به نام خدا

هکاتون جوانه و دیوار

چالش سلامت پلتفرم با رویکرد الگوریتم‌های کلاسیک یادگیری ماشین

محمدحسین ملک‌پور

mohammadhossein.malekpour@gmail.com

مقدمه

در این چالش برای تحلیل و مدلسازی از محیط جوپیتر نوت‌بوک[1] استفاده شده. سعی کردم مراحل پیش پردازش داده تا جای ممکن بصورت توابع[2] نوشته شود. از آن جایی که هدف از این چالش طراحی یک پایپ‌لاین[3]ماژولار برای پیش بینی در محیط پروداکشن[4] نبوده روی این موضوع مانوری داده نشده و روی مهندسی ویژگی[5] و مدلسازی تمرکز کردم.

در تمام مراحل لازم از random_seed=0 استفاده شده تا بعد از هربار اجرای کامل کدها نتیجه یکسانی حاصل شود.

لازم به ذکر است نوت‌بوک برای خوانایی و فهم بهتر کامنت گذاری شده تا در گزارش به فرایند کلی حل این مسئله بپردازیم و درگیر جزئیات ریز کدها نشویم. توضیحات در گزارش به ترتیب فهرست داخل نوت‌بوک است:

- Setup Environment
- Load and Read Data
- Data Preprocessing and EDA
  - Extract Messages
  - Post Title WordCloud
  - Post Categories
  - Normalize Texts
  - Word Tokenize
  - Swear Extraction
  - Remove Stopwords
  - Stemming & Lemmatization
  - Check #Words Distribution
  - Choosing Initial features
  - Split Data
  - TF-IDF Vectorizer
  - Feature Selection with Chi-Squared
- Model Data
  - Machine Learning Algorithm Selection and Initialization
  - Tune Model with Hyper-Parameters
  - Final Model and Competition Submission

** برای اجرای نوت‌بوک ابتدا path دیتاست‌ها را در قسمت Load and Read Data مشخص کنید. همه سلول‌ها را به ترتیب اجرا کنید، به جز دو قسمت رنگی مشخص شده در فهرست! زیرا صرفا برای مدل سلکشن هستند و اجرای آن تقریبا 3 ساعت زمان می‌خواهد. فقط کافی‌ست آخرین بخش Model Data را اجرا کنید تا مدل نهایی کمتر از 8 ثانیه لرن شود و خروجی را پیش بینی کند.

---

[1] Jupyter Notebook
[2] Functions
[3] Pipeline
[4] Production
[5] Feature Engineering

Load and Read Data

برای راحتی پیش پردازش داده‌های یادگیری و داده‌های تست(مسابقه)، هر دو دیتاست[6] در یک لیست نگهداری می شوند که در هر مرحله از پیش پردازش، توابع همزمان روی هر دو دیتاست اعمال شود.

Data Preprocessing and EDA

ابتدا مشخصات مربوط به پست ها در ستون post_data که بصورت json هستند را استخراج میکنیم تا کنار سایر فیچرها قابل استفاده باشند.

سپس از ستون messages پیام های فرد آزاردهنده (annoying_person) و فرد مورد آزار قرار گرفته (annoyed_person) را بصورت مجزا در دو ستون استخراج می‌کنیم. چرا بصورت مجزا؟ چون معمولا در چت هایی که مزاحمت صورت می‌گیرد، فردی که مورد آزار قرار گرفته اگر پاسخ داد، توهین نمی‌کند! متن پیام‌های فرد آزاردهنده برای ما ارزش بیشتری دارد زیرا دقیقا چیزی هست که مدل می‌خواهد یاد بگیرد و هنگام مدلسازی وزن بیشتری به آن می‌دهیم تا دقت مدل در تشخیص مزاحمت بیشتر شود.

