بسم الله الرحمن الرحیم

گزارش پروژه صفر درس آمار و احتمالات
مهندسی پاییز ۱۴۰۲

مهدی وجهی
۸۱۰۱۰۱۵۵۸

فهرست

پیش پردازش 4
خواندن فایل csv 5
نرمالایز کردن متن 6
شکستن متن به کلمات 6
ریشه یابی کلمات (امتیازی) 8
حذف کلمات ایست (امتیازی) 9
پردازش 10
ساخت BoW 10
محاسبه احتمالات 12
محاسبه با لگاریتم 12
محاسبه p(c) 12
محاسبه p(x|c) 13
Additive Smoothing 13
محاسبه p(x) 14
محاسبه p(c|x) برای داده های آزمایشی 15
انتخاب پر احتمال ترین دسته بندی برای هر کتاب 16
محاسبه میزان دقت 16
نتایج 18
نتیجه گیری 20

مقدمه

در دنیای امروز، با افزایش حجم عظیم داده‌ها و نیاز به استفاده بهینه از آن‌ها، استفاده از روش و مدل های آمار و احتمال به عنوان یک ابزار قدرتمند در حوزه تحلیل و پیش‌بینی داده‌ها روز به روز بیشتر می‌شود. یکی از حوزه‌های کاربردی این روش‌ها، پیش‌بینی ویژگی‌های مرتبط با داده‌ها است که می‌تواند در بسیاری از صنایع و زمینه‌ها از جمله علم کتابخانه‌ای مورد استفاده قرار گیرد.

در این پروژه، هدف ما پیش‌بینی دسته بندی کتاب‌ها است. با استفاده از داده‌های آموزشی که شامل نام کتاب، دسته بندی، توضیحات مربوط به کتاب‌ها می‌شود که به وسیله ی آن مدلی را آموزش می‌دهیم تا بتواند الگوها و روابط موجود در توضیحات و ارتباط آن با دسته بندی را یاد بگیرد. سپس با استفاده از مدل آموزش دیده، می‌توانیم دسته بندی کتاب‌های جدید را پیش‌بینی کنیم. به عنوان مثال، با ورودی دادن توضیحات یک کتاب جدید به مدل، می‌توانیم دسته بندی آن را دریافت کنیم.

برای دستیابی به این هدف، در این پروژه از قانون بیز استفاده می‌شود. با استفاده از داده‌های آموزشی، برنامه آموزش داده می‌شود تا بتواند شباهت ها را در توضیحات تشخیص دهد و دسته بندی کتاب‌ها را پیش‌بینی کند. سپس با استفاده از داده‌های تست، عملکرد مدل ارزیابی می‌شود تا صحت و دقت پیش‌بینی‌های آن بررسی شود.

با پیاده‌سازی این پروژه، امکان استفاده بهینه از داده‌های کتابخانه‌ای و پیش‌بینی دسته بندی کتاب‌ها به صورت خودکار و دقیق‌تر امکان‌پذیر خواهد شد. این امر می‌تواند در بهبود عملکرد و خدمات ارائه شده در حوزه کتابخانه‌ها و صنعت نشر نقش موثری داشته باشد.

پیاده سازی

پیش پردازش

در این قسمت ما باید داده ها (آموزشی و تست) را برای بررسی آماده کنیم. در نهایت هدف رسیدن به خروجی ای به شکل زیر است:

```
[
    {
        "title": name of book (normalized),
        "description": description of book (tokenized & normalized& remove Word that do not contain letters),
        "categories" : categories of book (normalized)
    },
    .
    .
    .,
    {...}
]
```

برای این کار لازم است مراحل زیر طی شود:

- خواندن فایل های csv
- نرمالایز کردن متن
- شکستن متن به کلمات
- ریشه یابی کلمات (امتیازی)
- حذف کلمات ایست (امتیازی)

خواندن فایل csv

در ابتدا لازم است که ما داده های خود را از فایل به برنامه بیاوریم. این کار به آسانی و با استفاده از کتابخانه csv انجام می شود.

```
books_train_csv = list()

with open("books_train.csv") as file:
   reader = csv.DictReader(file)
   books_train_csv = [i for i in reader]

books_test_csv = list()

with open("books_test.csv") as file:
   reader = csv.DictReader(file)
   books_test_csv = [i for i in reader]
```

حال داده ها را به صورت زیر داریم:

```
[
    {
     "title": ...,
     "description": ...,
     "categories": ...
    },
    {...},
    ...
]
```

نرمالایز کردن متن

در این بخش با استفاده از کتابخانه هضم متون را نرمالایز کرده و نیم فاصله ها را با فاصله جایگزین می کنیم. این کار باعث پردازش بهتر متن می شود.

