بسم الله الرحمن الرحیم

پروژه ۵ درس هوش مصنوعی

دکتر فدایی و دکتر یعقوب زاده

مهدی وجهی

۸۱۰۱۰۱۵۵۸

آماده کردن داده ها

lowercase

دلیل اصلی این موضوع این است که ما باید تا جایی که می توانیم عبارات مشابه و یکسان را تشخیص دهیم و برای این که بتوانیم یکسان بودن لغت را به مدل بفهمانیم این موضوع به خصوص در متون شبکه های اجتماعی که ممکن است شیوه نگارشی به درستی رعایت نشود اهمیتی دوچندان پیدا می کند. در نهایت می توان مزایا و معایب این کار را به صورت زیر بیان کرد:

مزایا:

- یکسان سازی و حذف تنوع نوشتاری
- یادگیری با داده های کمتر
- ساده سازی مدل و افزایش سرعت و کاهش میزان اورفیت
- حذف اطلاعات غیر مرتبط

معایب:

- از دست رفتن اطلاعات
- ایجاد ابهام در بعضی قسمت های متن
- عدم تشخیص اسامی خاص

حذف اعداد

با توجه به هدف این پروژه احتمالا اعداد موجود در متن کاربردی نیستند و صرفا باعث افزایش حجم و کاهش سرعت پردازش می شوند بنابراین آنها را حذف می کنیم.

مزایا:

- کاهش پیچیدگی مدل
- تمرکز روی محتوا متن
- حذف اطلاعات غیر مرتبط

معایب:

- از دست رفتن اطلاعات
- عدم درک درست مدل از جمله
- اهمیت اعداد در بعضی پروژه ها

هشتگ ها

هشتگ ها در واقع فضای کلی و موضوع کلی متن را مشخص می کنند. بنابراین برای درک متن توجه به آنها از اهمیت بالایی برخوردار است و مدل با ضریب دادن به آنها می تواند دقت خود را افزایش دهد.

بررسی نتایج پیش پردازش

```
'my life is meaningless i just want to end my life so badly my life is
completely empty and i dont want to have to create meaning in it
creating meaning is pain how long will i hold back the urge to run my
car head first into the next person coming the opposite way when will i
stop feeling jealous of tragic characters like gomer pile for the swift
end they were able to bring to their lives'
```

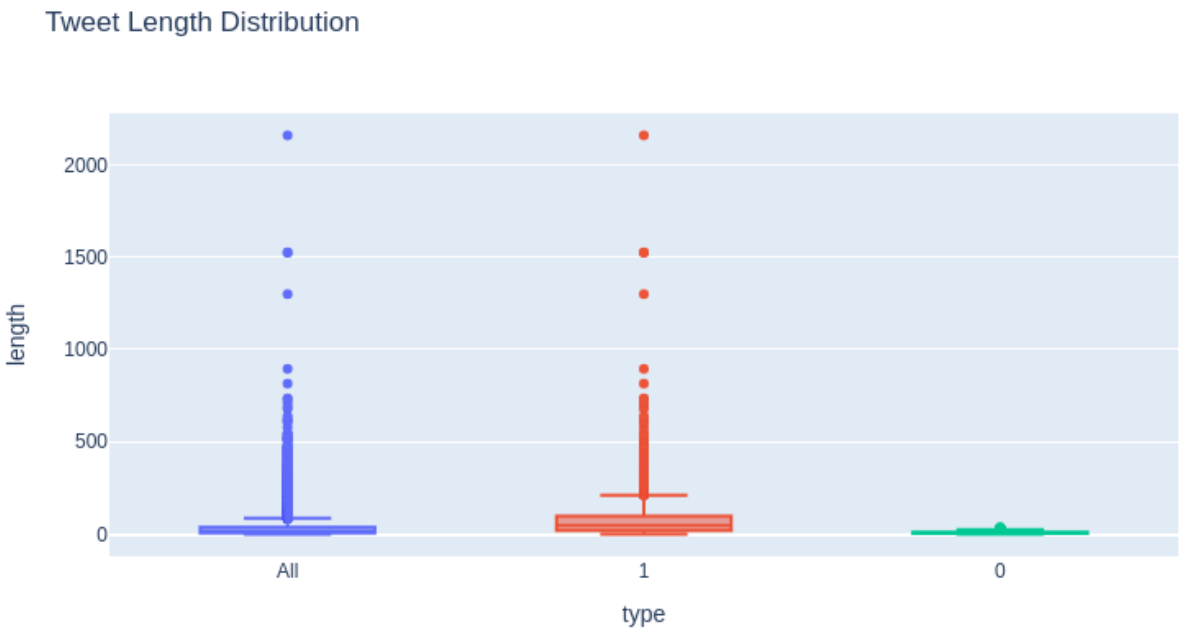
```
'life meaningless want end life badly life completely empty dont want
create meaning creating meaning pain long hold back urge run car head
first next person coming opposite way stop feeling jealous tragic
character like gomer pile swift end able bring life'
```

