چالش سوم: Toxic Comment Classification Challenge

فراهم‌آورنده: سینا حیدری

زمستان ۹۸

برای مشاهده‌ی جزئیات این چالش در کگل به لینک زیر مراجعه کنید.

https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview

۱ در مورد این چالش

معمولا، به‌اشتراک گذاردن مسائل شخصی کار دشواری انسان‌ها است؛ تهدید آزار و اذیت آنلاین باعث شده مردم از انجام این کار در پلتفرم‌های آنلاین صرف نظر کنند. پلتفرم‌ها برای آرامش بخشیدن به گفت‌وگوها با مشکلاتی دست‌وپنجه نرم می‌کنند؛ تا جایی که بسیاری از انجمن‌ها، قابلیت نظر دادن کاربران را از دسترس خارج کرده‌اند.
تیم Conversation AI، یک ابتکار دانش‌بنیان است که توسط Jigsaw و Google پایه‌گذاری شده است. این تیم در حال کار کردن روی ابزارهایی است که گفت‌وگوی آنلاین را بهبود دهد. یکی از زمینه‌هایی که این تیم روی آن تمرکز کرده، رفتارهای منفی آنلاین همچون نظرات سمی است. اینگونه نظرات بی‌ادبانه‌اند و همچنین، موجب بی‌احترامی به کاربران و در نهایت، ترک آن‌ها از گفت‌وگو می‌شود. تیم Conversation API تا کنون محدوده‌یی از مدل‌ها را از طریق Perspective API در اختیار عامه قرار داده است–از جمله مدل سمیت–. اما مدل‌های کنونی هنوز دچار خطا شده و به‌کاربر اجازه‌ی انتخاب نوع سمیت را نمی‌دهند–برای نمونه؛ بعضی پلتفرم‌ها با ناسزا مشکلی ندارند؛ اما با نوع‌های دیگر سمیت، اینگونه برخورد نمی‌کنند–.
در این رقابت، ما به چالش کشیده شده‌ایم تا یک مدل چندسر، با توانایی بهتر از مدل کنونی برای تشخیص انواع سمیت–از جمله؛ تهدید، فحاشی و تنفرهای بر پایه‌ی هویت–بسازیم. ما یک مجموعه‌داده‌ی نظرات که از گفت‌وگوی صفحه‌های ویرایش ویکی‌پدیا جمع‌آوری شده‌اند را استفاده خواهیم کرد.
هشدار: مجموعه داده‌ی این رقابت شامل متنی است که ناسزا، مبتذل و توهین‌آمیز است.

۲ در مورد این گزارش

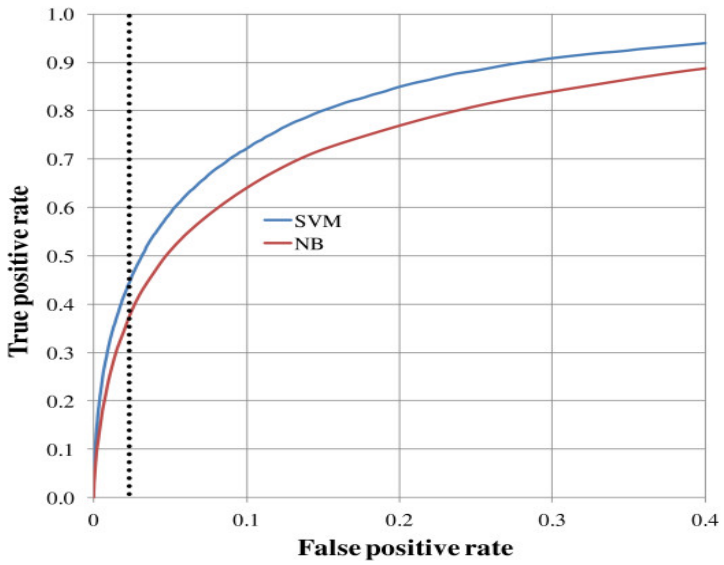
این گزارش چگونگی استفاده از NBSVM[۱] در جهت ساختن یک پایه‌ی قوی برای رقابت را نشان می‌دهد. NBSVM توسط Sida Wang و Chris Manning در مقاله‌ی:

Baselines and Bigrams: Simple, Good Sentiment and Topic Classification

معرفی شد. در این گزارش ما به جای SVM از رگرسیون لجستیک [۲] در کتاب‌خانه‌ی sklearn بهره می‌بریم–از آنجایی که عملا یکسان هستند–.