برای فهمیدن اینکه چه آگهی‌هایی بیشتر دارای مزاحمت هستند، نگاهی به ابرکلمات[7] عنوان[8] آگهی‌هایی که داری برچسب true هستند می‌کنیم:

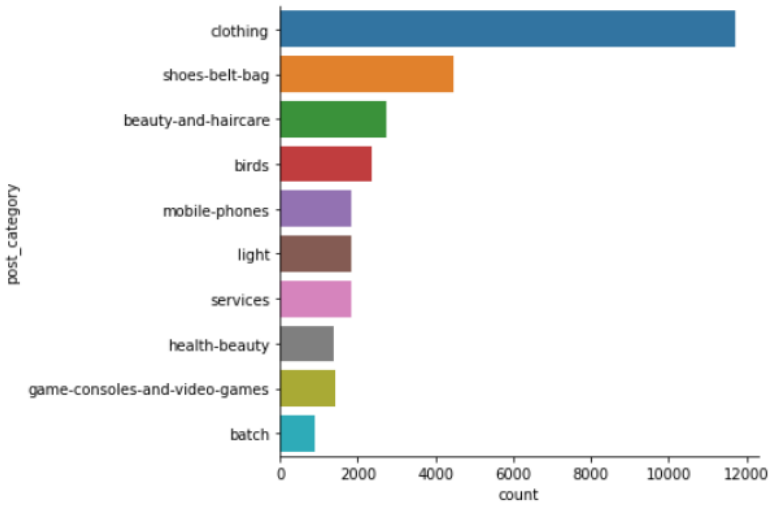
the most popular words of annoying people in the post title 1 Figure

چارت فوق علایق مزاحمان را نشان می‌دهد. همانطور که مشاهده می‌کنید ظاهرا شایعات شبکه‌های اجتماعی در خصوص آگهی‌های "کفش زنانه" در پلتفرم دیوار منجر به این شده تا مزاحمت‌های زیادی در آگهی‌های کفش زنانه داشته باشیم. همچنین

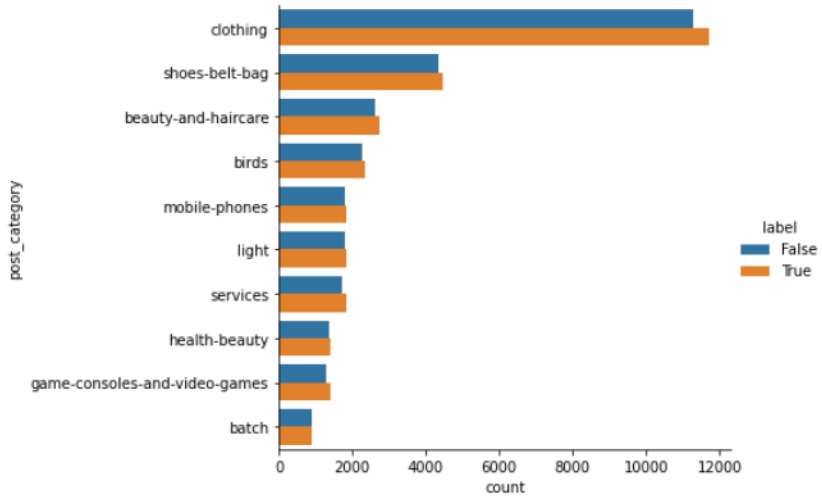
[6] Dataset
[7] Word Cloud
[8] Title

با توجه به کلمات روی نمودار عموما مزاحمت‌ها به نوعی برای خانم‌ها رخ می‌دهد! پس می‌توان نتیجه گرفت بین عنوان آگهی‌ها و مزاحمت‌ها رابطه وجود دارد و می توان از این فیچر متنی نیز در مدلسازی استفاده کرد.

می‌توان حدس زد که که نوع یا دسته بندی آگهی نیز با مورد مزاحمت قرار گرفتن، ارتباط خوبی دارد. به عنوان مثال احتمال اینکه در دسته‌بندی "تجهیزات صنعتی" مزاحمت رخ دهد به نسبت کمتر از دسته‌بندی "پوشاک زنانه" است. پس بیاید دسته بندی آگهی‌هایی که مزاحمت برای آن‌ها رخ داده را بررسی کنیم:



همانطور که مشاهده می‌کنید سه دسته‌بندی پوشاک، کفش و لوازم آرایشی بیشترین نرخ مزاحمت را دارند! اما با بررسی بیشتر، متوجه شدم توزیع دسته بندی آگهی‌های بدون مزاحم هم به همین شکل است(به اصطلاح imbalance است) پس الزاما از این دیتا نمی‌شود نتیجه مشخصی گرفت.