در این متن نکته قابل توجه حذف لغات ایست و لمتایز شدن کلمات است.

```
'muttering i wanna die to myself daily for a few months now i feel
worthless shes my soulmate i cant live in this horrible world without
her i am so lonely i wish i could just turn off the part of my brain
that feels '
```

```
'muttering wan na die daily month feel worthless shes soulmate cant live
horrible world without lonely wish could turn part brain feel'
```

این متن نیز مانند متن قبل است و نکته ی جدیدی ندارد.



برای مشاهده چارک ها به فایل کد مراجعه کنید.

ساخت بردار ویژگی

کلمات ناموجود در دیکشنری

هر چقدر هم که لغات دیکشنری زیاد باشد ممکن است به لغتی برسیم که در آن موجود نیست. برای این موضوع می توان از روش های زیر برای مواجه با آن استفاده کرد:

- حذف کلمه و استفاده از لغات بعدی
- جایگزین کردن با کلمه هم معنی و مشابه
- استفاده از شبکه های عصبی
- جایگزینی با بردار صفر
- جایگزینی با بردار میانگین

CNN

بهینه سازی به روش adam

این روش در واقع ترکیبی از روش SGD و RMSprop است. در adam برخلاف گرادیان کاهشی یک ضریب یادگیری ثابت داریم که تغییر نمی کند همچنین این روش گرادیان اول و دوم به صورت همزمان استفاده می کند و برای هرکدام از آنها نیز یک نرخ فروپاشی تعریف می شود. البته که در مقاله این الگوریتم ایرادی وجود دارد و برخلاف ادعایی که می شد این روش اکستمم اصلی را ممکن است پیدا نکند.
این روش به نسبت SGD سرعت همگرایی بالاتری دارد و برای مسائلی با مقیاس بزرگ مناسب است و در آخر هم می توان گفت که هایپر پارامتر های آن تفسیر پذیری و بصری سازی خوبی دارند.

Cross entropy

این تابع هزینه برای توزیع های احتمالی بهتر عمل می کند و همچنین در مقابل داده های نویز مقاومت بیشتری دارد و همچنین تفسیر پذیری خوبی دارد.

تقسیم داده های آموزش و آزمون

معمولا این نسبت در پروژه ها ۸۰/۲۰ یا ۹۰/۱۰ است که در این پروژه با توجه به حجم مناسب داده ها ۸۰/۲۰ را انتخاب می کنیم.

کرنل

کرنل ها در واقع میزان دید مدل را نشان میدهد کرنل های بزرگ تر میزان دید بیشتری دارند و می توانند فیچر های کلی تری تولید کنند و از آن طرف کرنل های کوچک تر روی بخش های کوچک تری و کلمات کمتری از متن متمرکز هستند. کرنل های بزرگ تر بیشتر مستعد اورفیت شدن هستند. معمولا ابتدا در شبکه ها از کرنل های کوچک استفاده می کنند و سپس از کرنل های بزرگ تر را وارد می کنند.