[۱] Naive Bayes - Support Vector Machine
[۲] logistic regression

۳ راه حل



ما در این پیاده‌سازی از NB-SVM برای مدل‌سازی استفاده خواهیم کرد.

Bag of Words

در این رویکرد، ابتدا مجموعه‌داده را که شامل داکیومنت‌ها و لیبل مربوطه است به‌نمایش bag of words در می‌آوریم؛ داده‌ساختار مورد استفاده یک ماتریس اسپارس می‌باشد که Term-document matrix نامیده می‌شود. متن اولین نظر و لیبل آن را در زیر مشاهده می‌کنید:

```
trn[0]
```

```
" A formal orchestra audience is turned into an insane, violent mob
by the crazy chantings of it's singers. Unfortunately it stays absurd
the WHOLE time with no general narrative eventually making it just
too off putting. Even those from the era should be turned off. The
cryptic dialogue would make Shakespeare seem easy to a third grader.
On a technical level it's better than you might think with some good
cinematography by future great Vilmos Zsigmond. Future stars Sally
Kirkland and Frederic Forrest can be seen briefly."
```

```
trn_y[0]
```

```
0
```

در ادامه، CountVectorizer مجموعه‌ی نظرات را به‌یک ماتریس که ستون‌های آن نشان دهنده‌ی توکن‌ها و مقادیر

نشان دهنده‌ی تعداد تکرار آن توکن است در می‌آورد. دستور fit_transform(trn) مجموعه‌داده‌ی آموزشی را به‌ماتریس term-document تبدیل می‌کند.

تعریف بیز ساده

ما در اینجا log-count ratio را برای هر کلمه تعریف می‌کنیم.

$$r = \log \frac{\text{ratio of feature } f \text{ in positive documents}}{\text{ratio of feature } f \text{ in negative documents}}$$

در فرمول بالا، **ratio of feature in positive documents** برابر است با: تعداد دفعاتی که یک داکیومنت با لیبل مثبت (یک) شامل کلمه می‌شود، بخش بر تعداد داکیومت‌هایی که لیبل مثبت دارند. نکته‌ی قابل توجه در اینجا، بهتر عمل کردن بیز ساده‌ی دودویی نسبت به بیز ساده است.

رگرسیون لجستیک

در علوم داده، رگرسیون لجستیک به‌دلیل واقع‌گرایانه بودن بهتر از نسبت به‌دست آمده در مرحله‌ی قبل عمل می‌کند. در زیر، پیاده‌سازی و پیشبینی با مدل رگرسیون لجستیک را مشاهده می‌کند. دقت پیش‌بینی برای نسخه‌های regularize نشده و regularize شده را در زیر مشاهده می‌کنید.

Unregularized version

```
m = LogisticRegression(C=1e8, dual=True)
m.fit(x, y)
preds = m.predict(val_term_doc)
(preds==val_y).mean()
```

```
0.85504000000000002
```

```
m = LogisticRegression(C=1e8, dual=True)
m.fit(trn_term_doc.sign(), y)
preds = m.predict(val_term_doc.sign())
(preds==val_y).mean()
```

```
0.85487999999999997
```

Regularized version

```
m = LogisticRegression(C=0.1, dual=True)
m.fit(x, y)
preds = m.predict(val_term_doc)
(preds==val_y).mean()
```

```
0.88275999999999999
```

```
m = LogisticRegression(C=0.1, dual=True)
m.fit(trn_term_doc.sign(), y)
preds = m.predict(val_term_doc.sign())
(preds==val_y).mean()
```

```
0.88404000000000005
```

