COMPUTATIONAL INTELLIGENCE

FINAL PROJECT DOCUMENTATION

Ferdowsi University of Mashhad
Department of Computer Engineering

SPRING 2025

نام و نام خانوادگی	شماره دانشجویی
امیرحسین افشار	۴۰۱۲۲۶۲۱۹۶
علیرضا صفار	۴۰۱۱۲۶۲۲۸۱

پیاده سازی پروژه در این رپو گیتهاب قابل مشاهده می باشد.

- https://github.com/AlirezaSaffar/ecommerce-text-classifier

فاز صفرم: دیتا پروفایلینگ

در ابتدا و مانند هر پروژه ای که با یادگیری سر و کار دارد، دیتا پروفایلینگ را انجام دادیم که بتوانیم insight هایی در رابطه با دیتایی که بر روی آن کار میکنیم بدست بیاوریم.

۱. بررسی تعداد سطر های داده:

تعداد سطر های داده را بدست آوردیم که به شرح زیر است:

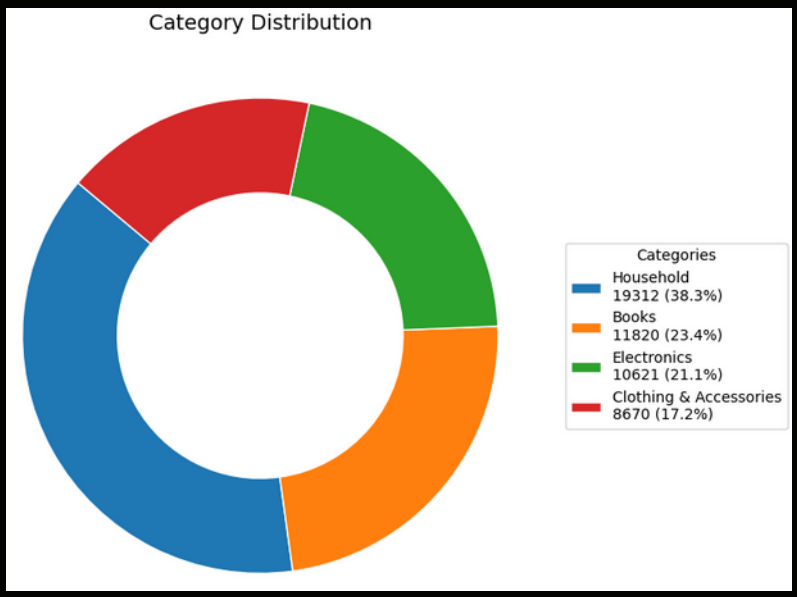
Property	Value
Shape	(۵۸۴۲۳، ۲)
Columns	’description’ ’category’

جدول ۱: اطلاعات کلی

۲. بررسی تعداد سطر های null و یا تکراری:

تعداد سطر های null برابر صفر بود، اما تعداد سطر های تکراری را برابر با مقدار تقریبی ۲۲k بدست آوردیم که تقریبا ۴۰ درصد دیتاست را تشکیل می داد. در فاز بعدی یعنی فاز اول: دیتا پریپروسسینگ، کل آنها را drop کردیم و فقط مقادیر unique را نگه داری کردیم. شایان ذکر است که در دیتاست اولیه، گاها حتی از یک دیتاپوینت بیش از ۳۰ بار تکرار داشتیم.

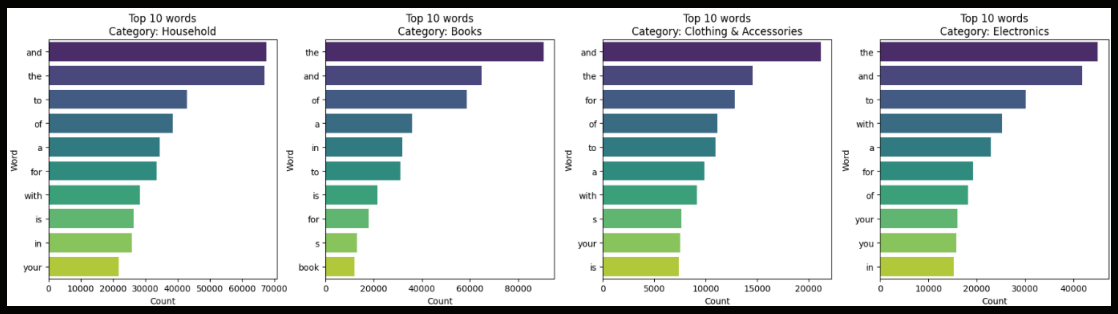
۳. بررسی تعداد کتگوری ها و میزان درصد هرکدام از آنها: همانطور که در شکل ۱ مشخص است، نزدیک به ۴۰ درصد داده ها را به تنهایی کتگوری household تشکیل داده اند و closing کمترین درصد را به خود اختصاص داده که نشان می دهد ممکن است در مرحله فاز اخر با بایاس شدن به سمت کتگوری ها رو به رو شویم. در این رابطه در بخش اخر بیشتر توضیح داده شده است.



