالت trainable خارج کردیم تا بتوان داده های نویز را با همان دقت قبل به کلاس verylike بیاندازد.

داده های نویز از لایه های trainable عبور کرده و تولید می شود و سپس وارد classifier فریز شده با وزن های قبلی، می شود که حکم discriminator را دارد که تشخیص دهد این داده های نویز وارد کلاس verylike شده اند یا نه.

معماری کلی برای تبدیل داده های نویز به کلاس لایک بالا دقتی معادل ۹۷٪ داده است و چون از مدل قبلی که pre-trained شده استفاده کردیم و داده های واقعی قبل از تولید مدل دیده شده بود، به این خاطر دقت بالایی به دست آورده ایم.

برای تولید داده های نویز با کلاس، داده های واقعی را گرفته و در ستون کامنت هایی که کلاس verylike داشتند (ستون تارگت مقدار ۱ داشت) بعد از Vectorize کردن، به جای کامنت های واقعی، مقادیر رندوم نویز در همان بازه ای مقادیر vectorize شده برای متون کامنت، قرار دادیم.

معماری مربوط به تولید نویز و تبدیل این داده های به کامنت هایی با کلاس verylike در فایل [model\\_final\(Fake Comments\).ipybn](#) قرار دارد که توسط خانم رقیه فرجی انجام شده است.

مسیر فایل:

Report2-> roghaye\_faraji

## برگرداندن داده های تولید شده به جملات

قرار شد بعد از تولید داده هایی با کامنت بالا، آن داده های را تبدیل به جملات کنیم و ببینیم که مدل پیشنهادی، چه جملات و کامنت هایی را تولید کرده است.

در معماری خانم فرجی، برای این کار، خروجی آخرین لایه قبل از لایه freeze شده در descriminator را گرفتیم و آن را در Inverse ماتریس embedding که وزن های آن را نگهداشتی بودیم ضرب کردیم و ماتریسی شبیه one-hot vector به دست آمد که حاوی ۲۰ سطر (طول یک جمله) و حدود ۵۳۰۰۰ ستون (هر ستون معادل یک کلمه vectorize شده است) بود. در هر سطر، ستونی را که بیشترین عدد را داشت به عنوان کلمه انتخابی برداشت و از word2vec دیکشنری Inverse که کلمات را به اعداد تبدیل کرده بودیم استفاده کرده و معادل کلمه آن عدد در آن ستون را دریافت کردیم. برای هر سطر به همین ترتیب عمل کرده و بزرگترین عدد موجود در هر ستون را به عنوان کلمه انتخابی در آن سطر، برگرداندیم و به این ترتیب ۲۰ کلمه انتخاب شد که معادل یک کامنت فیک تولید شده است.

خروجی تولید شده از داده های fake که در دسته verylike قرار میگیرند را در زیر برای ۵ کامنت مشاهده می کنید: کلمات و صفاتی که قرمز رنگ نشان داده شده اند کلمات با معنی ای هستند که امکان لایک خوردن را فراهم کرده اند.