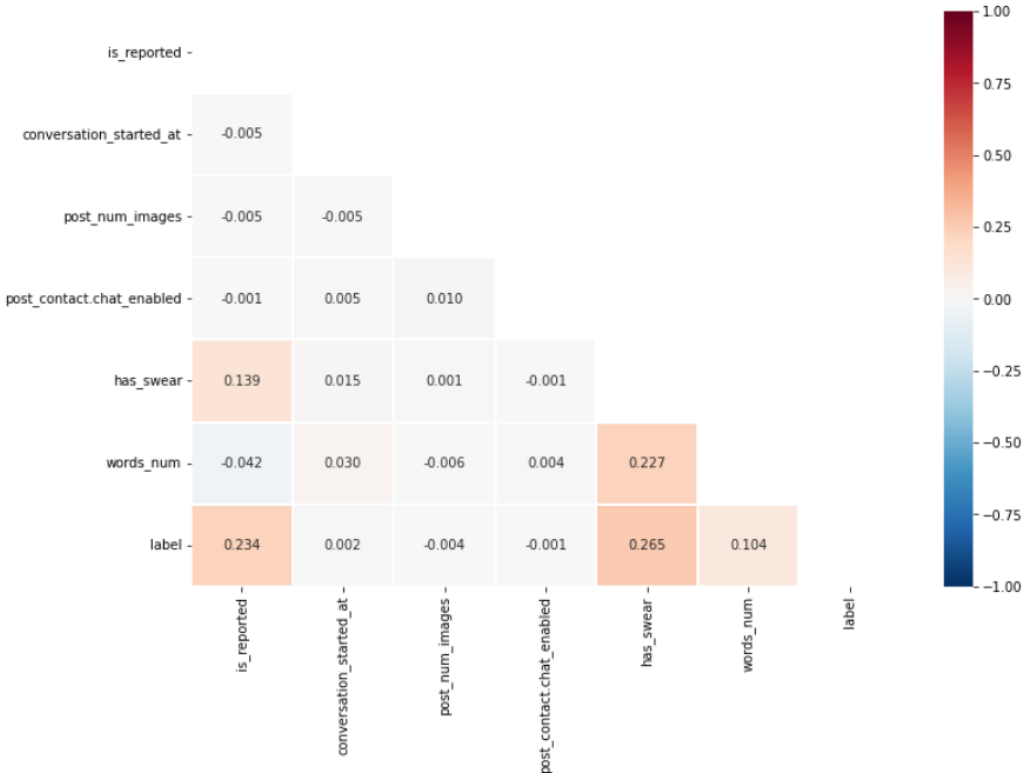
به همین دلیل از one-hot encodingی که برای post_category ایجاد کرده بودم، در مدلسازی بخاطر ایجاد بایاس نسبت به کتگوری های پرتکرار استفاده نکردم! (با این حال به اینکه دسته‌بندی آگهی در مزاحمت تاثیر مستقیم دارد شک نداریم)

برای نرمال سازی فیچرهای متنی حذف علائم نگارشی، صداهای کوتاه(فتحه، تشدید و ...)، حروف غیرفارسی را توسط سه تابع مختلف انجام می‌دهم، سپس با تبدیل هر متن به لیستی از لغات آن را به اصطلاح tokenize می‌کنم.

عموما در خیلی از مزاحمت‌ها و درگیری‌های لفظی مجازی از الفاظ رکیک استفاده می‌شود. با کمی جست و جو توانستم در اینجا لیستی از کلمات رکیک فارسی پیدا کنم. سپس در هر رکورد از دیتا بررسی کردم که آیا در پیام‌های فرد آزارگر از کلمه رکیکی استفاده شده یا خیر. متوجه شدم در 30٪ مزاحمت‌ها فرد مزاحم از حداقل یک کلمه رکیک استفاده کرده. پس ستون has_swaer را به عنوان یک فیچر جدید ذخیره کردم.

