دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

ترکیب داده/ اطلاعات

گزارش پروژه‌ی درس

محمدحسین بدیعی

شماره دانشجویی 810199106

گرایش : کنترل – هوش مصنوعی و رباتیک

استاد :جناب آقای دکتر بهزاد مشیری

بهار 1399-1400

مقدمه

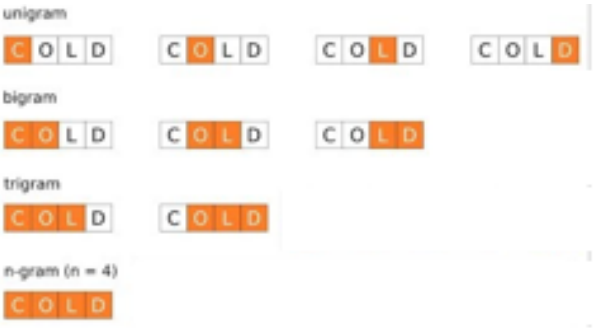
پردازش زبان طبیعیِ انسان یا NLPها کاربردِ محسوسی در سیستم‌های هوشمند دارند. از جمله کاربردهای آن برقراری یک تعامل دو طرفه بین انسان و ماشین است. در صورت برقراری این ارتباط به نحوِ مطلوب، ماشین قادر خواهد بود مکالمات و نوشته‌های انسانها را تشخیص داده و پاسخی مناسب را به آنها ارائه کند. نمود اجرا شده از این کاربردها را می‌توان در دستیارِ هوشمندِ اپل و گوگل مشاهده کرد. کاربردِ دیگری که حائز اهمیت می‌باشد، تحلیل و آنالیزِ داده‌های گفتاری و نوشتاریِ انسان‌هاست که هم اکنون شرکت‌هایی مانند کمپانی اپل، ماکروسافت، آمازون و ... بر روی این حوزه تحقیقاتِ وسیعی را انجام می‌دهند. یکی از اهدافِ این آنالیز کردن‌ها، ارائه‌ی خدماتی بهتر به مشتریان و دستیابی به سوددهیِ بیشتر برای شرکت است. نمونه‌ی عملیِ این واقعیت را می‌توان در تحلیلِ فیدبک‌های گرفته شده از مشتریانِ آمازون دانست. این فیدبک‌ها می‌توانند یک کالا را از دسته‌ی غیر محبوب به محبوب برده و یا به دسته‌ی پیشنهادهای ویژه‌ی سایت اضافه کند. از دیگر کاربردهای NLP نیز مباحثِ مربوط به hate speechها می‌باشد.

سیستمی که ما در این پروژه موفق به پیاده‌سازیِ آن شدیم یک سیستم تشخیصِ شاعر از روی مصراع داده‌شده به آن می‌باشد. این سیستم با آنالیزِ کلماتِ موجود در این مصراع قابلیتِ آن را دارد که با حداکثر دقتی در حدود 82.5% شاعر را تنها از روی یک مصراع تشخیص دهد.

مراحل اجرای پروژه

برای پیاده‌سازیِ این پروژه از مدل‌های unigram، bigram، backoff و همچنین کلاسیفایرِ naïve Bayes و مالتی کلاسیفایر سیستمِ soft majority voting و hard majority voting و همچنین از چندین رویکردِ ترکیبِ اطلاعاتِ مدل‌ها استفاده نمودیم که در ادامه به توضیحِ آن خواهیم پرداخت.

در ابتدا به پیاده سازی از سه مدل زبانی پرداختیم. یک مدلِ زبانی به صورتِ یک توزیعِ احتمالاتی پیاده‌سازی می‌شود. این مدل‌ها شامل تخمین‌هایی از کلمات با توجه به کلماتِ ماقبلِ آن کلمه (به تعدادی مشخص) می‌شوند. مدل های n-grams در واقع کلمات یا عبارات و .. را به گرام‌های کوچکتر تقسیم میکنند. این گرام ها برای یک کلمه، حرفهای موجود در کلمه است و برای یک جمله، کلماتِ موجود در آن و الی آخر. برای نمونه به شکل زیر توجه بفرمایید.

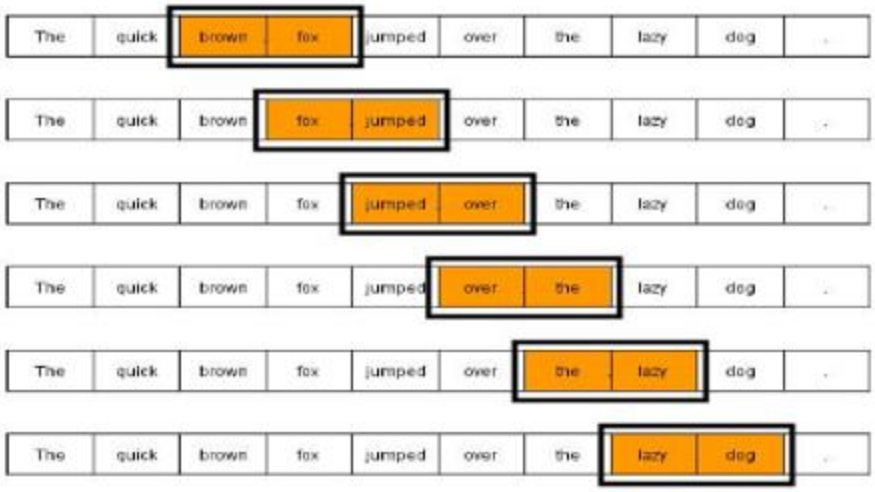


نمونه ای از یک مدلِ NGRAM که واحدهای آن حروف می‌باشند

همانطور که در شکل فوق مشاهده می‌کنید یک کلمه را به گرام‌های کوچک تقسم کردیم. این گرام‌ها ابتدا تکی هستند و در سطر دوم دوتایی و در سطر سوم سه تایی و در سطر آخر نیز چهار تایی می‌باشند.

در بحث مربوط به text classification، اغلب با جملات سر و کار داریم و لذا این جملات را به گرام هایی که در واقع جزئی از جمله یا به عبارتِ بهتر کلماتِ موجود در این جملات هستند، تقسیم می‌نماییم.

برای مثال جمله‌ی زیر را در نظر داشته باشید که به گرام‌های دوتایی تقسیم شده اند.



نمونه ای از یک مدلِ BIGRAM که واحدهای آن کلمه می‌باشند

بحث گرام ها از اینجا اهمیت پیدا می‌کند که برای تحلیل به جهتِ دسته‌بندیِ متون، تک تک کلمات می‌توانند اثر گذار باشند و به همین جهت مدل‌های bigram، trigram و ... نیز حائز اهمیت هستند.

احتمالاتِ شرطیِ هر یک از کلماتِ موجود در داده‌های آموزش به صورت زیر بدست آورده و در یک دیکشنری ذخیره کردیم.

احتمال شرطی در unigram:

$$P(w_i) = \frac{C(w_i)}{N} \quad , \quad C(w_i) = count\ of\ occurance\ of\ w_i\ ,$$

$$N = total\ number\ of\ words\ in\ the\ training\ data$$

سپس احتمالاتِ شرطیِ bigram را محاسبه نمودیم.

احتمال شرطی در bigram:

$$P(w_i|w_{i-1}) = \frac{C(w_i|w_{i-1})}{C(w_{i-1})}$$

و در نهایت نیز با ترکیبِ این دو مدل و اضافه نمودنِ بایاس، یک backoff مدل ایجاد نمودیم.

از این رو برای این بخش یک کلاسِ Documentation در پایتون ایجاد کردیم و تمامیِ مدل‌های فوق را در آن پیاده نمودیم.

لذا هر یک از این توابع در کلاس به صورت زیر است.

تعریفِ کلاس:

```
class DocumentClassification:
    def __init__(self, category):
        self.category = category
        self.documents = []
        self.unary_probabilities = {}
        self.frequency = {}
        self.binary_probabilities = {}
        self.binary_probabilities_backed_off = {}
        self.p = None
```

استخراجِ فرکانسِ کلمات در یک عبارت:

```
    def get_word_frequency(self):
        words_frequency = {'UNK': 0}
        for i in self.documents:
            for j in i.word_list:
                if j not in words_frequency.keys():
                    words_frequency['UNK'] = words_frequency.get('UNK') + 1
                    words_frequency[j] = 1
                else:
                    words_frequency[j] = words_frequency.get(j) + 1
        total_count = sum([v for v in words_frequency.values()])
        self.frequency = words_frequency
        return [words_frequency, total_count]
```

استخراجِ احتمالاتِ unary:

```
    def calculate_unary_probabilities(self):
        frequency, total_count = self.get_word_frequency()
        self.unary_probabilities = {k: v/total_count for k, v in frequency.items()}
```

استخراجِ احتمالاتِ باینری:

```
def calculate_binary_probabilities(self):
    pair_wise_words_frequency = {}

    for i in self.documents:
        for j, k in self.pair_wise(i.word_list):
            # print(j, k)
            if f"{j} {k}" in pair_wise_words_frequency:
                pair_wise_words_frequency[f"{j} {k}"] = pair_wise_words_frequency.get(f"{j} {k}") + 1
            else:
                pair_wise_words_frequency[f"{j} {k}"] = 1
    for key, value in pair_wise_words_frequency.items():
        self.binary_probabilities[key] = value/self.frequency.get(key.split()[0])
```