شکل ۱: توزیع کتگوری ها

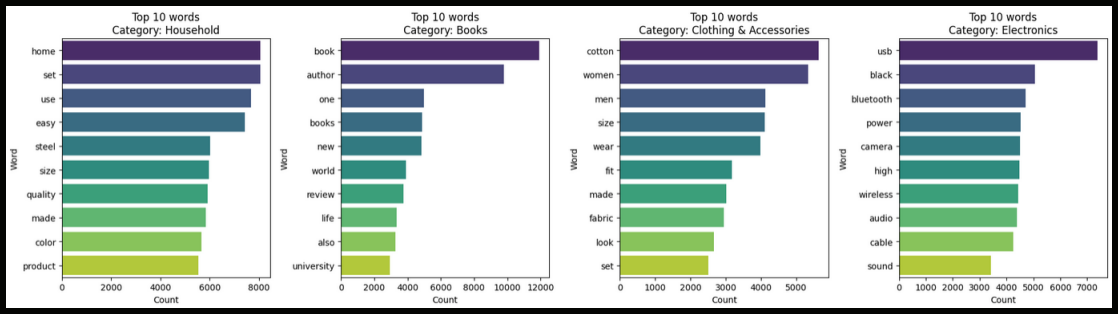
۴. بررسی کلمات پرتکرار هر کتگوری:

در ابتدا به شکل خام و سپس با اعمال حذف به شکل ساده این کار را انجام دادیم.



شکل ۲: نتیجه خام

همانطور که در شکل ۲ مشخص است نشان داده می شود که باید حذفیات کلمات غیرضروری اضافه صورت بگیرد تا بتوان به داده معناداری رسید.

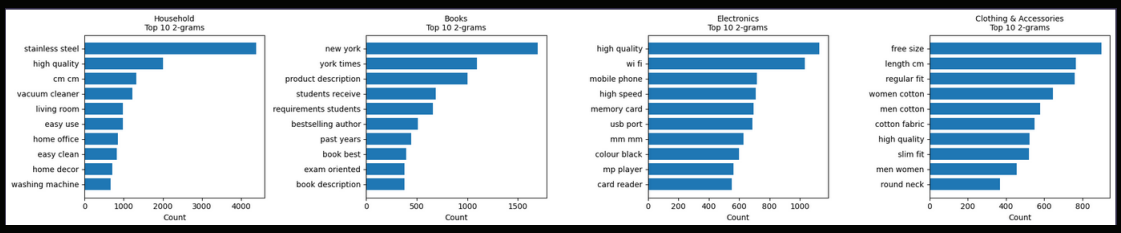


شکل ۳: نتیجه با اعمال حذف کلمات غیرضروری

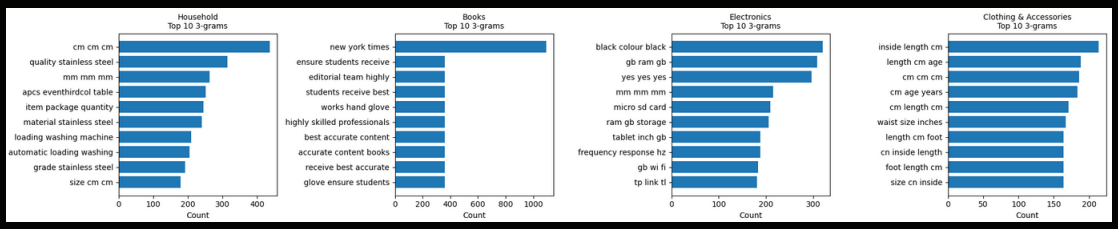
نتیجه شکل ۳ به طور کلی نشان می دهد که کلمات به خوبی در دامنه خود قابل تشخیص

هستند.

۵. بررسی n-gram ها به ازای ۲ و ۳:



شکل ۴: بررسی ۲گرم ها



شکل ۵: بررسی ۳گرم ها

شکل های ۴ و ۵ را بررسی کنید. با توجه به n-gram ها میتوان توجه ویژه ای به واحد ها و یونیت های اندازه گیری ای کرد. به این شکل که وقتی عبارتی نظیر:

a good quality table with 150 cm * 24 cm * 90 cm

رو به رو می شویم، پس از حذف ستاره و عدد ها با عبارت:

cm cm cm

رو به رو می شویم که به این صورت بیش از ۴۰۰ بار در دو کتگوری متفاوت رخ داده است و به طرز زیادی قرار است یادگیری بردارها را سخت کند. به این موضوع در پری پروسس توجه ویژه ای کردیم و عبارت بالا را کاملا درست کردیم و نرمالایز کردیم. توضیح بیشتر در بخش پری پروسس آمده است.