کامنت ۱	کامنت ۲	کامنت ۳	کامنت ۴	کامنت ۵
[، 'دماش' [، 'علیه' [، 'دیجیکالایم' [، 'البت' [، 'چسپندگی' [، 'استحکام' [، 'یسری' [، 'عماه' [، 'اسپلیتر' [، 'عسلشہ' [، 'بازدهه' [، 'قیمش' [، 'چنگالش' [، 'جبان' [، 'همراهانش' [، 'نظرمن' [، 'مدادی' [، 'اقدام' [، 'جیفه' [، 'یورو' [ ]]	[، 'ضعیفیه' [، 'خریدکالا' ۲۴۰' [، 'هیجان' [، 'انکیز' [، 'تاحلا' [، 'ومناسب' [، 'پشیمان' [، 'پک' [، 'موzd' [، 'معرکه' [، '۹۴۵۰' [، 'تواین' [، 'حجمش' [، 'داشتنیه' [، 'ارشمندی' [، 'چتری' [، 'سولفاتش' [، 'سامسونگش' [، 'ششم' [ ]]	[، 'بولمو' [، 'انصفان' [، 'مان' [، 'افتایی' [، 'افق' [، 'افتابی' [، 'باحالیه' [، 'موافق' [، 'آسپلیتر' [، 'بدنیزی' [، 'خوشکل' [، 'موافق' [، 'آسپلیتر' [، 'چنگالش' [، 'بنزینش' [، 'افتایی' [، 'چنگالش' [، 'چنیبی' [، 'چتری' [، 'تواین' [، 'استاندار' [، 'ششم' [ ]]	[، 'اندونزی' [، 'خریدکالا' ۲۴۰' [، 'اماکانتش' [، 'غلط' [، 'جاروی' [، 'قابلیتهاش' [، 'پاییز' [، 'دانم' [، 'کوهنوری' [، 'تصاویرش' [، 'متقل' [، 'نرو' [، 'میرسد' [، 'چسبونه' [، 'چارزکردم' [، 'توام' [، 'توو' [، 'توو' [، 'چسبونه' [، 'بادمجنون' [، 'بادمجنون' [، 'پیشمون' [، 'جی' ۷' [ ]]	[، 'چنگالش' [، 'بلنیچر' [، 'احتیاجتون' [، 'اسپلیتر' [، 'افتضاحی' [، 'معدرت' [، 'بابات' [، 'قلبلمه' [، 'آمپرش' [، 'بدردنمیخوره' [، 'سامسونگش' [، 'معضل' [، 'یکسالهاین' [، 'کارمون' [، 'یکسالهاین' [، 'خکین' [، 'درهر' [، 'شارژکردم' [، 'تمومه' [، 'کر' [، 'شارژکردم' [، 'عماه' [، 'وسیعی' [، 'هروز' [، 'بیصدا' [، 'جهات' [، 'هرکسی' [، 'اوکی' [، 'همیار' [، 'پیشنهادمیکنم' [، 'جی' ۷' [ ]]

البته دو کامنت تولید شده اول، با طول ۲۰ و سه کامنت بعدی با طول جمله ۳۰ تولید شده است.

اینکار نیز توسط خانم فرجی در همان فایل تولید کامنت، صورت گرفته است.

کد مدل زیر مشاهده می کنید:

```
model = Sequential()
model.add(Embedding(input_dim =53019, output_dim = 30, input_length = 30, name='embed'))
model.add(Dense(50, name='layer11'))

model.add(Conv1D(filters = 150, kernel_size = ( 8), strides = 1, padding = "valid", name='layer_1'))

model.add(Dropout(0.2, name='layer_2'))

model.add(GlobalMaxPooling1D(name='layer_3'))
model.add(Dense(50, activation = 'relu', kernel_initializer='he_normal',
kernel_regularizer=regularizers.l1(0.01), bias_regularizer=regularizers.l2(0.01), name='layer_4'))

model.add(BatchNormalization(name='layer_5'))
model.add(Dropout(0.2, name='layer_6'))
model.add(Dense(10, activation = 'relu', name='layer_7'))
model.add(BatchNormalization(name='layer_8'))
model.add(Dropout(0.2, name='layer_9'))
model.add(Dense(2, activation='softmax' ,name='layer_10'))

adam = Adam()

model.compile(loss='binary_crossentropy', optimizer=adam , metrics=['accuracy'])
model.summary ()
```

شکل زیر نیز معماری gan برای تولید کامنت های لایک بالا می باشد:

```
model3 = Sequential()
model3.add(Embedding(input_dim =53019, output_dim = 30, input_length =
30, name='embed', trainable=False))

model3.add(Dense(36, input_shape=(30,), activation='sigmoid'))
model3.add(BatchNormalization())

model3.add(Dense(100, activation='sigmoid'))
model3.add(BatchNormalization())

model3.add(Dropout(0.2))
model3.add(Flatten())
model3.add(Dense(30*30, activation='sigmoid', name='generate_layer1'))
model3.add(BatchNormalization())

model3.add(Reshape((30, 30), input_shape=(900,1), name='generate_layer2'))

model3.add(Dense(50, name='layer11', trainable=False))
model3.add(Conv1D(filters = 150, kernel_size = ( 8), strides = 1, padding =
"valid", name='layer_1', trainable=False))

model3.add(Dropout(0.2, name='layer_2', trainable=False))

model3.add(GlobalMaxPooling1D(name='layer_3', trainable=False))
model3.add(Dense(50, activation = 'relu', kernel_initializer='he_normal',
kernel_regularizer=regularizers.l1(0.01), bias_regularizer=regularizers.l2(0.01),
name='layer_4', trainable=False))

model3.add(BatchNormalization(name='layer_5', trainable=False))
model3.add(Dropout(0.2, name='layer_6', trainable=False))
model3.add(Dense(10, activation = 'relu', name='layer_7', trainable=False))
model3.add(BatchNormalization(name='layer_8', trainable=False))
model3.add(Dropout(0.2, name='layer_9', trainable=False))
model3.add(Dense(2, activation='sigmoid' ,name='layer_10', trainable=False))

adam = Adam()

model3.compile(loss='binary_crossentropy', optimizer=adam , metrics=['accuracy'])
```