استفاده از feed forward

ظاهرا استفاده از این روش باعث می شود که ما اطلاعات موجود را حفظ کنیم و مدل هایی پیچیده تر داشته باشیم. در واقع ما به جای حذف خروجی ها با وزن دهی به آنها و ترکیبشان می توانیم بازنمایی بهتری داشته باشیم. مثلا در پولینگ ما بخشی از داده خود را دور میریزیم ولی در این روش این مشکل را نداریم.

تاثیر اندازه پنجره‌ی متن در یادگیری مدل

افزایش اندازه پنجره‌ی متن

مزایا:

- **بازنمایی کاملتر و کلی تر از متن**
- **درک روابط طولانی تر**

معایب:

- **افزایش پیچیدگی مدل و محاسبات**
- **افزایش زمان پردازش**
- **اورفیت شدن**
- **تنظیمات سخت برای جلوگیری از اورفیت شدن**

بررسی نتایج

```
Recall:    0.912
F1:        0.914
Precision: 0.916
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      1052
           1       0.91      0.89      0.90       764

    accuracy                           0.92      1816
   macro avg       0.92      0.91      0.91      1816
weighted avg       0.92      0.92      0.92      1816
```

```
Recall:    0.911
F1:        0.914
Precision: 0.917
              precision    recall  f1-score   support

           0       0.91      0.95      0.93      1052
           1       0.92      0.88      0.90       764

    accuracy                           0.92      1816
   macro avg       0.92      0.91      0.91      1816
weighted avg       0.92      0.92      0.92      1816
```

تصویر سمت راست پنجره ۱۹۶ است. اما همانطور که می بینید بر خلاف انتظار تقریبا هیچ فرقی مشاهده نمی شود.