۱) فاز اول: پری پروسسینگ

در ابتدا سطر های تکراری را حذف کرده و سپس، برای پری پروسسینگ مراحل زیر را انجام دادیم و چنین پایپلاینی داشتیم:

Step	Function
0	Normalize units and remove singletons
1	Expand contractions
2	Convert to lowercase
3	Remove numbers
4	Remove punctuation
5	Remove special characters & emojis
6	Normalize whitespace
7	Tokenize text
8	Remove stopwords
9	Lemmatize tokens
10	Clean empty tokens

توضیحات بیشتر به شرح زیر است:

۱. Normalize-units-and-remove-singletons

همانطور که در بخش پروفایلینگ اشاره شد، واحدهای اندازه گیری مانند ،mhz gb، cm به شکل استاندارد تبدیل شدند و کاراکترهای تکراری حذف شدند و نرمالایز شدند.

۲. Expand-contractions

برای جملاتی که عموما به شکل not+verb خلاصه می شوند به کار بردیم.

۳. Convert-to-lowercase

برای این که همه کلمات یکنواخت باشند.

۴. Remove-numbers

از آنجا که اعداد نمی توانستند بردارهایی معنادار بسازند، همه اعداد را حذف کردیم

۵. Remove-punctuation

علائم نگارشی مانند کاما و نقطه برای تحلیل متن مفید نبودند و حذف شدند.

۶. Remove-special-characters-&-emojis

کاراکترهای خاص و ایموجی ها که معنای خاصی برای مدل نداشتند حذف شدند.

۷. Normalize-whitespace

فاصله های اضافی و تب ها به یک فاصله ساده تبدیل شدند.

۸. Tokenize-text

متن به کلمات جداگانه تقسیم شد تا قابل پردازش باشد.

۹. Remove-stopwords

کلمات رایج و بی معنی مانند ”the” و ”and” حذف شدند.

۱۰. Lemmatize-tokens

کلمات به شکل ریشه ای خود تبدیل شدند تا تنوع کاهش یابد.

۱۱. Clean-empty-tokens

توکن های خالی و بی معنی از نتیجه نهایی حذف شدند.

در نهایت یک مثال آورده می شود که اهمیت این پایپلاین دقیق تر نشان داده شود:
جمله ورودی:
SAF ’Floral’ Framed Painting (Wood, 30 inch x 10 inch, Special Effect UV Print Textured, SAO297) Painting made up in synthetic frame with UV textured print which gives multi effects and attracts towards it. This is an special series of paintings which makes your wall very beautiful and gives a royal touch (A perfect gift for your special ones).
و خروجی توکن های آن به این شکل در آمد:
[’saf’, ’floral’, ’frame’, ’paint’, ’wood’, ’numinch’, ’special’, ’effect’, ’uv’, ’print’, ’textured’, ’sao’, ’painting’, ’make’, ’synthetic’, ’frame’, ’uv’, ’textured’, ’print’, ’give’, ’multi’, ’effect’, ’attract’, ’towards’, ’special’, ’series’, ’painting’, ’make’, ’wall’, ’beautiful’, ’give’, ’royal’, ’touch’, ’perfect’, ’gift’, ’special’, ’one’]
مهم تر از همه توجهان را به بخش واحد های اندازه گیری جلب می کنیم که به جای
inch inch
به چنین توکن (بدون تکرار و یکبار آمده) تبدیل شده
numinch
و بنابراین کاملا هم ارتباط عدد و اندازه را حفظ می کند و هم اطلاعات با ارزشی را دور نمیریزد و هم نمایش بهتری را حاصل می شود.

۲) فاز دوم: تبدیل به بردار های عددی

برای محاسبه TF از تابع $compute\_tf$ را پیاده سازی کردیم. این تابع با شمارش تعداد وقوع هر کلمه در یک سند و سپس تقسیم آن تعداد بر مجموع کلمات موجود در همان سند، میزان تکرار هر کلمه در سند را به‌دست می‌آورد که اهمیت نسبی هر کلمه را در داخل همان سند اندازه‌گیری کنیم.
برای محاسبه IDF ، تابع $compute\_idf$ پیاده سازی کردیم که در این تابع، ابتدا تعداد اسنادی که هر کلمه در آن‌ها وجود دارد شمارش می‌شود. سپس با استفاده از فرمول