Pair-wise نمودنِ کلمات در یک عبارت برای بدست آوردنِ احتمالاتِ باینری:

```
@staticmethod
def pair_wise(word_list):
    a, b = tee(word_list, 2)
    next(b, None)
    return zip(a, b)
```

بدست آوردنِ احتمالاتِ backoff برای ساخت مدل:

```
def binary_back_off(self):
    y = [0.1,0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
    for key, value in self.binary_probabilities.items():
        self.binary_probabilities_backed_off[key] = \
            max([(i * self.unary_probabilities.get(key.split()[1])) + ((1 - i) * value) for i in y])
```

توجه داشته باشید که یکی از مشکلاتی که در پردازشِ زبانِ طبیعی با آن مواجه هستیم وجودِ داده‌هایی است که در داده‌های آموزش وجود ندارند و مدل ها احتمالاتِ مربوط به هر یک را برابر با صفر قرار می‌دهد و لذا کلِ عبارت را از عرصه‌ی تشخیص خارج می‌کند که ما این مشکل را با log likelihood حل نمودیم.

$$P_{eval}(\text{text}) = \prod_{\text{word}} P_{train}(\text{word})$$

$$\log(P_{eval}(\text{text})) = \sum_{\text{word}} \log(P_{train}(\text{word}))$$

$$\text{Average log likelihood}_{eval} = \frac{\sum_{\text{word}} \log(P_{train}(\text{word}))}{N_{eval}}$$

N_{eval}: total number of words in evaluation text

حال که فرکانس‌ها و احتمالات کلمات را به صورت یونری و باینری محاسبه کردیم، می‌توانیم عباراتمان را با توجه با توجه به احتمالاتی که برای گرام‌ها بدست آورده‌ایم برای ساخت مدل آماده کنیم و این احتمالات را متناسب با نوعِ خود در یک دیکشنری ذخیره کنیم.

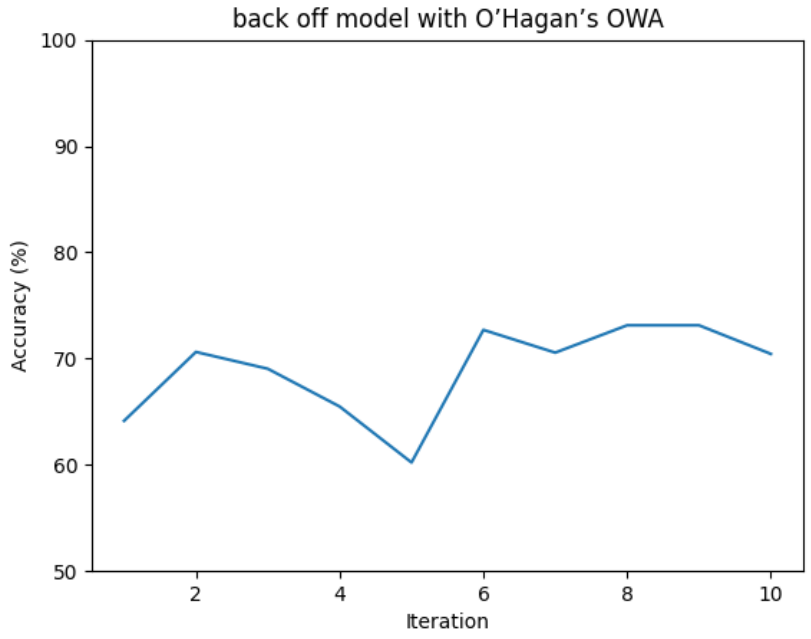
```
def get_unary_probability(self, doc):
    result = 1
    for i in doc.word_list:
        p = self.unary_probabilities.get(i)
        result += math.log10(p) if p is not None else math.log10(0.00000001)
    return result + math.log10(self.p)

def get_binary_probability(self, doc):
    result = 1
    for i, j in self.pair_wise(doc.word_list):
        temp = self.binary_probabilities.get(f"{i} {j}")
        result += math.log10(temp) if temp is not None else math.log10(0.00000001)
    return result + math.log10(self.p)

def get_binary_back_off_probability(self, doc):
    result = 1
    for i, j in self.pair_wise(doc.word_list):
        temp = self.binary_probabilities_backed_off.get(f"{i} {j}")
        result += math.log10(temp) if temp is not None else math.log10(0.00000001)
    return result + math.log10(self.p)
```

حال مدل‌ها را با توجه به توابع اولیه‌ای که در فوق مشاهده می‌کنید تعریف کرده‌ایم که قسمت‌های بیشتر را می‌توانید در فایلهای موجود در کد مشاهده کنید.

سپس با استفاده از متدهای مختلفِ دیتا فیوژن ادغام به ترکیبِ اطلاعاتِ این مدل‌ها نمودیم که نتایجِ آن به صورت زیر است.



از اولین متدی که برای ترکیبِ اطلاعاتِ احتمالاتیِ موجود در مدل‌ها استفاده نمویم، O’Hagan بود.

صحتی که این متد برای تشخیصِ متون در اختیارِ ما قرار داد چیزی در حدود 68.94738214668806 % بود. همانطور که مشاهده می‌نمایید، این نتایج برای ده نمونه‌ی مختلف از داده‌های test و train استخراج شده است. نتایج confusion matrix نیز به صورت زیر است که برای تعدادی از فایل‌های آموزش و تست قرار داده‌ایم.

	ferdowsi	hafez	molavi
ferdowsi	615	29	51
hafez	150	291	60
molavi	170	52	321

	ferdowsi	hafez	molavi
ferdowsi	563	27	47
hafez	26	36	16
molavi	170	52	321

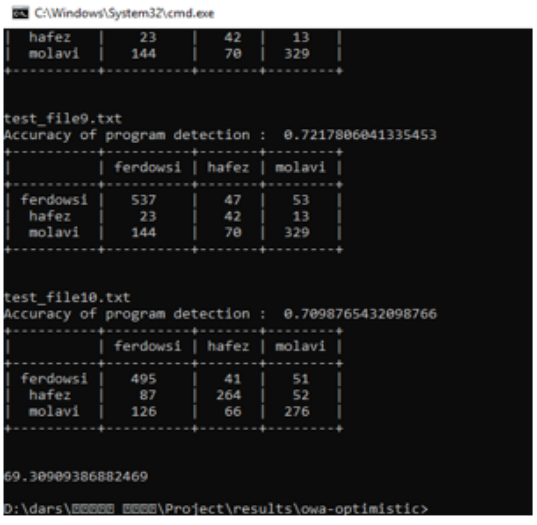
	ferdowsi	hafez	molavi
ferdowsi	563	27	47
hafez	26	36	16
molavi	170	52	321

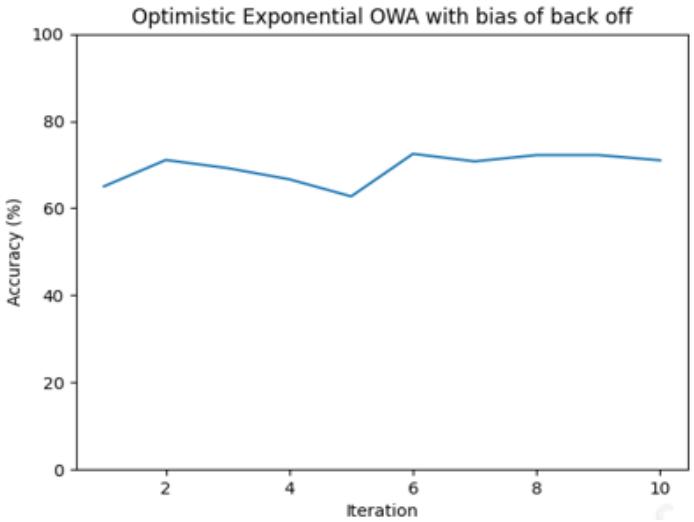
	ferdowsi	hafez	molavi
ferdowsi	516	27	44
hafez	99	246	58
molavi	152	51	265

در گامِ بعدی از متدِ فیوژن optimistic exponential و البته با اضافه نمودنِ بایاس به مدل استفاده کردیم که نتایجِ آن به صورتِ زیر درآمد.