Adding bigrams and trigrams

مدل بعدی ما یکی از نسخه‌های رگرسیون لجستیک با ویژگی‌های بیز ساده می‌باشد. حال برای هر داکیومنت ویژگی‌های باینری را-همانگونه که در مراحل قبلی توضیح دادیم-به‌دست می‌آوریم، با این تفاوت که این بار از bigram-ها و trigram-ها نیز بهره می‌بریم. هر ویژگی نسبت log-count می‌باشد.
پیش‌بینی وقتی ویژگی‌ها trigram-ها هستند:

```
m = LogisticRegression(C=0.1, dual=True)
m.fit(x, y);
```

```
preds = m.predict(val_x)
(preds.T==val_y).mean()
```

```
0.90500000000000003
```

```
np.exp(r)
```

```
matrix([[ 0.94678,  0.85129,  0.78049, ...,  3.     ,  0.5    ,  0.5
]])
```

پیش‌بینی وقتی ویژگی‌ها نسبت‌های log-count هستند:

```
x_nb = x.multiply(r)
m = LogisticRegression(dual=True, C=0.1)
m.fit(x_nb, y);
```

```
val_x_nb = val_x.multiply(r)
preds = m.predict(val_x_nb)
(preds.T==val_y).mean()
```

0.91768000000000005

مشاهده می‌کنید که دقت پیش‌بینی با این مدل نهایی 0.917680 می‌باشد. ما در پیاده سازی از NB-SVM-ای که توسط fast.ai ارائه شده استفاده خواهیم کرد، دقت این NB-SVM ، برای همین نمونه‌یی که بررسی شد، به 0.92129 می‌رسد.

۴ پیاده‌سازی

بارگذاری پکیج‌های مورد نیاز

```
import pandas as pd, numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer,
    TfidfVectorizer
```

خواندن داده

```
train = pd.read_csv('../input/train.csv')
test = pd.read_csv('../input/test.csv')
subm = pd.read_csv('../input/sample_submission.csv')

```

```
train.head()

```

Out[3]:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

دو نمونه از نظرهای این مجموعه‌داده که اولی سمی و دومی بدون لیبل است را در زیر مشاهده می‌کنید:

```
train['comment_text'][0]
```

Out[4]:
```
"Explanation\nWhy the edits made under my username Hardcore Metallica Fan were reverted? They weren't vandalisms, just closure on some GAs after I voted at New York Dolls FAC. And please don't remove the template from the talk page since I'm retired now.89.205.38.27"
```

```
train['comment_text'][2]
```

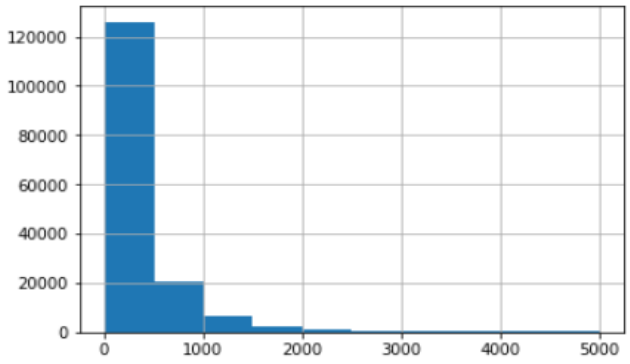
Out[5]:
```
"Hey man, I'm really not trying to edit war. It's just that this guy is constantly removing relevant information and talking to me through edits instead of my talk page. He seems to care more about the formatting than the actual info."
```

طول کامنت‌ها با هم بسیار متفاوت هستند:

```
lens = train.comment_text.str.len()
lens.mean(), lens.std(), lens.max()
```

Out[6]:
```
(394.0732213246768, 590.7202819048923, 5000)
```

```
lens.hist()
```



ما برای پیش‌بینی یک لیست از لیبل‌ها خواهیم ساخت، همچنین یک لیبل 'none' به داده اضافه می‌کنیم تا بدانیم چند نظر هیچ لیبلی نخورده‌اند. سپس می‌توانیم مجموعه‌داده را خلاصه‌سازی کنیم.

```
label_cols = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', '
    identity_hate']
train['none'] = 1-train[label_cols].max(axis=1)
train.describe()