## معیاری برای میزان شباهت بین هر کلاس

در بار اول که دو کلاس like-verylike و like-dislike را با هم ادغام کرده و تبدیل به یک کلاس کردیم و دو کلاس verylike-verydislike و verylike-dislike را نیز با هم یکی کردیم و اقدام به تولید کامنت کردیم، مدل bi-classifier دقت معادل ۵۰٪ یا رندوم و تصادفی تولید میکرد. این احتمال را دادیم که بعد از ادغام هر دو کلاس، میزان شباهت و نزدیکی کلاسهای با هم بسیار زیاد است و تفاوت چندانی بین دو کلاس جدید like و verylike وجود ندارد و مدل نمی‌تواند تمایزی بین دو کلاس قائل شود.

برای پی بردن به این موضوع، آقای غفوریان الگوریتمی را برای شباهت بین کامنت‌ها در هر چهار کلاس به صورت جدا به دست آورد. به این صورت که برای هر کلاس، کلمه‌های موجود در کامنت‌های آن کلاس را گرفته و در بین کلاسهای دیگر جستجو میکرد تا ببیند این کلمه در کامنت‌های کلاس دیگر، چند بار تکرار شده است. و برای هر کلمه این کار را انجام داده و تعداد تکرارهای هر کلمه را در آن کلاس با هم جمع کرد.

این کار را بین هر کلاس با سه کلاس دیگر انجام داد و برای مجموع تعداد کلمات تکرار شده بین هر دو کلاس، یک عدد به دست آمد که این عدد را تقسیم بر تعداد کل کلمات موجود در کامنت‌ها (۲۰۰۰ کلمه منحصر به فرد) کرد و توانست عددی به دست بیاورد که نشانگر میزان شباهت و نزدیکی بین کلاس‌ها باشد.