Orness = 0.8 و لذا وزن‌ها به صورت زیر در آمدند:

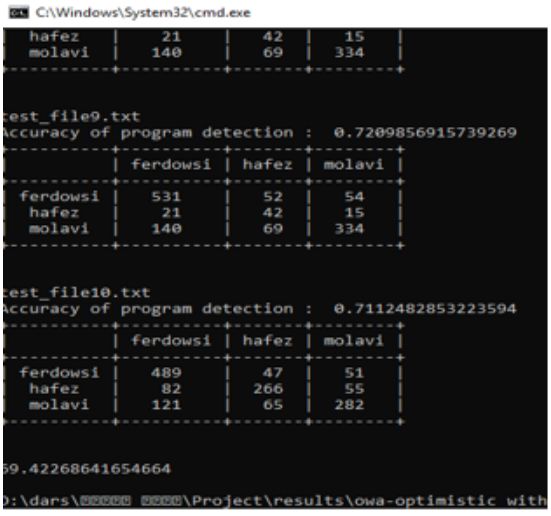
w1 = 0.6 w2 = 0.24 w3 = 0.16

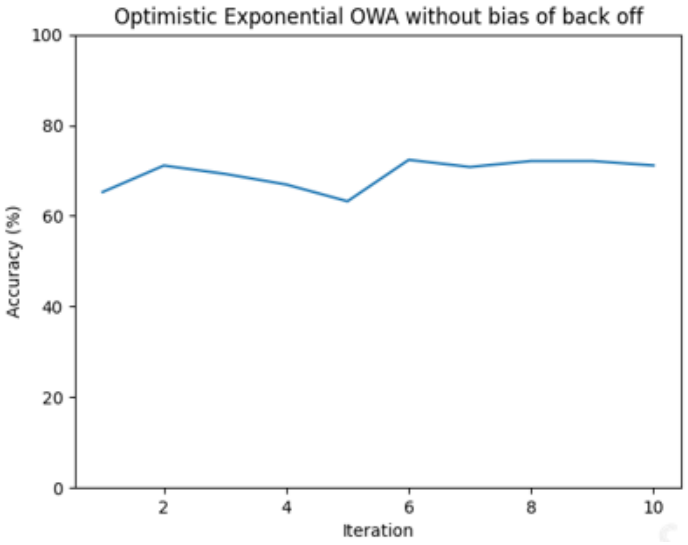




میانگین صحتِ بدست آمده با Orness = 0.8 برای این مدل نیز برابر با 69.30909386882469 % بدست آمد.

سپس بایاس را حذف نمودیم و تاثیر بایاس در شبکه‌ی گرام ها را بررسی کردیم. نتایج به صورت زیر در آمد



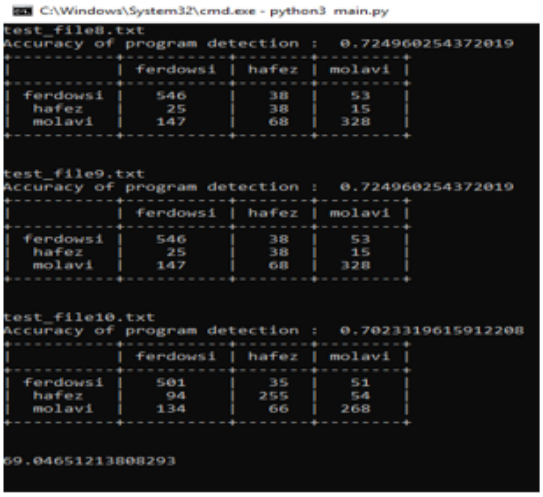


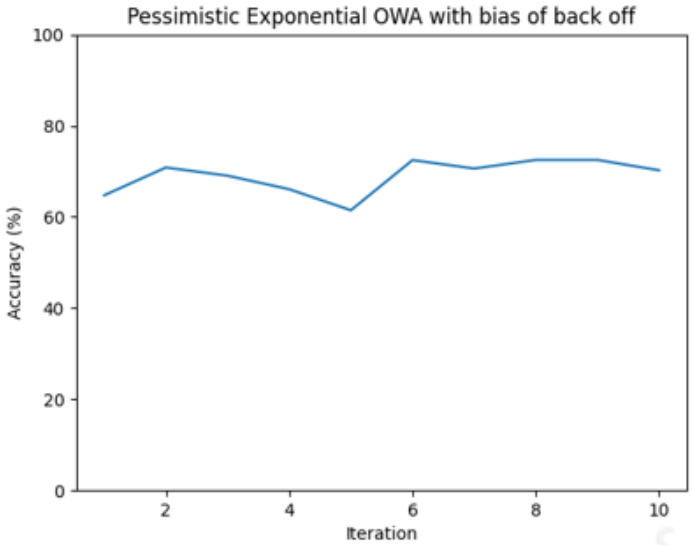
در این مدت میانگین صحت تا حد بسیار کمی برای دیتاست‌های مشابه افزایش یافت و برابر با 69.42268641654664 % شد.

متد فیوژن بعدی که به backoff اعمال نمودیم Pessimistic Exponential OWA با orness = 0.8 بود که نتیجه به صورتِ زیر درآمد.

با توجه به این orness وزن‌ها به صورت زیر درآمدند.

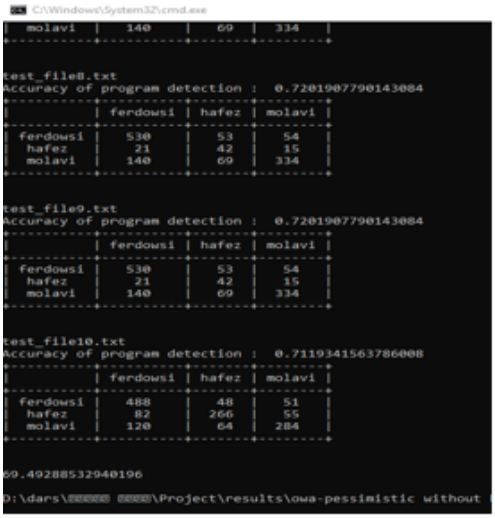
w1 = 0.4 w2=0.36 w3=0.24

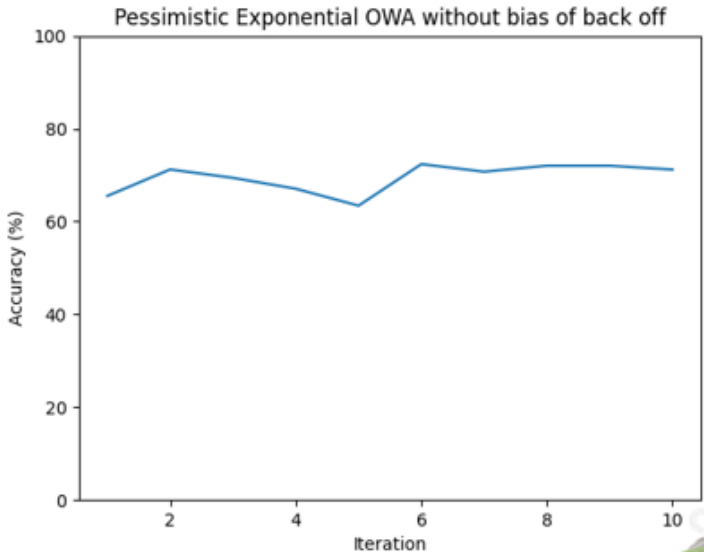




میانگینِ صحت برای این متد در حضورِ بایاس در شبکه برابر با 69.04651213808293 % بدست آمد.

و مجددا بایاس را از سیستم حذف نمودیم و نتیجه زیر را حاصل کرد که مجددا با افزایشِ 0.4 درصد برای صحت همراه بود.





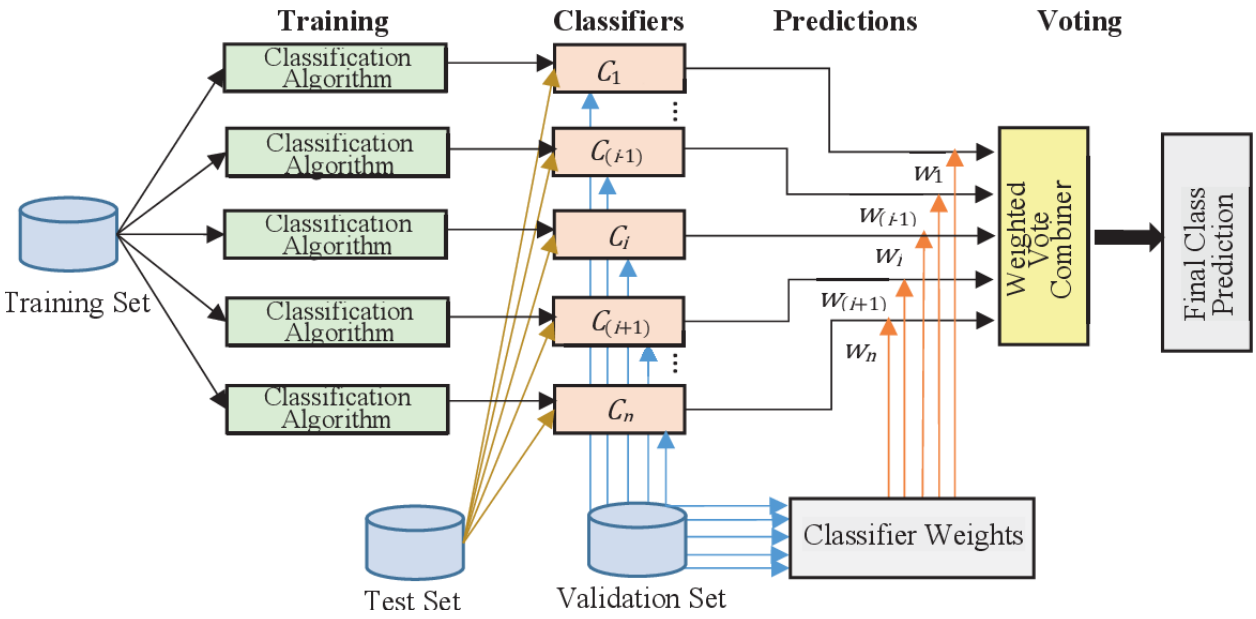
که صحتِ بدست آمده برای این مدل برابر با 69.49288532940196 % شد.