در ادامه‌ی پیش پردازش داده‌ها به حذف stopwordها و عملیات stemming می‌پردازیم.

برای بررسی فرضیه "افراد مزاحم بعد از چند پیام کوتاه ریپورت میشوند و افرادی که مزاحم نیستند وارد یک مکالمه طولانی‌تر می‌شوند"، تعداد کلماتی که فرد آزاردهنده در پیام‌هایش استفاده کرده را استخراج کردم تا در مرحله بعد کنار سایر فیچرها ببینم آیا با تارگت[9] کورلیشنی دارد یا خیر!



همانطور که مشاهده میکنید فیچر has_swear و is_reported با label کورلیشن مثبتی دارد (هرچند بسیار کم) و از این دو فیچر هم در مدلسازی استفاده خواهیم کرد.

[9] Target (label)

موقع بررسی داده‌ها متوجه شدم در 3363تا از رکوردهایی که label مزاحمت داشتند، فرد مزاحم هیچ پیامی متنی ارسال نکرده! حدس من این بود که این افراد پیام غیرمتنی ارسال کردند. به هر حال چون اطلاعات مفیدی نداشتند dropشان کردم.

در نهایت فیچرهای زیر برای ادامه کار انتخاب شدند:

```
fianl_features = ['annoying_person_messages', 'annoyed_person_messages', 'post_title', 'post_description', 'has_swear', 'is_reported']
```

حال در این مرحله دیتاست را به نسبت 90٪ و 10٪ split می‌کنیم (با توجه به تعداد رکوردها این نسبت بنظرم مناسب بود) و تا مرحله‌ی پایانی نیز از دیتای آزمایش[10] استفاده نمی‌کنم تا overfitting رخ ندهد.

برای بردارسازی[11] فیچرهای متنی از روش TF-IDF استفاده کردم. نکته‌ی مهم این است که 4 فیچر متنی داریم (پیام‌های فرد آزار دهنده، پیام‌های فرد آزار دیده، عنوان آگهی، توضیحات آگهی) ولی از آن جایی که هر کدام ارزش متفاوتی دارند آن‌ها را به یک دیگر append نکردم و هر کدام را بصورت جداگانه وکتورایز کردم تا در مرحله‌ی feature selection متناسب با ارزش هر کدام، تعداد فیچری(کلمات) که انتخاب میکنم متفاوت باشد.

برای feature selection از متود chi-2 (chi-squared) استفاده کردم. از آن جایی که ارزش هر کدام از 4 فیچر متنی که در بالا اشاره کردم متفاوت بود، پس در تعداد فیچر(لغت) انتخاب شده از هر یک بعد از وکتورایز شدن تفاوت قائل شدم . پیام های فرد آزار دهنده دارای بیشترین ارزش هستند زیرا ما می‌خواهیم مدلی بسازیم که دقیقا اینگونه پیام‌ها را شناسایی کند :) بعد از کمی آزمایش و خطا تعداد فیچرهای هر یک قبل و بعد از chi-2 به شرح زیر است:

Dimension before CHI-2:
X_train_vector_annoying_person_messages (89284, 111190)
X_train_vector_annoyed_person_messages (89284, 96880)
X_train_vector_post_title (89284, 16892)
X_train_vector_post_description (89284, 69817)

Dimension after CHI-2:
X_train_vector_annoying_best (89284, 20000)
X_train_vector_annoyed_best (89284, 10000)
X_train_vector_title_best (89284, 500)
X_train_vector_description_best (89284, 200)

حال 4 بردار بالا را علاوه بر 2 بردار has_swear و is_rported تجمیع[12] میکنیم تا دیتای آموزش نهایی بدست آید:

Final Dimension:
X_train_vector_best (89284, 30702)
X_test_vector_best (9921, 30702)