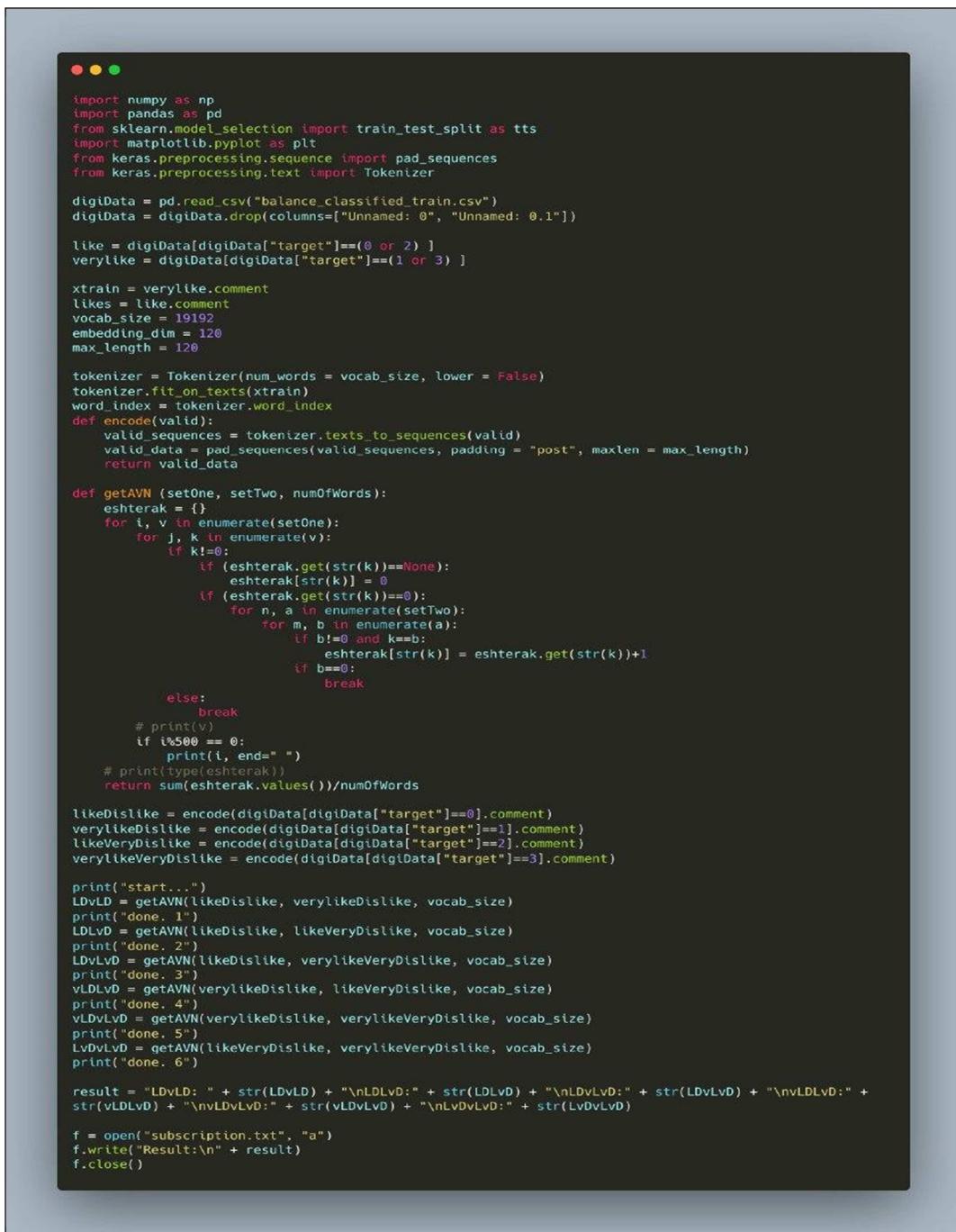
میزان شباهت بین هر دو کلاس را در جدول زیر مشاهده می‌کنید:

نام کلاس‌ها	مقدار شباهت و نزدیکی دو کلاس به هم
like-dislike & verylike-dislike	۲۴
like-dislike & like-verydislike	۱۹
like-dislike & verylike-verydislike	۲۴
verylike-dislike & like-verydislike	۱۹
verylike-dislike & verylike-verydislike	۲۴
like-verydislike & verylike-verydislike	۲۴

اگر هر کدام از این اعداد را در تعداد کل کلمات منحصر به فرد ۲۴۰۰۰ ضرب کنیم مجموع تعداد کلمات تکرار شده در بین دو کلاس را به ما میدهد که اعداد بزرگی است.

این میزان اعداد برای شباهت اعداد بزرگی هستند که نشان میدهد کلاسها از لحاظ تعداد کلمات مشترک بسیار به هم نزدیک هستند و برای همین ادغام هر دو کلاس باعث بیشتر شدن این شباهت شد.