لذا به طور کلی صحت بدست آمده برای متدهای فیوژنِ فوق باحضورِ یا عدمِ حضورِ بایاس به شکل زیر شد.

Fusion Method	Average accuracy
O'Hagan's OWA	68.94738214668806 %
Optimistic Exponential OWA with bias of back off	69.30909386882469 %
Optimistic Exponential OWA without bias of back off	69.42268641654664 %
Pessimistic Exponential OWA with bias of back off	69.04651213808293 %
Pessimistic Exponential OWA without bias of back off	69.49288532940196 %

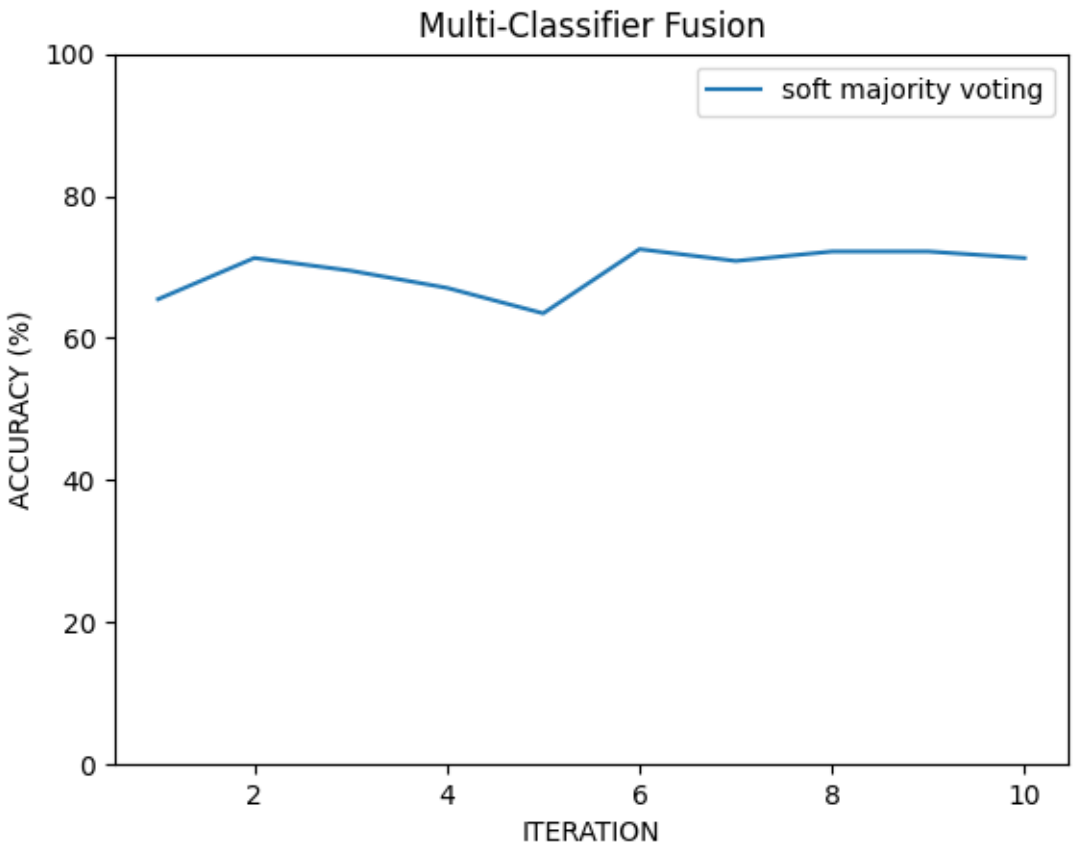
Multi-Classifier Fusion System

در این گام از soft majority voting و hard majority voting استفاده نمودیم که صحت بالاتری را تسب به متدهای فیوژنی که در مرحله‌ی قبل بررسی نمودیم، ارائه داد.



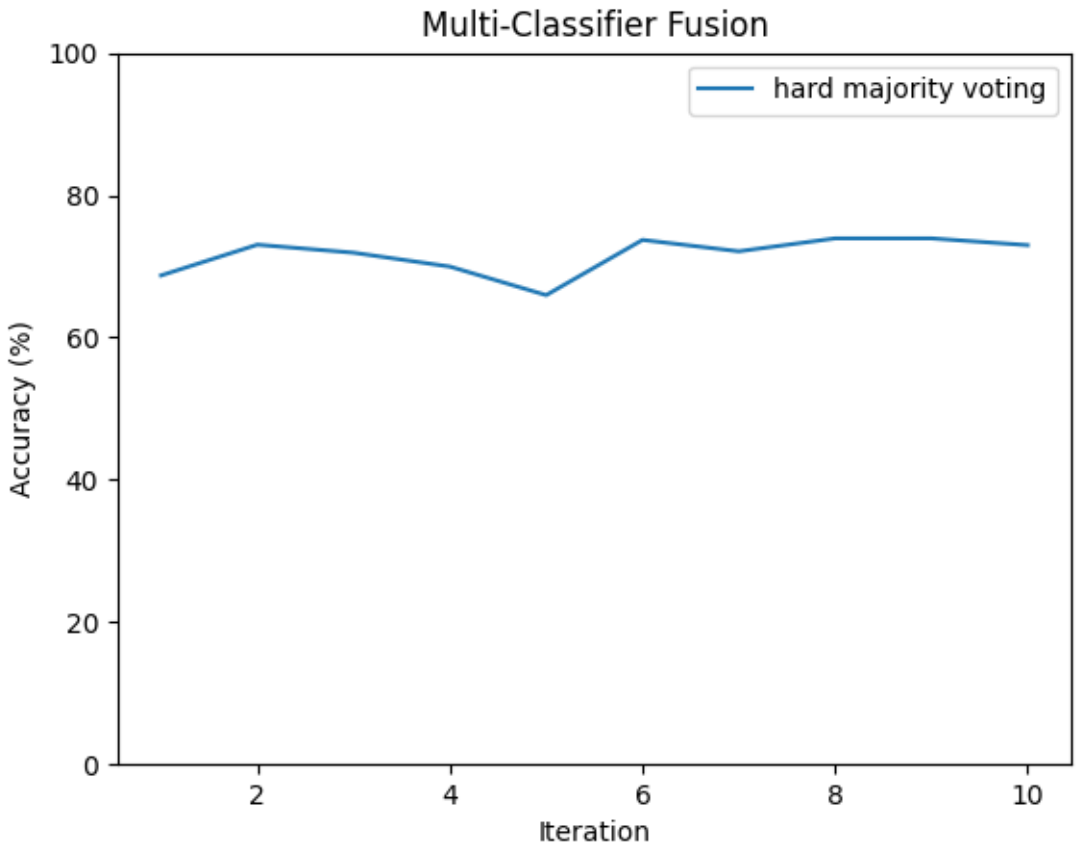
خروجی حاصل برای soft majority voting به صورت زیر است.

Soft Majority Voting



صحتِ بدست آمده برای این الگوریتمِ مالتی کلاسیفایر برابر با 69.57154727221628 % می‌باشد که به طور میانگین حدودِ 0.3 درصد از میانگین صحت‌های متدهای فیوژن بیشتر است و نشان از کارایی این روش در هنگامِ استفاده از چندین مدل را ارائه می‌دهد. برای soft majority voting احتمالات خروجی از مدل‌ها را برای تخصیص به یک کلاسِ خاص با یکدیگر جمع زدیم و هر کدام که بیشترین مقدارِ تجمیعی را داشت به عنوانِ برچسبِ مربوط به آن ذخیره نمودیم.