برای این کار روی کتاب ها پیمایش کرده و متون را نرمالایز می کنیم.

```
normalizer = Normalizer()

books_train_csv = [
                    {j : normalizer.normalize(i[j]).replace("\u200c", " ")
                          for j in i.keys()}
                    for i in books_train_csv
                  ]

books_test_csv = [
                    {j : normalizer.normalize(i[j]).replace("\u200c", " ")
                          for j in i.keys()}
                    for i in books_test_csv
                  ]
```

شکستن متن به کلمات

حال مجدد با استفاده از کتابخانه هضم ابتدا متن را به جملات و سپس به کلمات شکسته و برای مرحله بعدی آماده می کنیم. در این بخش ما **فقط کلمات حاوی حروف فارسی** را نگه می داریم و مابقی را دور میریزیم.

ابتدا روی کتاب ها پیمایش می کنیم و بخش توضیحات را به تابع مربوطه ارسال می کنیم.

```
for i in range(len(books_train_csv)):
   for j in {"description"}:
       books_train_csv[i][j] = tokenize_str(books_train_csv[i][j])

for i in range(len(books_test_csv)):
   for j in {"description"}:
       books_test_csv[i][j] = tokenize_str(books_test_csv[i][j])
```

سپس در تابع مربوطه متن را به جمله و جمله را به کلمه شکسته و بعد از بررسی شرایط کلمه که در ادامه ذکر می کنیم آنها را در لیستی ذخیره کرده.

```
def tokenize_str(txt : str) -> list(str()):
   token = list()
   for sent in sent_tokenize(txt):
       for word in word_tokenize(sent):
           if is_valid_word(word):
               token.append(word)
   return token
```

در تابع مربوط به شرایط کلمه بررسی می شود که کلمه شامل حروف فارسی باشد که این کار با استفاده از تابع پایتون و یونیکد حروف فارسی انجام می شود.

```
def is_valid_word(word : str) -> bool:
   if has_farsi_letters(word):
       return True
   return False

def has_farsi_letters(text : str) -> bool:
   for char in text:
           if char.isalpha() and ord(char) >= 0x0600 and ord(char) <=
0x06FF:
                   return True
   return False
```

حال داده های ما به شکل زیر شده:

```
[
    {
     "title": ...,
     "description": [word1, word2, ...],
     "categories": ...
    },
    {...},
    ...
]
```

ریشه یابی کلمات (امتیازی)

برای بررسی دقیق تر لازم است ما کلمات را به ریشه ی آن تبدیل کنیم که کلمات یکسان را اجماع کنیم.

برای این کار مجددا روی کتاب ها پیمایش کرده در کلمات را با استفاده از تابع مربوطه در کتابخانه هضم ریشه یابی می کنیم. لازم به ذکر است برای افزایش سرعت پردازش هر کلمه را یکبار پردازش می کنیم سپس ذخیره می کنیم که دفعه بعدی مجددا لازم به پردازش نباشد.

```
unique_words = dict()
lemmatizer = Lemmatizer()
for i in range(len(books_train_csv)):
   for j in {"description"}:
       for k in range(len(books_train_csv[i][j])):
           if books_train_csv[i][j][k] not in unique_words.keys():
               unique_words[books_train_csv[i][j][k]] =
lemmatizer.lemmatize(books_train_csv[i][j][k])
           books_train_csv[i][j][k] = unique_words[books_train_csv[i][j][k]]


for i in range(len(books_test_csv)):
   for j in {"description"}:
       for k in range(len(books_test_csv[i][j])):
           if books_test_csv[i][j][k] not in unique_words.keys():
               unique_words[books_test_csv[i][j][k]] =
lemmatizer.lemmatize(books_test_csv[i][j][k])
           books_test_csv[i][j][k] = unique_words[books_test_csv[i][j][k]]
```

حذف کلمات ایست (امتیازی)

برای این که نتایج برنامه دقت بالاتری داشته باشد می توان لغاتی که مربوط به دسته بندی خاصی نمی شود و داده خاصی به ما نمی دهند را حذف کرد. این کلمات، کلمات ایست نام دارند. در این برنامه از کلمات ایست کتابخانه هضم استفاده شده.