$$log(Ndf(t))\log\left(\frac{N}{df(t)}\right)log(df(t)N)$$

که در آن N تعداد کل اسناد و df(t) تعداد اسنادی است که کلمه t در آن‌ها ظاهر می‌شود، میزان IDF محاسبه می‌شود.
در نهایت، با ترکیب مقادیر TF و IDF تابع نهایی برای هر کلمه در هر سند مقدار TF-IDF آن را محاسبه کرده و نتیجه را در قالب یک لیست از دیکشنری‌ها ذخیره می‌کند. این لیست، شامل TF-IDF کلمات در گروه های مختلف است.
به منظور انتخاب بهترین classifier برای داده‌های خود، ابتدا به انجام هایپرپارامتر تیونینگ پرداختیم. از آنجا که تعداد مدل‌های مختلف و پارامترهای آن‌ها زیاد بود، تصمیم گرفتیم برای تسریع در روند تست مدل‌ها، از یک زیرمجموعه کوچک از داده‌ها استفاده کنیم.
ما برای انجام هایپرپارامتر تیونینگ، دو مدل SVM و Decision Tree را در نظر گرفتیم. برای هرکدام از این مدل‌ها مجموعه‌ای از کاندیدها (بیگ‌بندی‌ها) را امتحان کردیم:

۱. برای مدل SVM:

- با کرنل خطی (linear) و پارامتر C متفاوت (۱ و ۱۰).
- با کرنل rbf و پارامترهای مختلف C و gamma.

کاندیدهای مورد استفاده برای SVM به شکل زیر بود:

```
svm_configs = [
{'kernel': 'linear', 'C': 1},
{'kernel': 'linear', 'C': 10},
{'kernel': 'rbf', 'C': 1, 'gamma': 0.1},
{'kernel': 'rbf', 'C': 10, 'gamma': 0.01},
```

۲. برای مدل Decision Tree:

- با معیار تصمیم‌گیری gini و entropy و پارامترهای max_depth و min_samples_split مختلف.

کاندیدهای مورد استفاده برای Decision Tree به این صورت بودند:

```
dt_configs = [
{'criterion': 'gini', 'max_depth': 5,
 'min_samples_split': 2},
{'criterion': 'gini', 'max_depth': 10,
 'min_samples_split': 5},
{'criterion': 'entropy', 'max_depth': 5,
 'min_samples_split': 2},
{'criterion': 'entropy', 'max_depth': 10,
 'min_samples_split': 5},
]
```

برای کاهش زمان محاسبات و افزایش سرعت، تصمیم گرفتیم هایپرپارامتر تیونینگ را بر روی یک زیرمجموعه کوچک از داده‌ها انجام دهیم. این زیرمجموعه تنها ۱۰% از داده‌های اصلی را شامل می‌شد که از طریق تابع train_test_split ایجاد کردیم.
پس از انجام هایپرپارامتر تیونینگ با استفاده از تابع hyperparametertuning_tfidf، بهترین کانفیگ انتخابی برای مدل SVM به دست آمد که شامل کرنل خطی (linear) و C برابر با ۱۰ بود:

```
best_config = {'kernel': 'linear', 'C': 10}
```

این انتخاب در نهایت به عنوان بهترین تنظیمات مدل استفاده شد. این کار به ما این امکان را داد که بهترین عملکرد ممکن را در مدت زمان کم و با استفاده از منابع پردازشی محدود به‌دست آوریم. سپس با این کانفیگ مدل خود را train کردیم که نتایج آن در انتها داک بررسی شده است.

انتخاب کانفیگ مناسب برای مدل Word2Vec

سپس به به آموزش مدل Word2Vec پرداختیم. برای آموزش این مدل، از کانفیگ های مختلف استفاده کردیم تا بهترین کانفیگ برای داده‌های خود را پیدا کنیم. به دلیل عدم امکان دسترسی مستقیم به دقت نهایی مدل، برای ارزیابی عملکرد مدل از روش‌های ارزیابی کلی استفاده کردیم که به کمک آن می‌توانستیم کیفیت مدل را با توجه به روابط معنایی میان کلمات بسنجیم.
دلیل استفاده از آزمون‌های کلی برای ارزیابی کیفیت مدل‌ها این بود که به طور مستقیم نمی‌توانستیم به دقت نهایی مدل دسترسی پیدا کنیم و بتوانیم توانایی مدل را در درک روابط معنایی بین کلمات مشابه و غیر مشابه بررسی کنیم؛ به عبارتی بررسی کردیم که مدل‌های مختلف چطور کلمات مشابه را به هم نزدیک کرده و کلمات غیر مشابه را از هم دور می‌کنند، بدون اینکه نیاز به ارزیابی مستقیم دقت مدل داشته باشیم.
ما برای ارزیابی عملکرد مدل‌های مختلف، چندین پیکربندی مختلف از جمله اندازه بردار (vector size)، پنجره (window)، تعداد کلمات حداقل (min_count) و نوع مدل (Skip-Gram یا CBOW) را بررسی کردیم. این پیکربندی‌ها به شکل زیر بودند:

```
configs = [
{"vector_size": 50, "window": 3,
 "min_count": 1, "sg": 0}, # CBOW with small
vector size

{"vector_size": 100, "window": 5,
 "min_count": 1, "sg": 1}, # Skip-Gram with
larger window

{"vector_size": 100, "window": 3,
 "min_count": 2, "sg": 0}, # CBOW with stricter
frequency filtering

{"vector_size": 200, "window": 5,
 "min_count": 1, "sg": 1}, # Skip-Gram with
larger vector size
]
```

ارزیابی مدل‌ها

برای ارزیابی کیفیت مدل‌ها از مواردی که بحث شد استفاده کردیم که شامل بررسی شباهت کلمات معنایی این آزمون‌ها بین کلمات مختلف می‌باشد.