Hard Majority Voting

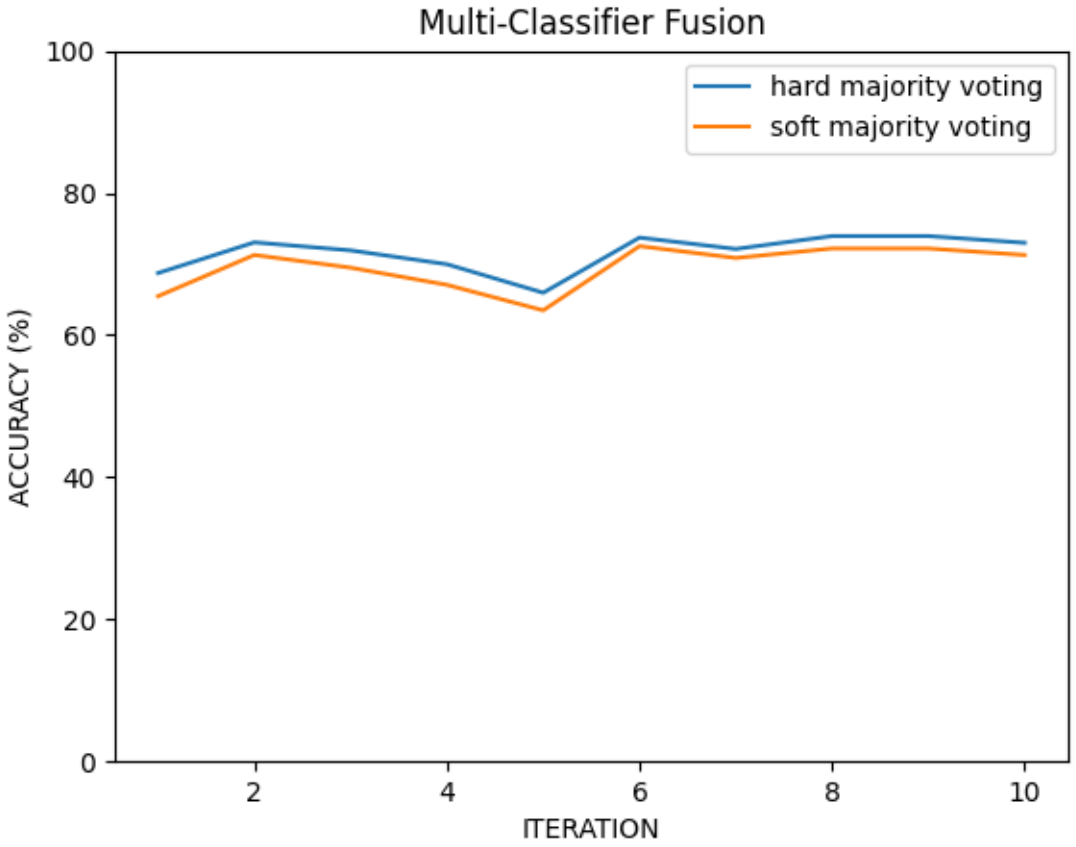


در این روش که خروجی‌های مدل‌ها را نه به صورتِ احتمالاتی بلکه به صورتِ تصمیم برای تشخیصِ برچسب کرفتیم، به بیشترین صحت تا این لحظه رسیدیم.

میانگین صحتِ این مدل برای مجموعه دیتاست‌های فوق برابر با 71.62333879424502% و با انحراف معیارِ 2.635618776987621 رسیدیم.

دلیلِ این امر می‌تواند قدرتِ بالای majority باشد زمانیکه دو یا تعداد بیشتری مدل تصمیم به برچسب زدنِ یکسان به یک دیتا به صورت همزمان باشند. طبیعتا هر چه آرا بیشتر باشد و تصمیمِ مدل‌های بیشتری برای

اختصاص دادنِ برچسبِ خاصی به داده باشد، لذا آن داده با احتمالِ بالاتری متعلق به همان کلاسِ برچسب زده شده است.



Naïve Bayes Classifier with Bigram and Unigram model

همانطوری که می‌دانیم قانونِ بیز روشی برای دسته‌بندی بر پایه‌ی احتمال وقوع یا عدمِ وقوع یک پدیده است که کاربردش را در کنار مدل‌های ترکیب شده با owa و همچنین hard majority voting و soft majority voting مشاهده نمودیم. در واقع در این گام به استفاده از مدلهای unigram و bigram به صورتِ جداگانه و

با بهره‌گیری از کلاسیفایرِ naïve Bayes می‌باشد، می‌پردازیم. در واقع ما در این مرحله با استفاده از قانونِ بیز، برای این مدلها به صورتِ جداگانه، محتمل‌ترین زبان را انتخاب می‌کنیم.

$$l^* = argmax\ P(l|c_{1:N})$$

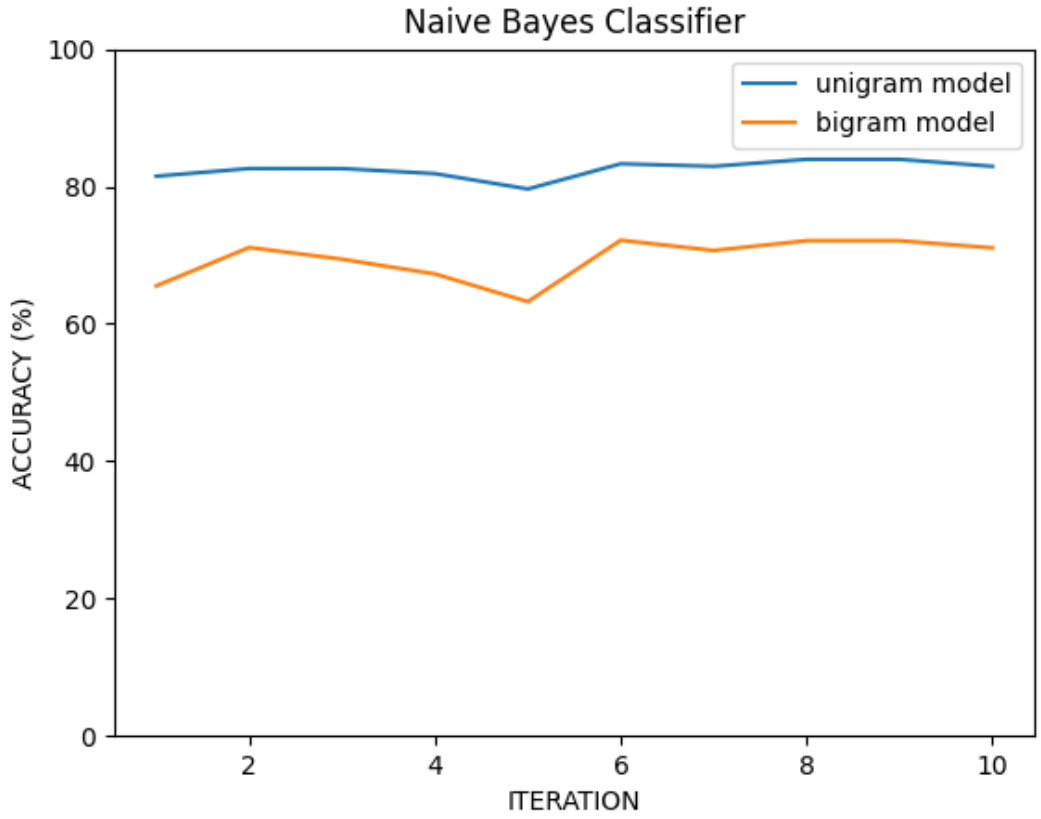
$$= argmax\ P(l)P(c_{1:N}|l)$$

$$= argmax\ P(l)\prod_{i=1}^{N} P(c_i|c_{i-2:i-1}, l)$$

$$P(l|c_{1:N}) = \frac{P(l)P(c_{1:N}|l)}{P(c_{1:N})}$$

حال نتایج این دو مدل را به صورت جداگانه در یک شکل ارائه می‌کنیم.

خروجی مدل‌های unigram و bigram:



بررسی میانگین صحت و انحراف معیار:

	Average accuracy of accuracy curve	Standard deviation of accuracy curve
unigram	**82.51816581708587**	**1.2860424633208134**
bigram	**69.4594504770292**	**3.11665339550396**

همانطور که مشاهده می‌کنید بهترین نتیجه‌ی ممکن را unigram با صحتی نزدیک به 82.51% در اختیار ما قرار داده است.

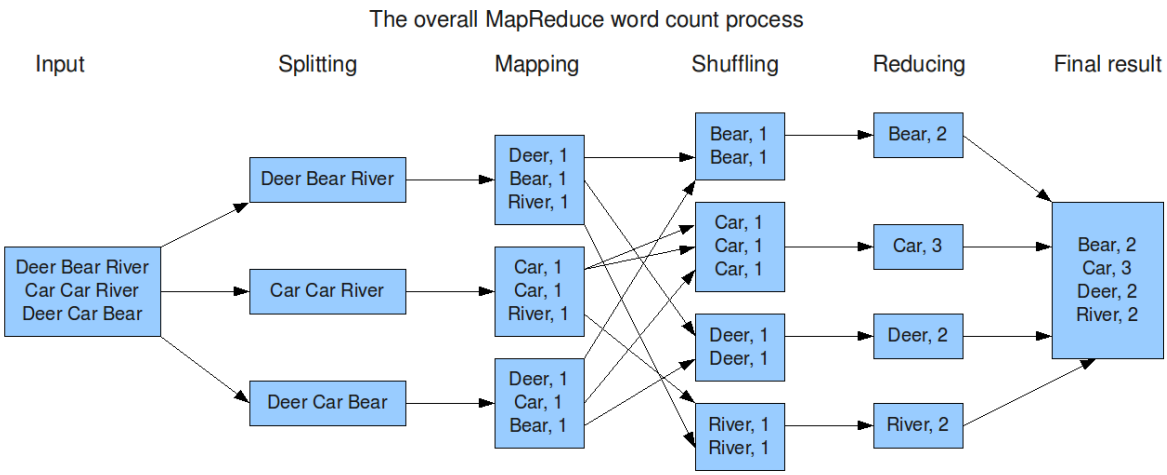
حال تمامی نتایج را در کنار یکدیگر قرار می‌دهیم و یک ارزیابی کلی انجام داده و سپس به سراغِ الگوریتمِ MapReduce در سیستمِ HDFS برای پردازش‌های زبانِ طبیعی خواهیم رفت.