```

Out[8]:

	toxic	severe_toxic	obscene	threat	insult	identity_hate	
count	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.00
mean	0.095844	0.009996	0.052948	0.002996	0.049364	0.008805	0.898321
std	0.294379	0.099477	0.223931	0.054650	0.216627	0.093420	0.302226
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
len(train),len(test)

```

Out[9]:

```
(159571, 153164)
```

تعدادی نظر خالی در مجموعه‌داده وجود دارد که باید حذف شود؛ در غیر این صورت، sklearn خطا خواهد داد.

```
COMMENT = 'comment_text'
train[COMMENT].fillna("unknown", inplace=True)
test[COMMENT].fillna("unknown", inplace=True)

```

Out[9]:

```
(159571, 153164)
```

ساختن مدل

کار را با پدیدآوردن یک نمایش bag of words، به‌عنوان term document matrix آغاز می‌کنیم. همانگونه که در مقاله‌ی NBSVM پیشنهاد شده، از ngrams استفاده خواهیم کرد.

```
import re, string
re_tok = re.compile(f'([{string.punctuation""}£‘’¨«»®´·º½¾¿¡§£])')
def tokenize(s):
  return re_tok.sub(r' \1 ', s).split()

```

Out[9]:
```
(159571, 153164)
```

ظاهرا استفاده از TF-IDF، prior بهتری نسبت به ویژگی‌های دودویی استفاده شده در مقاله نتیجه می‌دهد. در این چالش، امتیاز در جدول با استفاده از TF-IDF، از 0.59 به 0.55 کاهش می‌یابد.

```
n = train.shape[0]
vec = TfidfVectorizer(ngram_range=(1,2), tokenizer=tokenize,
min_df=3, max_df=0.9, strip_accents='unicode', use_idf=1,
smooth_idf=1, sublinear_tf=1 )
trn_term_doc = vec.fit_transform(train[COMMENT])
test_term_doc = vec.transform(test[COMMENT])

```

Out[9]:
```
(159571, 153164)
```

این یک ماتریس اسپارس با تعداد کمی از مقادیر غیر صفر می‌سازد-ذخیره شده در نمایش زیر-.

```
trn_term_doc, test_term_doc

```

Out[13]:

```
(<159571x426005 sparse matrix of type '<class 'numpy.float64'>'
        with 17775104 stored elements in Compressed Sparse Row format>,
 <153164x426005 sparse matrix of type '<class 'numpy.float64'>'
        with 14765755 stored elements in Compressed Sparse Row format>)
```

تابعی برای محاسبه‌ی پیش‌آمدها توسط فرمول بیز:

```
def pr(y_i, y):
  p = x[y==y_i].sum(0)
  return (p+1) / ((y==y_i).sum()+1)

```

```
x = trn_term_doc
test_x = test_term_doc

```

تابعی برای فیت کردن مدل‌ها-برای هر لیبل یک مدل فیت می‌کنیم-.

```
def get_mdl(y):
  y = y.values
  r = np.log(pr(1,y) / pr(0,y))
  m = LogisticRegression(C=4, dual=True)
  x_nb = x.multiply(r)
  return m.fit(x_nb, y), r

```

فیت کردن مدل‌ها و اضافه کردن پیش‌بینی برای هر سطر به preds:

```
preds = np.zeros((len(test), len(label_cols)))

for i, j in enumerate(label_cols):
  print('fit', j)
  m,r = get_mdl(train[j])
  preds[:,i] = m.predict_proba(test_x.multiply(r))[:,1]

```

```
fit toxic
fit severe_toxic
fit obscene
fit threat
fit insult
fit identity_hate
```

References

[1] Naive Bayes(NB)-Support Vector Machine(SVM): Art Of State Result Hands-On Guide using Fast.ai

[2] NB-SVM strong linear baseline

[3] NER - training using spacy (Ensemble)

[4] Baselines and Bigrams: Simple, Good Sentiment and Topic Classification (Also saved in the ZIP file!)