روش های منظم سازی

نتایج در حالت های مختلف به شرح زیر است:

```
def forward(self, x):
    x = torch.permute(x, (0, 2, 1))

    x1 = nn.functional.relu(self.bn1_1(self.conv1_1(x)))
    x2 = nn.functional.relu(self.bn1_2(self.conv1_2(x)))
    x3 = nn.functional.relu(self.bn1_3(self.conv1_3(x)))

    x1 = nn.functional.relu(self.conv2_1(x1))
    x2 = nn.functional.relu(self.conv2_2(x2))
    x3 = nn.functional.relu(self.conv2_3(x3))

    x1 = self.pool1_1(x1)
    x2 = self.pool1_2(x2)
    x3 = self.pool1_3(x3)

    x1 = self.pool2(x1)
    x2 = self.pool2(x2)
    x3 = self.pool2(x3)

    x = torch.cat((x1, x2, x3), dim=2)
    x = self.flatten(x)

    x = self.dropout1(x)
    x = nn.functional.relu(self.fc1(x))

    x = self.fc2(x)

    return x
```

```
Recall:    0.917
F1:        0.915
Precision: 0.915
              precision    recall  f1-score   support

           0       0.93      0.92      0.93      1052
           1       0.89      0.91      0.90       764

    accuracy                           0.92      1816
   macro avg       0.91      0.92      0.92      1816
weighted avg       0.92      0.92      0.92      1816
```

```
def forward(self, x):
    x = torch.permute(x, (0, 2, 1))

    x1 = nn.functional.relu(self.bn1_1(self.conv1_1(x)))
    x2 = nn.functional.relu(self.bn1_2(self.conv1_2(x)))
    x3 = nn.functional.relu(self.bn1_3(self.conv1_3(x)))

    x1 = nn.functional.relu(self.bn2_1(self.conv2_1(x1)))
    x2 = nn.functional.relu(self.bn2_2(self.conv2_2(x2)))
    x3 = nn.functional.relu(self.bn2_3(self.conv2_3(x3)))

    x1 = self.pool1_1(x1)
    x2 = self.pool1_2(x2)
    x3 = self.pool1_3(x3)

    x1 = self.pool2(x1)
    x2 = self.pool2(x2)
    x3 = self.pool2(x3)

    x = torch.cat((x1, x2, x3), dim=2)
    x = self.flatten(x)

    x = self.dropout1(x)
    x = nn.functional.relu(self.fc1(x))

    x = self.fc2(x)

    return x
```

```
Recall:    0.906
F1:        0.902
Precision: 0.899
              precision    recall  f1-score   support

           0       0.94      0.89      0.91      1052
           1       0.86      0.93      0.89       764

    accuracy                           0.90      1816
   macro avg       0.90      0.91      0.90      1816
weighted avg       0.91      0.90      0.90      1816
```

```
def forward(self, x):
    x = torch.permute(x, (0, 2, 1))

    x1 = nn.functional.relu(self.conv1_1(x))
    x2 = nn.functional.relu(self.conv1_2(x))
    x3 = nn.functional.relu(self.conv1_3(x))

    x1 = nn.functional.relu(self.conv2_1(x1))
    x2 = nn.functional.relu(self.conv2_2(x2))
    x3 = nn.functional.relu(self.conv2_3(x3))

    x1 = self.pool1_1(x1)
    x2 = self.pool1_2(x2)
    x3 = self.pool1_3(x3)

    x1 = self.pool2(x1)
    x2 = self.pool2(x2)
    x3 = self.pool2(x3)

    x = torch.cat((x1, x2, x3), dim=2)
    x = self.flatten(x)

    x = self.dropout1(x)
    x = nn.functional.relu(self.fc1(x))

    x = self.fc2(x)

    return x
```

```
Recall:    0.902
F1:        0.900
Precision: 0.898
              precision    recall  f1-score   support

           0       0.93      0.90      0.91      1052
           1       0.87      0.90      0.89       764

    accuracy                           0.90      1816
   macro avg       0.90      0.90      0.90      1816
weighted avg       0.90      0.90      0.90      1816
```

```
def forward(self, x):
    x = torch.permute(x, (0, 2, 1))

    x1 = nn.functional.relu(self.bn1_1(self.conv1_1(x)))
    x2 = nn.functional.relu(self.bn1_2(self.conv1_2(x)))
    x3 = nn.functional.relu(self.bn1_3(self.conv1_3(x)))

    x1 = nn.functional.relu(self.conv2_1(x1))
    x2 = nn.functional.relu(self.conv2_2(x2))
    x3 = nn.functional.relu(self.conv2_3(x3))

    x1 = self.pool1_1(x1)
    x2 = self.pool1_2(x2)
    x3 = self.pool1_3(x3)

    x1 = self.pool2(x1)
    x2 = self.pool2(x2)
    x3 = self.pool2(x3)

    x = torch.cat((x1, x2, x3), dim=2)
    x = self.flatten(x)

      x = self.dropout1(x)
    x = nn.functional.relu(self.fc1(x))

    x = self.fc2(x)

    return x
```

```
Recall:    0.894
F1:        0.886
Precision: 0.885
              precision    recall  f1-score   support

           0       0.94      0.86      0.90      1052
           1       0.82      0.93      0.87       764

    accuracy                           0.89      1816
   macro avg       0.88      0.89      0.89      1816
weighted avg       0.89      0.89      0.89      1816
```

همانطور که مشاهده می شود تقریبا تفاوتی نکرده شاید در حالت اول مقداری بهتر شده و در باقی حالات مقداری کاهش پیدا کرده.

منابع

- به زبان ساده – فرادرس - مجله — Adam الگوریتم بهینه سازی آدام
- Cross-Entropy Loss Function in Machine Learning: Enhancing Model Accuracy | DataCamp
- pytorch.org