	Average accuracy of accuracy curve	Standard deviation of accuracy curve
Backoff model with O'Hagan's OWA	**68.947 %**	**4.320**
Backoff model with optimistic exponential OWA	**69.422 %**	**3.210**
Backoff model with optimistic exponential OWA and bias	**69.309 %**	**3.396**
Backoff model with pessimistic exponential OWA	**69.492 %**	**3.111**
Backoff model with pessimistic exponential OWA and bias	**69.0465 %**	**3.762**
Hard majority voting	**71.623 %**	**2.635**
Soft majority voting	**69.571 %**	**3.160**
Unigram model with naïve Bayes	**82.518 %**	**1.286**
Bigram model with naïve Bayes	**69.459 %**	**3.116**

همانطور که از نمودارِ فوق مشاهده می‌شود بهترین عملکرد را unigram و سپس hard majority voting و سپس soft majority voting، سپس Backoff model with pessimistic exponential OWA، سپس bigram و بعد از آن نیز متدهای دیگر قرار می‌گیرند. دلیل این امر را می‌توان در دیتاست جستجو کرد که به صورت شعرهایی هستند که ممکن است هر شاعر از کلماتی منحصر بفرد استفاده نماید که شاعر دیگری از این کلمات بهره نگیرد. همچنین تعداد تعکرارِ کلماتی خاص توسطِ شاعران نیز از این قاعده مستثنی نیست. دلایلِ مربوط به majority voting و عماکردِ خوب آن را نیز در بخشِ خود بررسی نمودیم. و همچنین در اینجا بینِ متدهای مختلفِ OWA، نمایی بدبینانه بهترین عملکرد را از خود نشان داده که از آن می‌توان استنباط کرد که از آنجا که در این روش، محتاطانه وزن‌ها انتخاب می‌شوند، لذا نتایجِ مدل‌ها هر یک برای تشخیص دارای اطلاعاتِ کافی برای تصمیم گیری می‌باشند و به همین دلیل است که وزن‌ها می‌توانند در یک محدوده‌ی نزدیک به هم برای ضریبِ هر یک از مدل‌ها قرار گیرند.

استفاده از سیستمِ HDFS برای انجامِ پردازش‌های زبانِ طبیعی

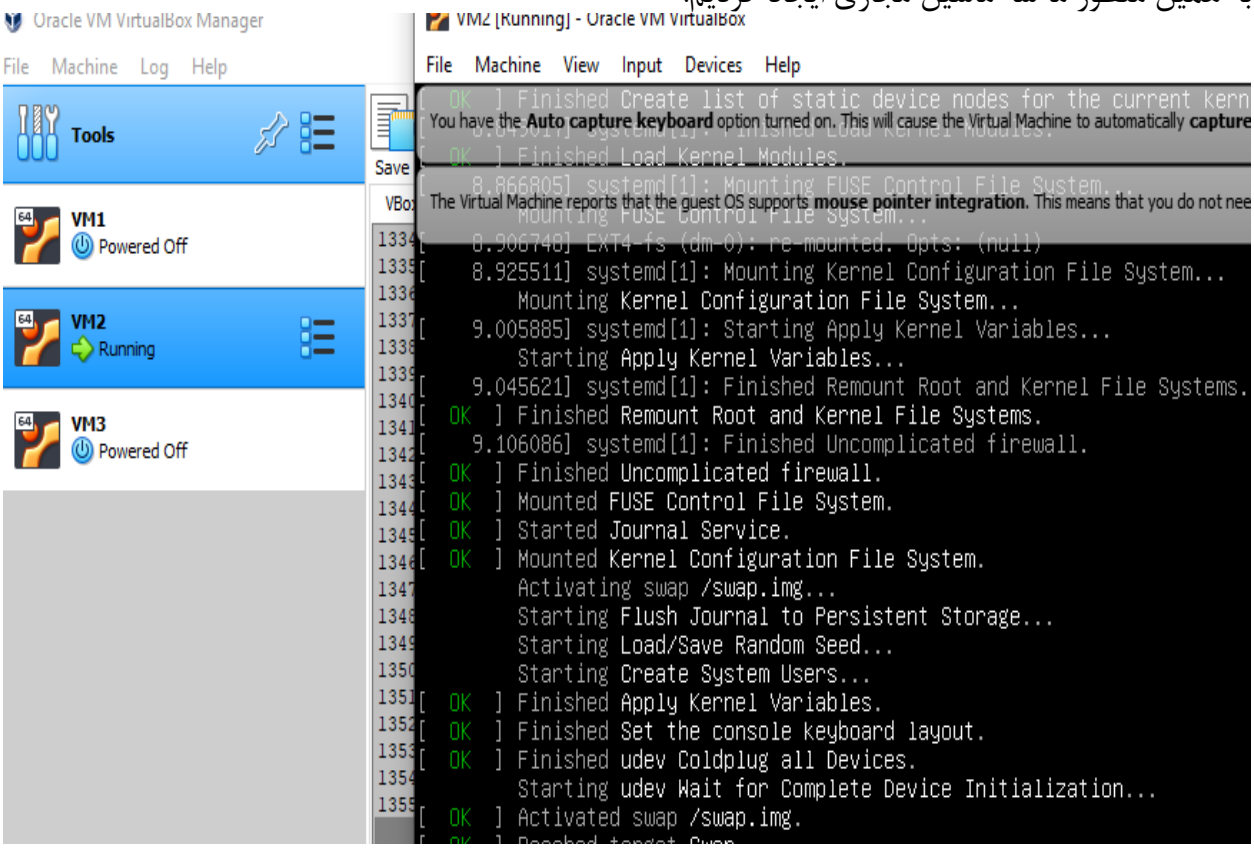
برای انجام این پردازش‌ها از ابزارِ هدوپ استفاده نمودیم. الگوریتمِ اجزا شده بر روی این بخش الگوریتمِ MapReduce است. شمای کلی این الگوریتم را در زیر مشاهده می‌کنید.



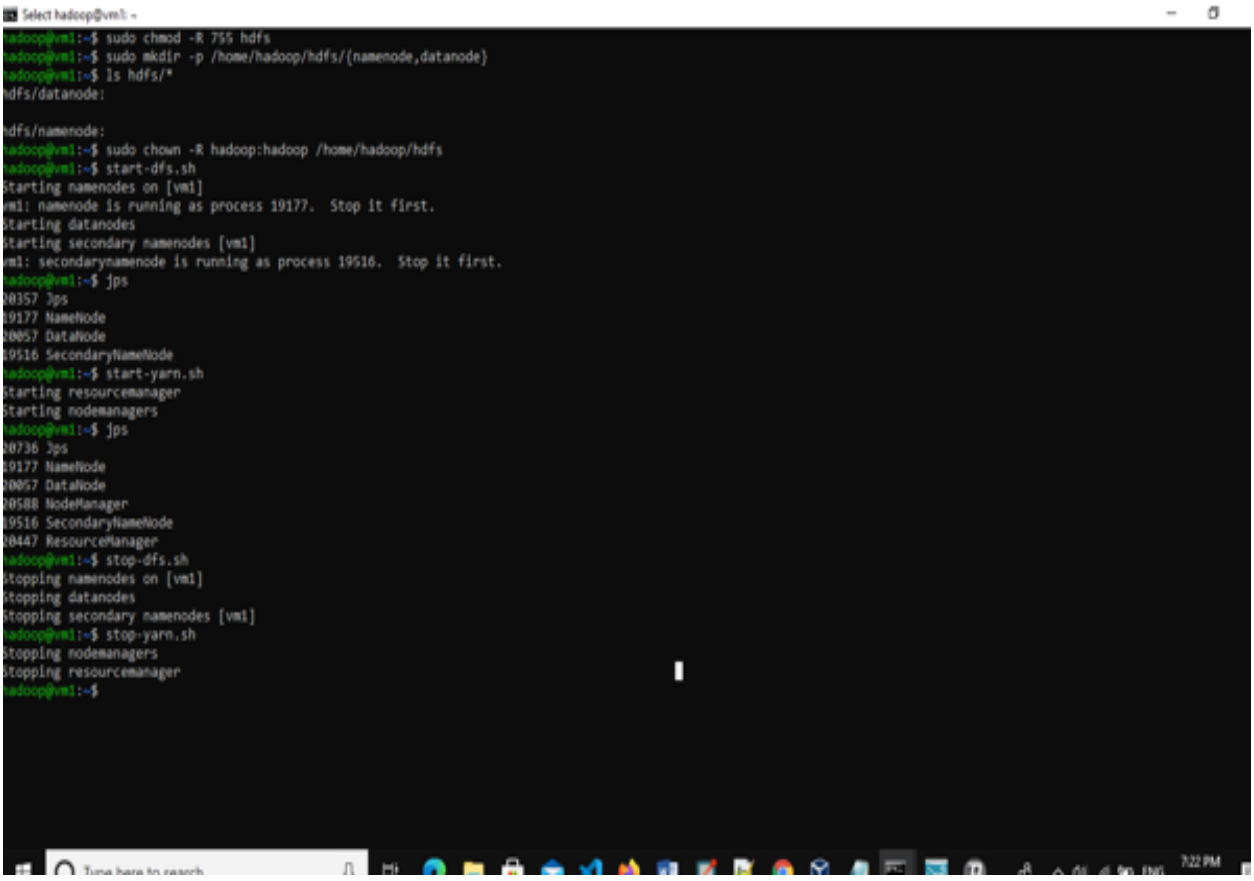
این سیستم عملیات ها را بدین صورت که مشاهده می‌کنید انجام می‌دهد. مثلا برای آموزشِ مدل از طریقِ دیتاها باید جملات را به گرام‌هایی که کلمات هستند تقسیم کرده و سپس کلمات را شمارش کرده و فرکانسِ آنها را بیابیم. این الگوریتم در ابتدا Splitting را انجام می‌دهد و جملاتِ موجود در Dataset را متناسب با خواسته‌ی ما تفکیک کرده و به گروه‌هایی تقسیم می‌کند و سپس جملات را با استفاده از عملیاتِ splitting به گرام‌ها تفکیک می‌نماید. حال در عملیاتِ Mapping به ازای حضورِ هر کلمه، یک واحد به آن اختصاص می‌دهد. سپس در عملیاتِ shuffling کلماتِ مشابه را در کنارِ یکدیگر قرار داده و نهایتا در عملیاتِ Reducing آن‌ها را با یکدیگر جمع زده و تعداد تکرارِ کلمات را با سرعت بالایی می‌یابد. حال که تعداد تکرار هر کلمه را یافت، آنها را به عنوانِ خروجی بر می‌گرداند. این نحوه‌ی عملِ سیستمِ فوق برای هر تسکی از جمله پردازش‌های NLP است که مثال آن را ذکر کردیم.