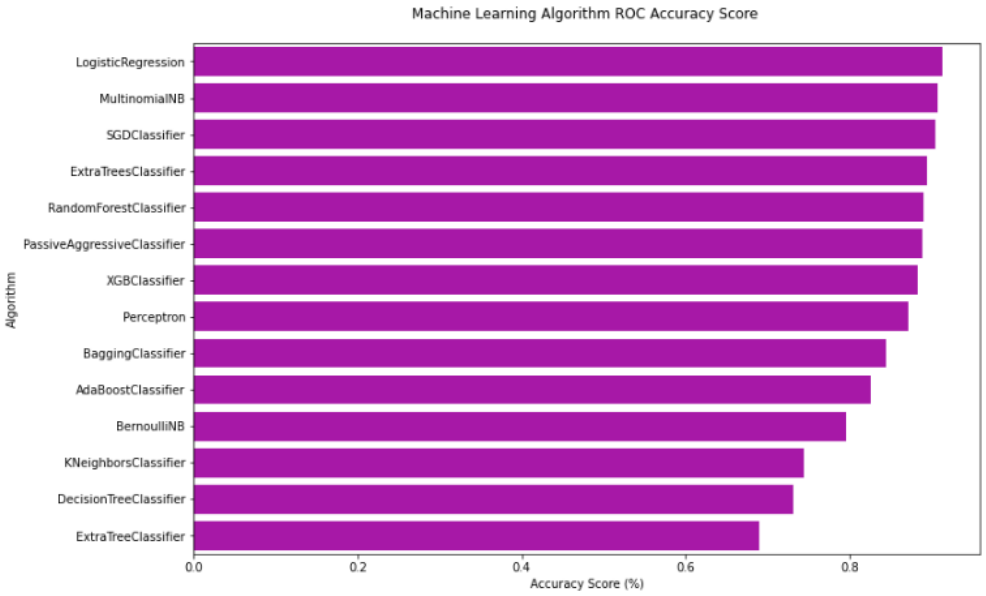
** برای وکتورایز و استخراج فیچر داده‌های آزمایش باستفاده از همان ابجکتی که به دیتای آموزش fit کردم داده‌ی آزمایش را transform کردم! در واقع لغت جدید از توی داده‌ی آزمایش توی فیچرها نخواهد بود. در نتیجه ارزیابی نهایی درست خواهد بود.

[10] Test
[11] Vectorize
[12] Stack

Model Data

برای انتخاب مدل بر اساس تجربه‌ی قبلی خودم لیستی از کلسیفایرها[13] ساختم و با استفاده از ShuffleSplit داده‌های آموزش را به 5 قسمت تقسیم کردم تا از متود cross_validate برای یادگیری و انتخاب مدل اولیه استفاده کنم. نوع امتیازدهی با توجه به معیار مسابقه ROC قرار داده شد. اجرای این قسمت تقریبا 2 ساعت زمان می‌برد. نتیجه بعد بر اساس میانگین دقت ROC روی پنج فولد[14]، به شکل است: (میانگین دقت دیتای آموزش و واریانس دقت دیتای آزمایش نیز مشخص است)

	MLA Name	MLA Parameters	MLA Train Accuracy Mean	MLA Test ROC Accuracy Mean	MLA Test Accuracy 3*STD	MLA Time
7	LogisticRegression	{'C': 1.0, 'class_weight': None, 'dual': False...	0.937108	0.912721	0.00262451	1.95441
9	MultinomialNB	{'alpha': 1.0, 'class_prior': None, 'fit_prior...	0.923974	0.906406	0.0077516	0.0481992
5	SGDClassifier	{'alpha': 0.0001, 'average': False, 'class_wei...	0.918745	0.904851	0.00384588	0.275797
3	ExtraTreesClassifier	{'bootstrap': False, 'ccp_alpha': 0.0, 'class_...	0.999914	0.893985	0.00681539	816.263
2	RandomForestClassifier	{'bootstrap': True, 'ccp_alpha': 0.0, 'class_w...	0.999658	0.890009	0.00886945	476.27
4	PassiveAggressiveClassifier	{'C': 1.0, 'average': False, 'class_weight': N...	0.968277	0.887852	0.0143718	1.12201
13	XGBClassifier	{'objective': 'binary:logistic', 'use_label_en...	0.921137	0.882983	0.0088656	25.7326
6	Perceptron	{'alpha': 0.0001, 'class_weight': None, 'early...	0.950479	0.871529	0.0221399	0.444197
1	BaggingClassifier	{'base_estimator': None, 'bootstrap': True, 'b...	0.998133	0.844656	0.00947547	738.903
0	AdaBoostClassifier	{'algorithm': 'SAMME.R', 'base_estimator': Non...	0.827987	0.825466	0.014059	78.4025
8	BernoulliNB	{'alpha': 1.0, 'binarize': 0.0, 'class_prior':...	0.810951	0.796024	0.02773	0.0713992
10	KNeighborsClassifier	{'algorithm': 'auto', 'leaf_size': 30, 'metric...	0.882671	0.744056	0.0229652	0.0378054
11	DecisionTreeClassifier	{'ccp_alpha': 0.0, 'class_weight': None, 'crit...	0.999914	0.731637	0.00817811	97.3886
12	ExtraTreeClassifier	{'ccp_alpha': 0.0, 'class_weight': None, 'crit...	0.999914	0.690221	0.0217813	8.0138