برای این کار روی کتاب ها پیمایش می کنیم و سپس در توضیحات هر کتاب را با نسخه جدید (که کلمات ایست حذف شده اند) جایگزین می کنیم.

```
stop_word = utils.stopwords_list()
for i in range(len(books_train_csv)):
   for j in {"description"}:
       books_train_csv[i][j] = [word for word in books_train_csv[i][j] if
word not in stop_word]

for i in range(len(books_test_csv)):
   for j in {"description"}:
       books_test_csv[i][j] = [word for word in books_test_csv[i][j] if word
not in stop_word]
```

پردازش

بعد از آماده سازی داده ها باید با استفاده از قانون بیز احتمالات را حساب کنیم. اما پیش از آن باید با داده های آموزشی خود bag of word تشکیل دهیم.

کار هایی که در پردازش باید انجام دهیم به شرح زیر است:

- ساخت BoW
- محاسبه احتمالات
  - محاسبه با استفاده از لگاریتم
  - محاسبه p(c)
  - محاسبه p(x|c)
    - Additive Smoothing
  - محاسبه p(x)
- محاسبه p(c|x) برای داده های آزمایشی
- انتخاب پر احتمال ترین دسته بندی برای هر کتاب
- محاسبه میزان دقت

ساخت BoW

برای این که BoW را تشکیل دهیم. ابتدا لازم است لغات هر دسته را بشماریم. برای این کار ابتدا فهرستی از تمام لغات تشکیل می دهیم که با پیمایش روی کتاب ها و اضافه کردن لغات آن در فهرست تشکیل شده از کل لغات انجام میشود.

```
all_words = list()
for i in books_train_csv:
   all_words.extend(i["description"])
```

سپس شروع به شمارش می کنیم. به ازای هر لغت (x) در دسته (c) لازم است. در موارد زیر شمارش شود:

- شمارش در شمارنده لغت x در دسته c
- شمارش در شمارنده total دسته c
- شمارش در شمارنده لغت x در دسته total
- شمارش در شمارنده total دسته total

برای این کار دیکشنری ای از دسته بندی های ساخته که مقدار هر دسته خود یک دیکشنری از لغات است که در آن لغات شمارش می شوند. سپس با پیمایش روی کتاب ها و پیمایش روی لغات آن داده ها را به روشی که بالاتر گفته شد شمارش می کنیم. (اگر دسته بندی کتاب را قبلا تشکیل نداده بودیم تشکیل می دهیم.)

```
categories_data = {"total" : {"total" : 0 , **{word : 0 for word in
all_words}}}
for book in books_train_csv:
   if book["categories"] not in categories_data.keys():
       categories_data[book["categories"]] = {"total" : 0, **{word : 0 for
word in all_words}}
   for word in book["description"]:
       categories_data[book["categories"]][word] += 1
       categories_data[book["categories"]]["total"] += 1
       categories_data["total"][word] += 1
       categories_data["total"]["total"] += 1
```

حال BoW ساخته شده. برای فهم بهتر می توانیم آن را با استفاده از dataframe های کتابخانه ی pandas نمایش دهیم.

```
df = pd.DataFrame(categories_data)
df = df.transpose()
```

	total	ساختار	نظریه	های	جامعه	شناخت #شناس	ایران	نوشته	...	فلاحت	صناعت
total	292599	84	188	6139	664	517	701	940	...	1	1
جامعه شناسی	56331	51	144	1517	454	317	318	157	...	0	0
کلیات اسلام	39832	3	19	612	85	67	58	110	...	0	0
داستان کودک و نوجوانان	24768	0	1	595	5	13	45	155	...	0	0
داستان کوتاه	54734	7	1	1072	34	24	126	171	...	1	1
مدیریت و کسب و کار	46431	12	20	1192	35	50	35	176	...	0	0
رمان	70503	11	3	1151	51	46	119	171	...	0	0

7 rows × 20910 columns

محاسبه احتمالات

محاسبه با لگاریتم

حال باید احتمالات را حساب کنیم. توجه به این نکته ضروریست که وقتی ما احتمالات را حساب می کنیم چون احتمالات کوچکی را در هم ضرب می کنیم عدد به دست آمده به حدی کوچک می شود که پایتون آنها را صفر حساب می کند. برای حل این مشکل ما می توانیم از **لگاریتم احتمالات** استفاده کنیم و در نتیجه ضرب را هم به جمع تبدیل کنیم. در ادامه نیز به جای خود احتمال لگاریتم آن را حساب کرده.