لازم است که ماشین‌های دیگری در خدمتِ این سیستم باشند و سیستم بتواند تسک‌ها را بینِ آنها تقسیم کرده و سپس خروجی‌های را گرفته و ترکیب کند.

به همین منظور ما سه ماشین مجازی ایجاد کردیم.



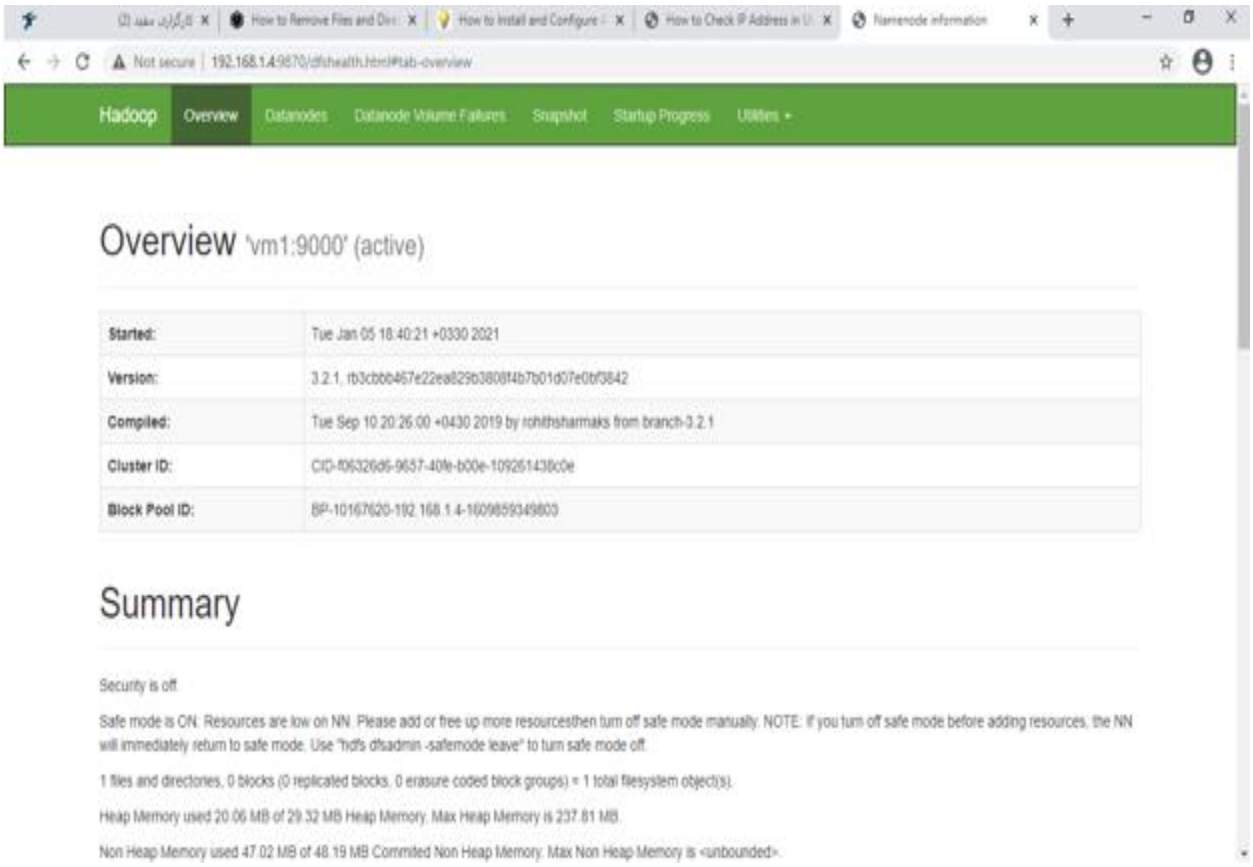
و پس از نصبِ این سیستم و قرار دادنِ الگوریتم لازم برای ترکیب در دایرکتوریِ آن، اقدام به start کردنِ yarn میکنیم تا نودها شروع به کار کنند.



همانطور که مشاهده می‌کنید تمامی نودها در حال فعالیت و در خدمتِ ماشینِ پردازنده‌ی NLP هستند. این ماشین‌ها در هر لحظه می‌توانند وضعیت خود را به اشتراک گذارند و سیستم نیز در این صورت هیچ اضافه باری به ماشینی نخواهد داد.

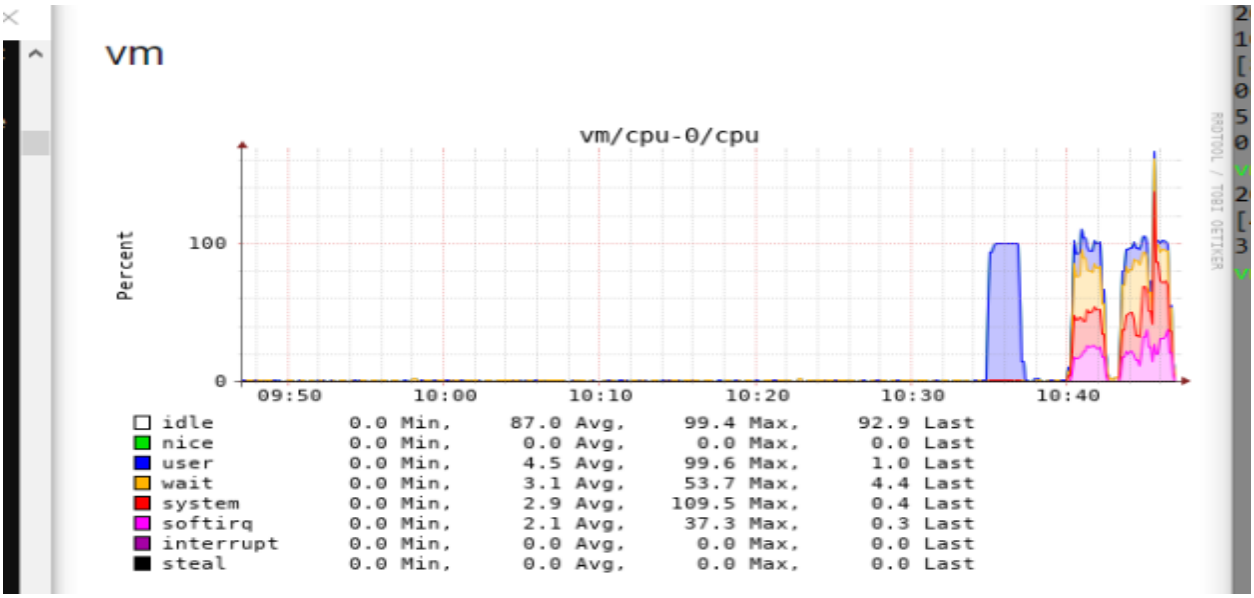
همچنین در صورتِ fail شدنِ یک تسک مثل یافتنِ تعداد تکرارِ کلمه در دیتاست، ماشین اصلی این تسک را مجددا به یک ماشین دیگر می‌دهد.

در شکل زیر می‌توانید دسترسی به هدوپ را از طریقِ ویندوز مشاهده کنید. این دسترسی پس از start خوردن و شروعِ فعالیت‌ها برقرار می‌شود و همانطور که مشاهده می‌کنید سیستم به درستی پردازش‌ها را انجام داده و خروجی‌های خود با فراخوانی url متناظرِ مربوط به ماشینِ اصلی نمایش می‌دهد.