همانطور که ملاحظه می‌کنید LogisticRegression نسبتا عملکرد بهتری دارد.

[13] Classifier
[14] Fold

برای tune کردن hyperparameterها از روش grid search استفاده کردم. پارامترهای مدل و دقت آن را قبل و بعد از tune شدن در قسمت زیر مشاهده می‌کنید:

```
BEFORE Parameters:  {'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_rati
o': None, 'max_iter': 100, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': None, 'solver': 'lbfgs', 't
ol': 0.0001, 'verbose': 0, 'warm_start': False}
BEFORE train score mean: 93.71
AFTER test ROC score mean: 91.27
BEFORE test score 3*std: +/- 0.26
----------
AFTER Parameters:  {'C': 3, 'max_iter': 300, 'penalty': 'l2', 'solver': 'liblinear', 'tol': 1e-05}
AFTER train score mean: 95.28
AFTER test ROC score mean: 91.65
AFTER test score 3*std: +/- 0.21
----------
```

در بخش آخر، مدل نهایی با در کمتر از 8 ثانیه learn می‌شود و می‌توانیم با آن دیتاست مسابقه را پیش بینی می‌کنیم. چرا از همان مدل تیون شده قسمت قبل استفاده نکردم؟ چون اجرای کد قسمت قبل حدود یک ساعت زمان می‌برد. مدل نهایی با همان پارامترهای تعیین شده در قسمت قبل لرن می‌شود تا هیئت داوران هم بتوانند کد را سریع اجرا کنند.

عملکرد مدل نهایی:

	precision	recall	f1-score	support
False	0.82	0.86	0.84	4898
True	0.86	0.82	0.84	5023
accuracy			0.84	9921
macro avg	0.84	0.84	0.84	9921
weighted avg	0.84	0.84	0.84	9921

```
ROC Accuracy:  0.9128107073326317 %
```

جمع بندی

- با توجه به اینکه از روش های کلاسیک یادگیری ماشین استفاده شده، تنها بخاطر feature engineering مناسب، دقت تا 91٪ افزایش یافت. عمده تلاش من در حل این مسئله حول این قسمت بود.
- تمام ملاحظات لازم برای جلوگیری از overfitting و generalize بودن مدل صورت گرفته و احتمالا روی هر گونه دیتایی با توزیع مشابه دیتاست مسابقه، دقتی بین 89 تا 92 خواهد گرفت.
- از نقاط قوت این مدل می‌توان به این اشاره کرد که با 37502 فیچر، learning آن در کمتر از 8 ثانیه انجام می‌شود و در محیط پروداکشن در بازه‌های زمانی کوتاهی می‌توان آن را مجدد learn و آپدیت کرد.
- با وجود مدل‌های deep learning استفاده از مدل‌های کلاسیک مخصوصا در تسک‌های NLP تقریبا کمرنگ شده. من در این چالش سعی کردم با سواد و تجربه اندک خودم به بهترین نتیجه برسم.
- در نهایت ممنون بابت برگزاری این ایونت :)