محاسبه p(c)

این احتمال در واقع احتمال این که کتاب های فایل آموزشی در دسته بندی خاصی باشند بدون در نظر گرفتن پارامتر دیگری را بیان می کند. پس به آسانی نسبت تعداد کتاب های هر دسته را به کل می گیریم و احتمال را حساب می کنیم.

برای این کار تعداد کتاب ها را شمارش می کنیم که با پیمایش روی کتاب ها و ذخیره تعداد آن انجام می شود.

```
sum_book = {"total" : 0}
for book in books_train_csv:
   if book["categories"] not in sum_book.keys():
       sum_book[book["categories"]] = 0
   sum_book[book["categories"]] += 1
   sum_book["total"] += 1
```

حال فقط لازم است احتمال را حساب کرده و لگاریتم آن را ذخیره کنیم.

```
log_p_c = dict()
for category in sum_book.keys():
   log_p_c[category] = log10(sum_book[category] / sum_book["total"])
```

نتیجه به صورت زیر است:

```
{'total': 0.0,
 ',0.7781512503836436- :'جامعه شناسی
,0.7781512503836436- :'کلیات اسلام'
,0.7781512503836436- :'داستان کودک و نوجوانان'
,0.7781512503836436- :'داستان کوتاه'
,0.7781512503836436- :'مدیریت و کسب و کار'
0.7781512503836436- :'رمان' }
```

محاسبه p(x|c)

این احتمال بیان می کند که یک مجموعه لغت به چه احتمالی در هر دسته وجد دارند.

برای محاسبه آن ابتدا به دست می آوریم که هر لغت به چه احتمال در هر دسته بندی وجود دارد.

برای محاسبه آن کافیست روی دسته بندی های پیمایش کرده سپس روی لغات آن پیمایش کنیم و نسبت آن را به کل لغات بگیریم.

```
p_x_c_word = dict()

for category in categories_data.keys():
   p_x_c_word[category] = dict()
   for word in categories_data[category].keys():
       p_x_c_word[category][word] = categories_data[category][word] /
categories_data[category]["total"]
```

Additive Smoothing

حال باید لگاریتم این احتمال را برای هر کتاب و با توجه به توضیحات آن به دست بیاوریم. اما مشکلی وجود دارد. باید برای لغاتی که در داده های آزمایشی وجود دارد اما در داده های آموزشی خیر چاره ای اندیشید. برای این کار ما از روش **Additive Smoothing** استفاده می کنیم. این روش بیان می کند که اگر لغتی در BoW ما موجود نبود آن را به اندازه کوچکی (اپسیلون) موجود فرض می کنیم و بعد احتمال آن را حساب می کنیم. ما برای این عدد کوچک متغیر a را گذاشتیم و مقدار آن را ۰.۴ دادیم.(این عدد را دستی تنظیم شده است اما بهتر است برای دقت بهتر برای تعیین آن ساز و کاری تعریف کرد.)

با پیمایش روی کتاب های آزمایشی، دسته بندی ها و سپس لغات لگاریتم احتمال آنها را حساب و با یکدیگر جمع می کنیم و در متغیر مربوطه ذخیره کرده، همچنین اگر لغتی در داده های آموزشی نبود یا تعداد صفر داشت از روشی که بالا گفته شد استفاده می کنیم. (اگر هم نخواهیم از **Additive Smoothing** استفاده کنیم صرفا کد آن خط را با pass جایگزین کرده)

```
log_p_x_c = dict()

for book in books_test_csv:
   log_p_x_c[book["title"]] = dict()
   for category in categories_data.keys():
       log_p_x_c[book["title"]][category] = 0
       for word in book["description"]:
           if word in p_x_c_word[category].keys() and
p_x_c_word[category][word] != 0:
               log_p_x_c[book["title"]][category] +=
log10(p_x_c_word[category][word])
           else:
               # if we don't want to use Additive Smoothing comment next
line
               log_p_x_c[book["title"]][category] += log10(a /
categories_data[category]["total"])
               pass
```