جفت های مشابه و غیرمشابه

جدول ۲: جفت‌های مشابه

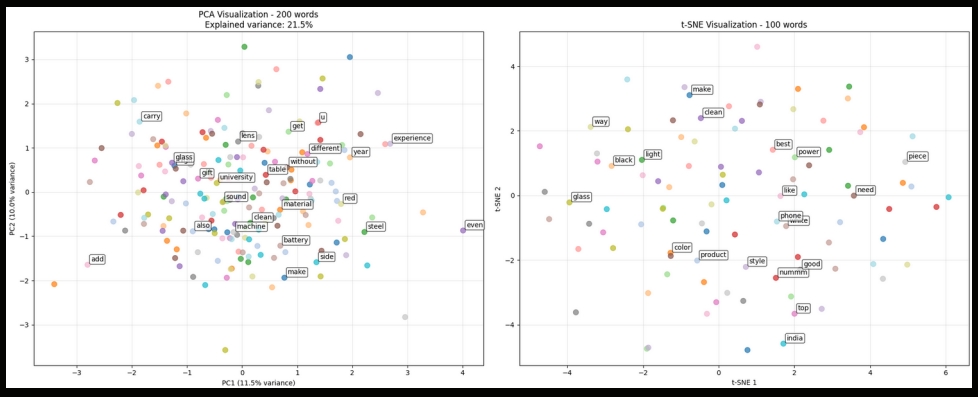
کلمه اول	کلمه دوم
paint	painting
frame	frames
gift	present
wood	timber
canvas	fabric
art	artwork

جدول ۳: جفت‌های غیر مشابه

کلمه اول	کلمه دوم
paint	table
frame	gift
art	kitchen
wall	water
home	camera
light	shoe

نحوه ارزیابی کانفیگ ها

برای هر کانفیگ، ابتدا مدل Word2Vec را با تنظیمات خاص خود آموزش دادیم و سپس شباهت کسینوسی بین کلمات مشابه و غیر مشابه را محاسبه کردیم. این شباهت‌ها به ما کمک کردند تا کیفیت مدل را ارزیابی کنیم. در نهایت، میانگین شباهت‌ها برای جفت‌های مشابه و غیر مشابه محاسبه شد.
با استفاده از نتایج به‌دست آمده کانفیگ
best_config = {”vector_size”: 200, ”window”: 5, ”min_count”: 1, ”sg”: 1}
را برگزیدیم و مدل را تریین کردیم.
نمودار زیر نیز برای مدل word۲vec برای بیان ارتباط میان لغات آورده شده است:



شکل واضح تر را در انتهای کد جوپیتر می توانید ببینید.

۳) فاز سوم: طبقه بندی و ویژوالیزیشن

پس از آموزش مدل Word2Vec و ارزیابی آن با استفاده از روش‌های مختلف، از دو روش متفاوت برای استفاده از این بردارها به منظور دستیابی به عملکرد استاندارد استفاده کردیم:

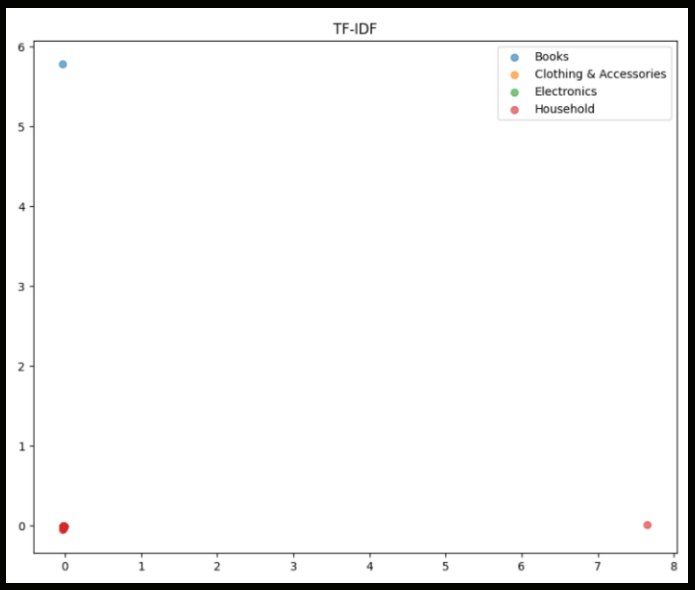
۱. روش اول: استفاده از میانگین بردارهای Word2Vec