الگوریتم آقای غفوریان در شکل زیر دیده می شود:



```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split as tts
import matplotlib.pyplot as plt
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer

digiData = pd.read_csv("balance_classified_train.csv")
digiData = digiData.drop(columns=["Unnamed: 0", "Unnamed: 0.1"])

like = digiData[digiData["target"]==0]
verylike = digiData[digiData["target"]==1]

xtrain = verylike.comment
likes = like.comment
vocab_size = 19192
embedding_dim = 120
max_length = 120

tokenizer = Tokenizer(num_words = vocab_size, lower = False)
tokenizer.fit_on_texts(xtrain)
word_index = tokenizer.word_index
def encode(valid):
    valid_sequences = tokenizer.texts_to_sequences(valid)
    valid_data = pad_sequences(valid_sequences, padding = "post", maxlen = max_length)
    return valid_data

def getAVN (setOne, setTwo, numOfWords):
    eshterak = {}
    for i, v in enumerate(setOne):
        for j, k in enumerate(v):
            if k!=0:
                if (eshterak.get(str(k))==None):
                    eshterak[str(k)] = 0
                if (eshterak.get(str(k))==0):
                    for n, a in enumerate(setTwo):
                        for m, b in enumerate(a):
                            if b!=0 and k==b:
                                eshterak[str(k)] = eshterak.get(str(k))+1
                            if b==0:
                                break
                else:
                    break
    # print(v)
    if numOfWords == 0:
        print(i, end=" ")
    # print(type(eshterak))
    return sum(eshterak.values())//numOfWords

likeDislike = encode(digiData[digiData["target"]==0].comment)
verylikeDislike = encode(digiData[digiData["target"]==1].comment)
likeVeryDislike = encode(digiData[digiData["target"]==2].comment)
verylikeVeryDislike = encode(digiData[digiData["target"]==3].comment)

print("start...")
LDvLD = getAVN(likeDislike, verylikeDislike, vocab_size)
print("done. 1")
LDLvD = getAVN(likeDislike, likeVeryDislike, vocab_size)
print("done. 2")
LDvLvD = getAVN(likeDislike, verylikeVeryDislike, vocab_size)
print("done. 3")
vLDvLD = getAVN(verylikeDislike, likeVeryDislike, vocab_size)
print("done. 4")
vLDLvD = getAVN(verylikeDislike, verylikeVeryDislike, vocab_size)
print("done. 5")
LvDvLD = getAVN(likeVeryDislike, verylikeVeryDislike, vocab_size)
print("done. 6")

result = "LDvLD: " + str(LDvLD) + "\nLDLvD: " + str(LDLvD) + "\nLDvLvD: " + str(LDvLvD) + "\nvLDvLD: " + str(vLDvLD) + "\nvLDLvD: " + str(vLDLvD) + "\nLvDvLD: " + str(LvDvLD)

f = open("subscription.txt", "a")
f.write("Result:\n" + result)
f.close()
```

کد مربوط به این الگوریتم در فایل `CalculatTheSubscription.py` نوشته شده است.

مسیر فایل: Report2-> VahidGhafourian

## تولید لغات ابری با اشکال مختلف (درختی)

در این بخش به تولید لغات ابری یا word cloud پرداختیم اما به شکل درختی که توسط خانم زهرا نفریه انجام شد. از همان کتابخانه قبلی Persian\_wordcloud استفاده کردیم اما حالت mask آن را روی گذاشتیم تا شکل تولید شده حالت درختی به خود گیرد.

کد در شکل زیر مشاهده می شود:

```
With mask Example
=====
Generating a square wordcloud from the US constitution using default arguments.
"""
from persian_wordcloud.wordcloud import STOPWORDS, PersianWordCloud

from os import path
from PIL import Image
import numpy as np

d = path.dirname('/home/zahra-linux/Desktop/')

text = open(path.join(d, 'freq_words.txt'), encoding='utf-8').read()

# Add another stopword
twitter_mask = np.array(Image.open(path.join(d, "tree.jpeg")))
STOPWORDS.add('می')
stopwords = set(STOPWORDS)

# Generate a word cloud image

wordcloud = PersianWordCloud(
    only_persian=False,
    max_words=200,
    stopwords=stopwords,
    margin=0,
    width=800,
    height=800,
    min_font_size=1,
    max_font_size=500,
    random_state=True,
    background_color="white",
    mask=twitter_mask
).generate(text)

image = wordcloud.to_image()
image.show()
image.save('tree.png')
```

فایل کدهای لغات ابری و تصاویر آن در مسیر زیر موجود است:

Report2-> Zahra\_Nafarieh->word\_cloud

شکل های زیر توسط کد بالا یک با پس زمینه درخت و دیگری با پس زمینه پرنده توییتر تولید شده است:



## جمع بندی و تحقیقات آتی

در این گزارش، به بررسی تولید کامنت های فیک با کلاس verylike پرداختیم. همچنین میزان شباهت بین هر چهار کلاس را سنجیدیم و شکل جدید از تشکیل لغات ابری را به دست آوردیم.

کلیه فایل های مربوط به این گزارش نسخه ۴ در گیت هاب زیر و در فolder Report2 موجود است.

<https://github.com/phoenix-dataminers/Digikala2>