خروجی کد بالا به شکل زیر است:

```
{'کآشوب': {'total': -1297.90618088977732,
  ',1335.6181366632281- :'جامعه شناسی
,1318.0487457036613- :'کلیات اسلام'
,1298.2100363150287- :'داستان کودک و نوجوانان'
,1296.2418822126288- :'داستان کوتاه'
,1358.1117820180332- :'مدیریت و کسب و کار'
1298.5289187745032- :'رمان'  }, ...}
```

محاسبه p(x)

محاسبه p(x) هم درست مثل p(x|c) است با این تفاوت که به جای این که روی دسته ها این احتمال را حساب کنیم روی total حساب می کنیم. لازم به ذکر است که اصلا نیازی هم به محاسبه آن نیست زیرا ما در نهایت می خواهم احتمالات را مقایسه کنیم و در مقایسه عدد ثابت در تمامی احتمالات در نتیجه مقایسه تاثیری ندارد. اما چون در صورت پروژه گفته شده ما نیز آن را لحاظ می کنیم.

کد آن به صورت زیر است:

```
p_x_word = dict()

for word in categories_data["total"].keys():
   p_x_word[word] = categories_data["total"][word] /
categories_data["total"]["total"]

log_p_x = dict()

for book in books_test_csv:
   log_p_x[book["title"]] = 0
   for word in book["description"]:
       if word in p_x_word.keys():
           log_p_x[book["title"]] += log10(p_x_word[word])
       else:
           log_p_x[book["title"]] += log10(a /
categories_data["total"]["total"])
```

محاسبه p(c|x) برای داده های آزمایشی

برای محاسبه این احتمال ما از قانون بیز به صورت زیر استفاده می کنیم.

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}$$

که بعد از اعمال لگاریتم به صورت زیر می شود:

$$log\,(p(c|x)) = log\,(p(x|c)) + log\,(p(c)) - log(p(x))$$

با پیمایش روی کتاب ها و دسته بندی ها، احتمالات را حساب و ذخیره می کنیم.

```
log_p_c_x = dict()

for book in books_test_csv:
   log_p_c_x[book["title"]] = dict()
   for category in categories_data.keys():
       log_p_c_x[book["title"]][category] =
log_p_x_c[book["title"]][category] + log_p_c[category] -
log_p_x[book["title"]]
```

انتخاب پر احتمال ترین دسته بندی برای هر کتاب

حال که احتمالات به دست آمد باید پراحتمال ترین دسته را به عنوان حدس خود اعلام کنیم.
در قطعه کد زیر ما پر احتمال ترین دسته را انتخاب و در کنار جواب واقعی می گذاریم و ذخیره می کنیم.

```
result = {
   i: {
       "guess": max((value, key) for key, value in log_p_c_x[i].items() if
key != 'total')[1],
       "real category": next(j["categories"]
                             for j in books_test_csv if j["title"] == i)
   } for i in log_p_c_x.keys()
}
```

محاسبه میزان دقت

برای محاسبه میزان دقت کافیست که بین حدس خود و جواب مقایسه ای انجام دهیم و در نهایت درصد درستی را حساب کنیم.

```
correct_amount = 0
for i in result.keys():
   if result[i]["guess"] == result[i]["real category"]:
       result[i]["is true"] = True
       correct_amount += 1
   else:
       result[i]["is true"] = False

correct_amount *= 100 / len(result)
correct_amount
```

در نهایت خروجی با استفاده از پانداس به صورت زیر می شود:

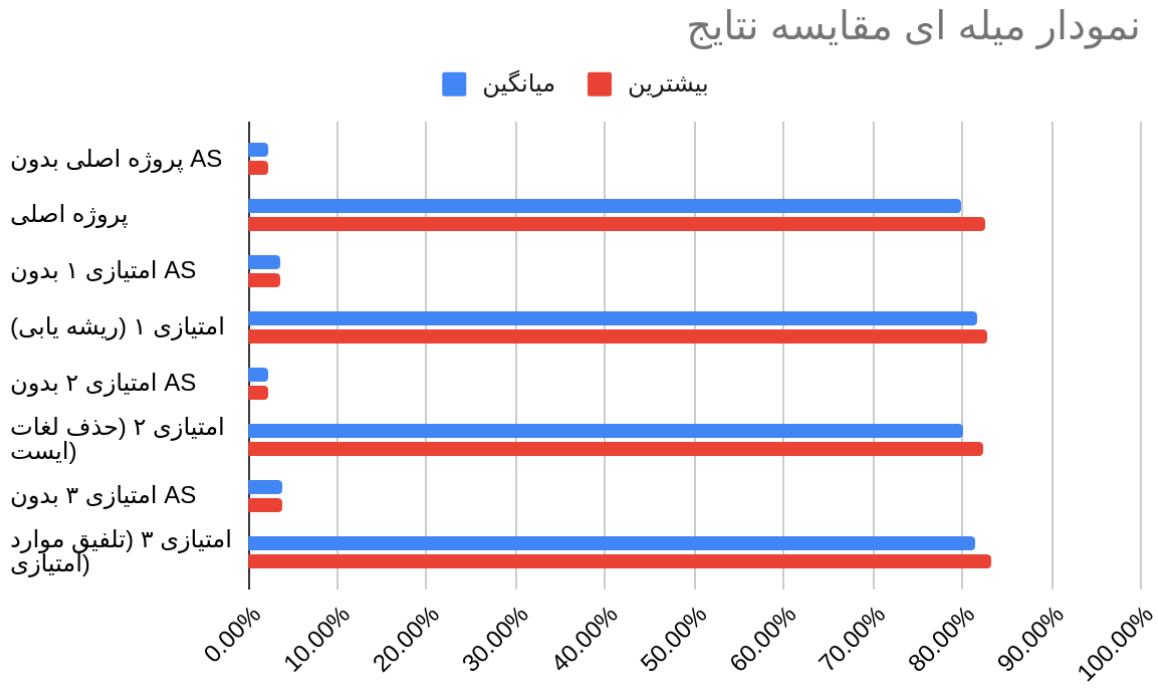
	guess	real category	is true
کآشوب	داستان کوتاه	داستان کوتاه	True
داستان های برق آسا	داستان کوتاه	داستان کوتاه	True
بحثی درباره مرجعیت و روحانیت	کلیات اسلام	کلیات اسلام	True
قلعه ی حیوانات	رمان	رمان	True
(قصه ما مثل شد (۱	داستان کودک و نوجوانان	داستان کودک و نوجوانان	True
...	...	...	...
(سیره اقتصادی امام علی (ع	کلیات اسلام	کلیات اسلام	True
تفنگ پدر بر بام های تهران	داستان کودک و نوجوانان	رمان	False
اصالت من	جامعه شناسی	جامعه شناسی	True
تأمین مالی آموزش و پرورش در ایران	جامعه شناسی	جامعه شناسی	True
(جایی که ماه نیست (شوک سقوط	داستان کودک و نوجوانان	رمان	False