در این روش، برای هر داک، میانگین بردارهای کلمات آن که توسط مدل Word2Vec تولید شده‌اند محاسبه شد. این روش به این صورت است که برای هر کلمه در سند، بردار مربوطه از مدل Word2Vec

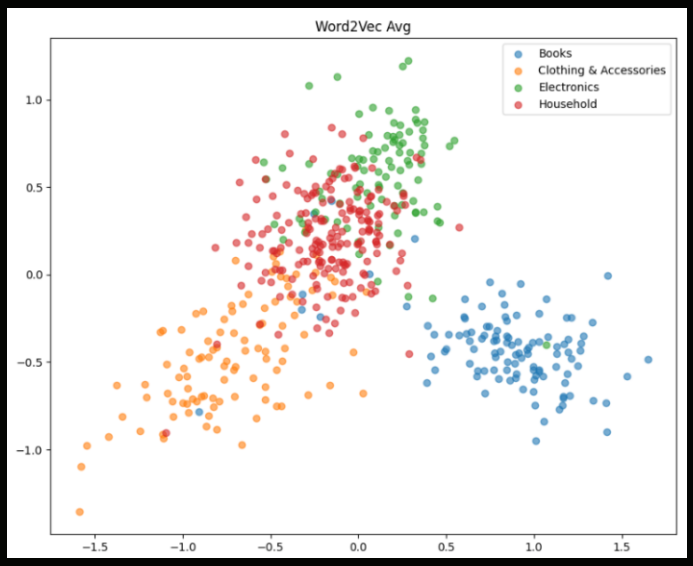
استخراج می شود و میانگین آن‌ها به عنوان نمایندهٔ سند در نظر گرفته می‌شود. پس از آن، از این ویژگی‌ها برای آموزش مدل دستهٔ‌بندی SVM استفاده شد.
تابع train_word2vecavg برای این کار پیاده‌سازی شده است که از document_vector_avg برای محاسبهٔ میانگین بردارهای کلمات هر سند استفاده می‌کند.

۲. روش دوم: استفاده از ترکیب TF-IDF و Word2Vec

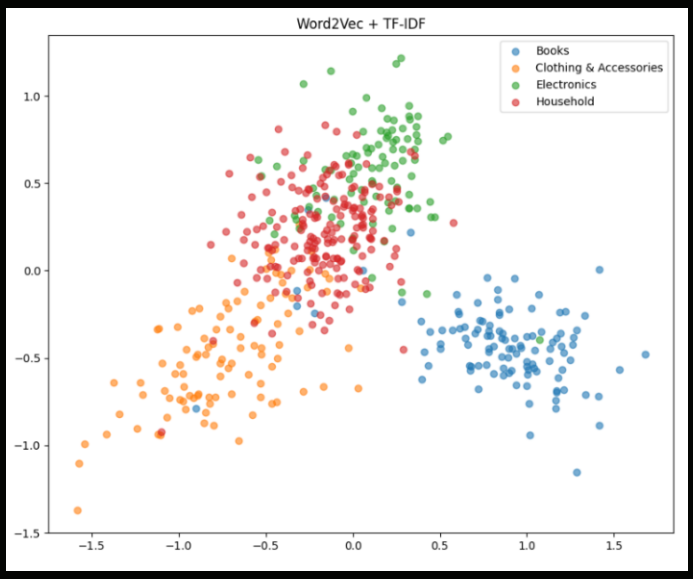
در این روش، ابتدا از ترکیب TF-IDF برای وزن‌دهی به کلمات هر سند استفاده کردیم و سپس این وزن‌ها را با بردارهای Word2Vec ترکیب کردیم. برای هر کلمه در سند، بردار Word2Vec از استخراج شده و با وزن مربوطه از TF-IDF ضرب می‌شود. سپس میانگین این بردارها به عنوان نمایندهٔ سند در نظر گرفته می‌شود. این روش به ما امکان می‌دهد که اهمیت کلمات مهم‌تر را با استفاده از وزن‌های TF-IDF در مدل Word2Vec لحاظ کنیم.
تابع train_word2vectfidf برای این روش پیاده‌سازی شده است. این تابع ابتدا مدل TF-IDF را روی داده‌های آموزشی آموزش داده و سپس از آن برای وزن‌دهی به بردارهای Word2Vec استفاده می‌کند.
در هر دو روش، پس از استخراج ویژگی‌ها، داده‌ها به مجموعه‌های آموزشی و از میانهٔ تقسیم شده و مدل دستهٔ‌بندی SVM با استفاده از داده‌های آموزشی آموزش داده شد. سپس، دقت مدل روی داده‌های آزمایشی ارزیابی گردید که این دقت ها در فاز چهارم داکیومنت بررسی شده اند.
در نهایت با کاهش ابعاد بردار ها به دو بعد با استفاده از pca، به چنین پلات هایی رسیدیم:



شکل ۶: tfidf



شکل ۷: avg word۲vec



شکل ۸: word۲vec+tfidf

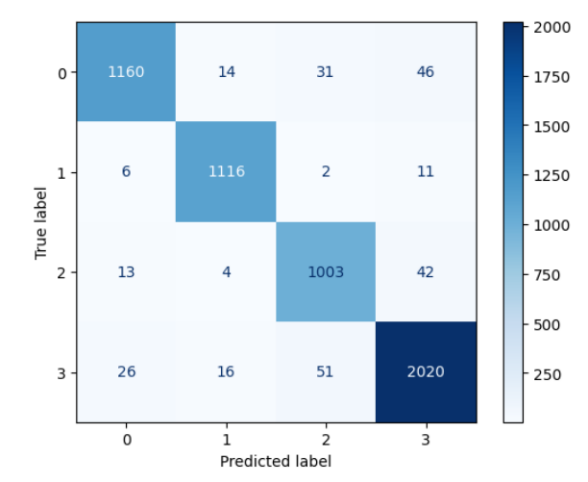
همانطور که مشخص است بعد از کاهش ابعاد به دو بعد در مدل های word۲vec نقاط پخش هستند در حالی که در مدل tfidf نقاط کلاس ها یکسان روی هم افتاده اند که این به دو دلیل است: ۱- تعداد ابعاد بردار ها در مدل ها word۲vec کمتر از مدل tfidf است.(در مورد مدل ها word۲vec این عدد ۲۰۰ است اما در مورد tfidf این عدد به اندازه تعداد کلمات است.) نتیجتا پس از کاهش ابعاد فاصله نقاط مدل ها word۲vec بیشتر خواهد بود. ۲- در بردارهای TF-IDF تفاوت‌ها بین سمپل ها کلاس‌ها متفاوت خیلی بیشتر است و تفاوت بین سمپل ها کلاس ها مشابه به مراتب کمتر است. اما در Word۲Vec چون ما از میانگین بردارهای کلمات استفاده کردیم، تفاوت بین سمپل ها کلاس‌ها مشابه کمتراست و تفاوت بین سمپل ها کلاس ها متفاوت کمتر است. در نتیجه پس از کاهش ابعاد فاصله نقاط مدل tfidf کمتر خواهد بود.

۴) فاز چهارم: ارزیابی عملکرد

برای هر سه مدل با استفاده از تابع evaluate_model معیارهای accuracy، precision، recall و confu-sion matrix را نمایش دادیم:

:Tfidf

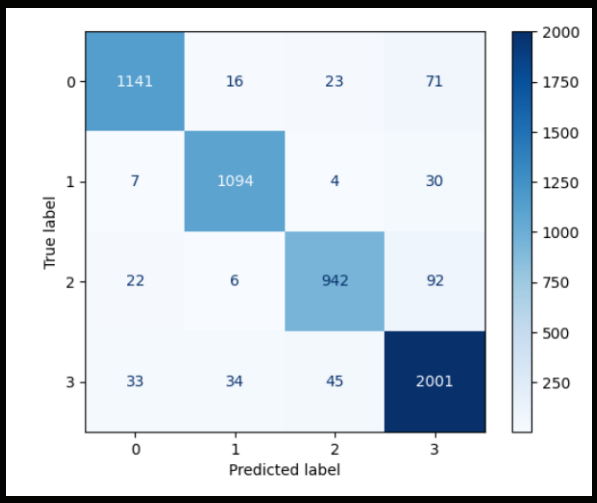
```
Accuracy: 0.9529
Precision: 0.9531
Recall: 0.9529
```



شکل ۹: ماتریس TFIDF

:Word2vecavg

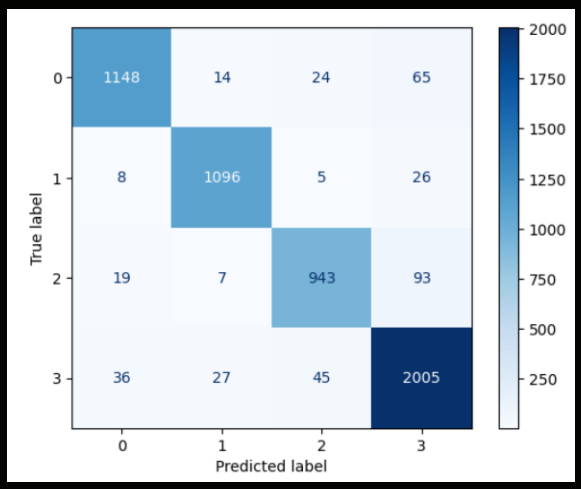
```
Accuracy: 0.9311
Precision: 0.9315
Recall: 0.9311
```



شکل ۱۰: ماتریس Word۲vecavg

:word2vectfidf

```
Accuracy: 0.9336
Precision: 0.9339
Recall: 0.9336
```



شکل ۱۱: ماتریس word۲vectfidf

۱. دقت TF-IDF بالاتر از Word2Vec با میانگین‌گیری:

- TF-IDF (0.9529) به طور قابل توجهی دقت بیشتری نسبت به Word2Vec با میانگین‌گیری (0.9311) دارد. این نشان می‌دهد که TF-IDF توانسته ویژگی‌های مهم‌تر هر سند را شناسایی کند و تاثیر مثبت‌تری در فرآیند دسته‌بندی داشته باشد.
- **دلایل:** TF-IDF قادر است کلمات مهم و متمایز کننده را با وزن بیشتر در نظر بگیرد و این موضوع کمک می‌کند که مدل در شبیه‌سازی و تفکیک دسته‌ها بهتر عمل کند. برخلاف Word2Vec که

صرفاً به روابط معنایی بین کلمات توجه دارد، TF-IDF مستقیماً بر اساس حضور کلمات در سند و تعداد دفعات آن‌ها در اسناد مختلف عمل می‌کند. این ویژگی باعث می‌شود که TF-IDF برای بسیاری از مسائل دسته‌بندی متنی به ویژه در داده‌های کوچک یا با پیچیدگی کم‌تر عملکرد بهتری داشته باشد.

۲. Word2Vec با میانگین‌گیری و کمک TF-IDF (0.9336):

- ترکیب Word2Vec با TF-IDF عملکرد بهتری نسبت به Word2Vec با میانگین‌گیری دارد (دقت ۰.۹۳۳۶ در مقابل ۰.۹۳۱۱). این نشان می‌دهد که اضافه کردن وزن‌های TF-IDF به بردارهای Word2Vec کمک می‌کند که مدل بتواند ویژگی‌های کلمات مهم‌تر را بهتر شبیه‌سازی کند.
- **دلیل:** استفاده از وزن‌های TF-IDF در ترکیب با Word2Vec می‌تواند به مدل کمک کند تا روی کلمات با وزن بیشتر (از نظر اطلاعاتی) تمرکز کند و در عین حال از توانایی‌های معنایی Word2Vec برای شبیه‌سازی روابط بین کلمات استفاده کند. با این حال، ترکیب این دو ویژگی در این مرحله نتایج خیلی بهینه نبوده و شاید نیاز به تنظیمات بیشتری داشته باشد تا این ترکیب به نتیجه مطلوب‌تر برسد.

دلایل بالاتر بودن عملکرد TF-IDF نسبت به Word2Vec:

۱. ویژگی‌های مستقیماً قابل تفسیر

TF-IDF یک روش مستقیماً قابل تفسیر است که بر اساس تعداد تکرار کلمات در هر سند و در کل مجموعه داده‌ها عمل می‌کند. این ویژگی می‌تواند مستقیماً برای الگوریتم‌های ماشین لرنینگ مثل SVM، که به ورودی‌های عددی و ساختار یافته نیاز دارند، مناسب باشند. در نتیجه، این ویژگی‌ها برای غیرمتخصصان و کاربران عملکرد و تفکیک دقیق‌تر دسته‌ها کار آمدتر هستند.

۲. مناسب بودن TF-IDF برای داده‌های کوچک و پیچیدگی کم

TF-IDF به دلیل سادگی و سرعت بالا در پردازش، برای داده‌های کوچک و با پیچیدگی کم بسیار مناسب است. این الگوریتم به طور مستقیم تعداد و توزیع کلمات را بررسی کرده و کلمات با ویژگی‌های مهم‌تر را در نظر می‌گیرد، که باعث می‌شود مدل سریع‌تر و با دقت بیشتری عمل کند، به‌ویژه در مجموعه‌ی داده‌هایی که اطلاعات معنایی پیچیده کمتری دارند.

۳. عدم وابستگی به داده‌های بزرگ و تعداد زیاد کلمات

یکی از چالش‌های Word2Vec این است که به مجموعه داده‌های بزرگی نیاز دارد تا بتواند روابط معنایی بین کلمات را به درستی مدل‌سازی کند. در حالی که TF-IDF از خود داده‌های کوچک‌تر برای تشخیص و دسته‌بندی استفاده می‌کند. بنابراین، زمانی که داده‌های آموزشی محدود باشد، TF-IDF می‌تواند عملکرد بهتر از Word2Vec داشته باشد.

۴. کمبود داده‌های آموزشی برای Word2Vec

مدل‌های Word2Vec برای آموزش بهتر نیاز به داده‌های بیشتری دارند تا بتوانند روابط معنایی بین کلمات را به درستی یاد بگیرند. وقتی داده‌ها کافی نیستند، این مدل ممکن است نتایج دقیق‌تری

نسبت به روش‌های ساده‌تری مانند TF-IDF نداشد. بنابراین، زمانی که مجموعه داده‌ها محدود باشد، TF-IDF می‌تواند عملکرد بهتری داشته باشد.