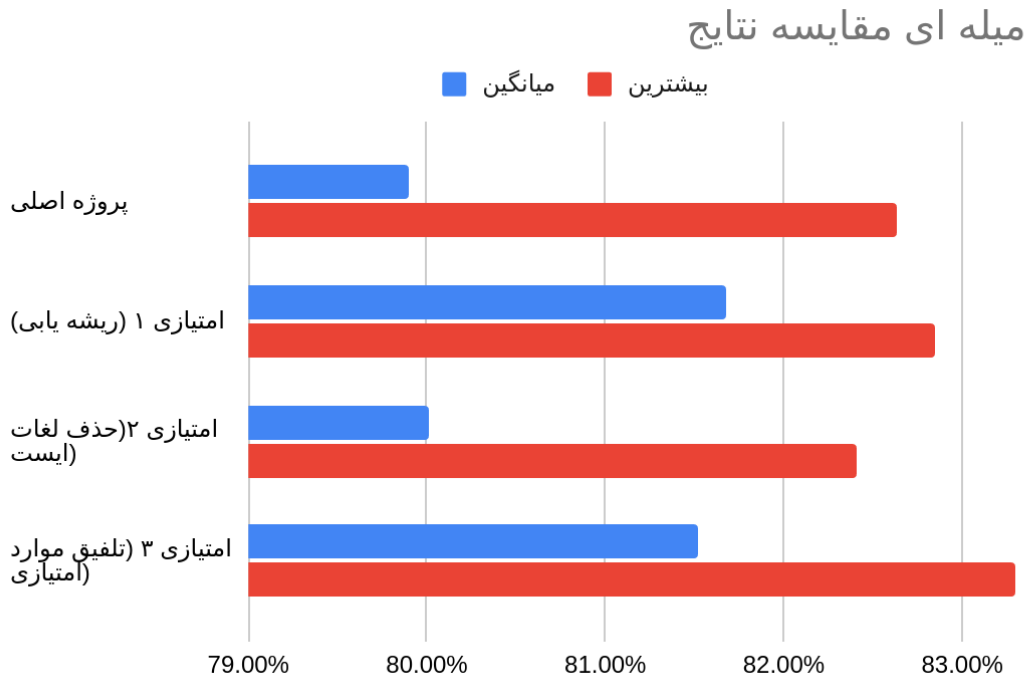
449 rows × 3 columns

نتایج

در نهایت نتایج با ضرایب مختلف برای (AS) Additive Smoothing و یا بدون در نظر گرفتن آن محاسبه شده در جدول و نمودار زیر ارائه می گردد. همچنین لازم به ذکر است زمان اجرای برنامه چیزی بین ۶ تا ۱۰ ثانیه می باشد.

	ضریب AS	دقت	میانگین	بیشترین
پروژه اصلی بدون AS	بدون AS	2.22%	2.22%	2.22%
پروژه اصلی	1	75.95%	79.90%	82.63%
	0.4	81.07%		
	0.3	82.63%		
	0.1	79.96%		
امتیازی ۱ بدون AS	بدون AS	3.56%	3.56%	3.56%
امتیازی ۱ (ریشه یابی)	1	79.51%	81.68%	82.85%
	0.4	82.85%		
	0.3	82.85%		
	0.1	81.51%		
امتیازی ۲ بدون AS	بدون AS	2.22%	2.22%	2.22%
امتیازی ۲ (حذف لغات ایست)	1	74.39%	80.01%	82.41%
	0.4	81.29%		
	0.3	82.41%		
	0.1	81.95%		
امتیازی ۳ بدون AS	بدون AS	3.78%	3.78%	3.78%
امتیازی ۳ (تلفیق موارد امتیازی)	1	79.06%	81.51%	83.30%
	0.4	82.18%		
	0.3	83.30%		
	0.1	81.51%		


نمودار میله ای مقایسه نتایج
میانگین
بیشترین
پروژه اصلی بدون AS
پروژه اصلی
امتیازی ۱ بدون AS
امتیازی ۱ (ریشه یابی)
امتیازی ۲ بدون AS
امتیازی ۲ (حذف لغات ایست)
امتیازی ۳ بدون AS
امتیازی ۳ (تلفیق موارد امتیازی)
0.00%
10.00%
20.00%
30.00%
40.00%
50.00%
60.00%
70.00%
80.00%
90.00%
100.00%


نمودار میله ای مقایسه نتایج
میانگین
بیشترین
پروژه اصلی
امتیازی ۱ (ریشه یابی)
امتیازی ۲(حذف لغات ایست)
امتیازی ۳ (تلفیق موارد امتیازی)
79.00%
80.00%
81.00%
82.00%
83.00%

نتیجه گیری

با بررسی نتایج توجه می شویم که اگر از لغاتی که در BoW وجود ندارد صرف نظر کنیم و از AS استفاده نکنیم نتیجه برنامه تقریبا نادرست می شود و درست کار نمی کند اما وقتی از AS استفاده می کنیم دقت به صورت قابل توجه افزایش می یابد و به میزان مطلوبی میرسد. نکته عجیب این است که وقتی کلمات را ریشه یابی می کنیم و لغات ایست را حذف می کنیم با این که تعداد لغات $\frac{۳}{۵}$ می شود و تقریبا ۲۰۰ هزار لغت حذف می شود اما دقت برنامه فقط چیزی حدود ۱.۵٪ درصد افزایش می یابد که میزان ناچیزی در مقابل افزایش بار پردازشی است. حتی در بخش ۲ امتیازی، بیشترین میزان دقت با حذف کلمات ایست کاهش میابد. این موضوع می تواند به این علت باشد که کلمات ایست ما عمومی است و برای این پروژه شخصی سازی نشده است. البته این موضوع نیاز به بررسی بیشتر دارد. از نمودار میتوان دریافت که ریشه یابی کلمات موثر تر از حذف لغات ایست است.

در نهایت می توان دریافت که مدل بیز که در این پروژه پیاده شده نسبت به سادگی دقت خوبی به ما ارائه می دهد. دقت برنامه حدود ۸۰ درصد است که این عدد مناسب به نظر می رسد.