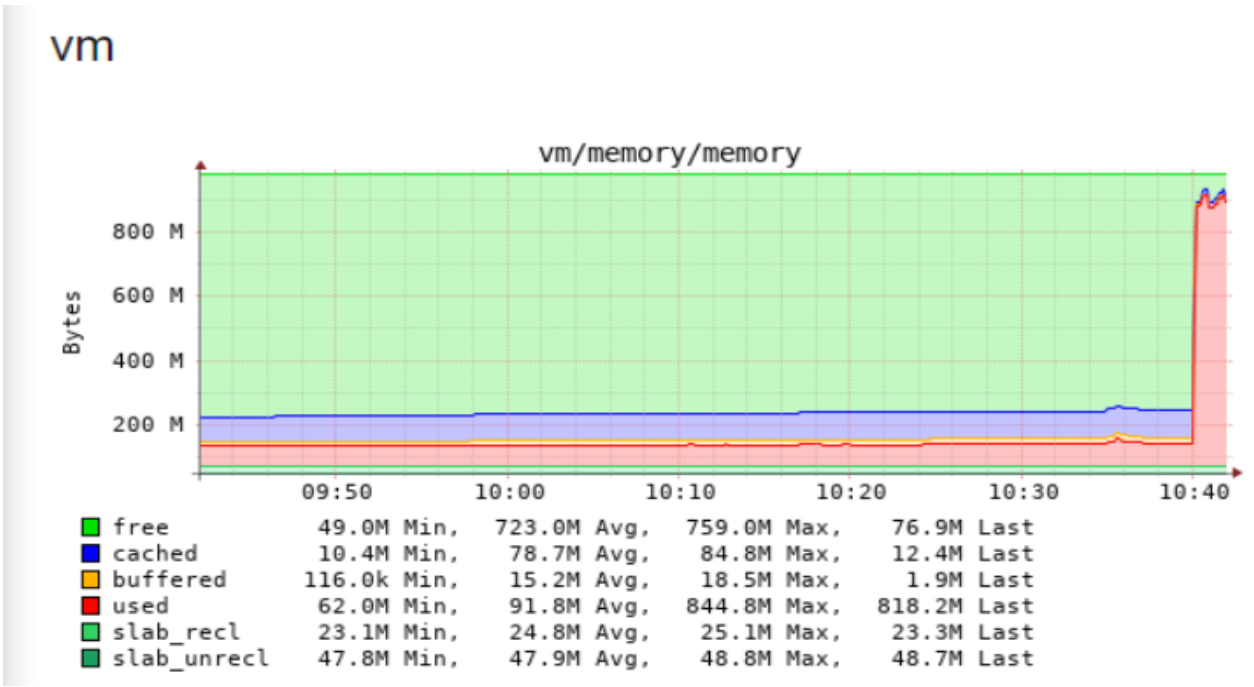


در نهایت طبق مقاله‌ای که ضمیمه نمودیم و از این سیستم نیز برای پردازش‌های NLP استفاده نموده، ما نیز عملیات‌ها را بر روی آن انجام دادیم و نتیجه‌ی زیر برای پردازش‌های cpu حاصل شد.

وضعیتِ cpu :

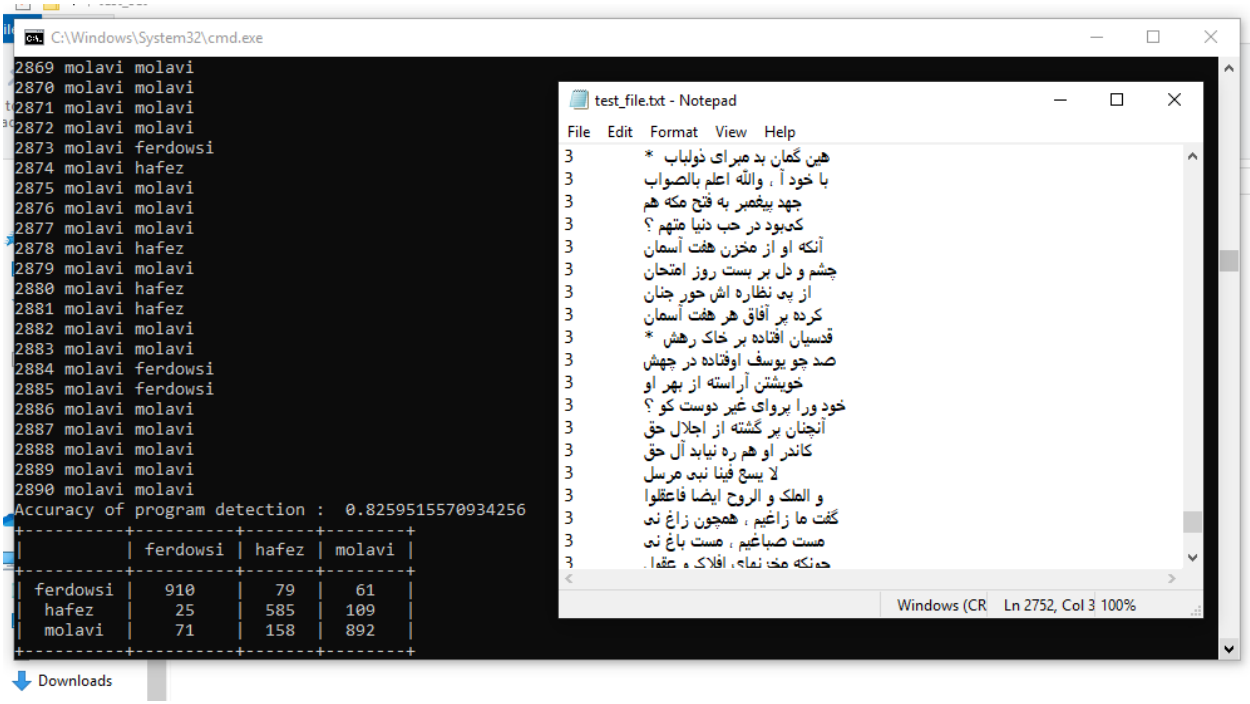


همچنین وضعیتِ memory نیز به صورت زیر در آمده است.



با توجه به میزان فضای کم و cpu کمی که vmها در اختیار دارند، وضعیت خوبی را مشاهده می‌کنیم و به نظر می‌رسد این سیستم با الگوریتم‌های ذکر شده می‌تواند یک سیستمِ کارا برای پردازش‌های سنگینِ NLP باشد.

در آخر نیز نمای کلیِ سیستمِ تشخیصِ شاعر از روی شعر را قرار داده‌ایم. همانطور که مشاهده می‌کنید این سیستم برای ID های مختلف که در واقع سطرهای فایلِ ورودی هستند، شاعر را تشخیص می‌دهد. در واقع ستونِ وسط، شاعر اصلی این شعر و ستونِ آخر شاعرِ پیش‌بینی شده نمایش داده می‌شود که با مراجعه به ستونِ اول و تطابق دادنِ ID با سطرِ موجود در فایل، مصراعی را که پیش‌بینی بر روی آن انجام گرفته است را با حداکثر صحتِ 82.5% (دقتِ مربوط به unigram model with naïve Bayes) مشاهده خواهید کرد.



پیشنهادات:

در این نرم‌افزار حداکثر دقتی که در بین تمامیِ الگوریتم‌ها پیاده‌سازی شده داستیم چیزی در حدود 82.5% بود. پیشنهاد من این است که از الگوریتم‌های learning استفاده کنیم و فیوژن را با وزن‌هایی به مدل‌ها اعمال کنیم، که جمله‌ی ورودی را به بهترین نحو تشخیص دهد. این وزن‌ها را باید از روی احتمالِ کلمات و نیز گرام‌های موجود در جمله‌ای را که به عنوانِ ورودی به سیستم داده می‌شود، بیابیم. همچنین از جمله برنامه‌های من بررسیِ

SVM نیز علاوه بر متدهای فوق بود و پیشنهاد می‌کند الگوریتمِ SVM نیز در کنار مقایساتی که انجام شده، مورد بررسی قرار گیرد.

مقالات

نرم‌افزارِ فوق در واقع پیاده‌سازیِ چهار رفرنسِ اصلی به همراه اضافه نمودنِ متدهای فیوژن به الگوریتم‌های این رفرنس‌ها می‌باشد.

- ✓ **A Comparative Study of Bing Web N-gram Language Models for Web Search and Natural Language Processing, Jianfeng Gao, Patrick Nguyen, Xiaolong Li, Chris Thrasher, Mu Li, Kuansan Wang**

این مقاله به پیاده‌سازیِ مدل‌های پایه‌ای ngram برای کاربردهای NLP پرداخته بود که ما دو مدل از مدل‌هایش را پیاده‌سازی نمودیم. و البته کلاسیفرهای و مالتی کلاسیفرهای جدیدی را اضافه کردیم.

- ✓ **Comparing Approaches to Convert Recurrent Neural Networks into Backoff Language Models For Efficient Decoding, Heike Adel, Katrin Kirchhoff, Ngoc Thang Vu, Dominic Telaar, Tanja Schultz**

این مقاله به پیاده‌سازی مدلِ backoff پرداخته بود و ما نیز این مدل را پیاده کردیم و الگوریتم ها ترکیبِ اطلاعات را روی آن اعمال نمودیم.

- ✓ **Managed N-gram Language Model Based on Hadoop Framework and a Hbase Tables, Tahani Mahmoud Allam, Hatem M. Abdullkader, Alsayed Abdelhameed Sallam**

این مقاله پردازش‌های NLP را بر روی بستر هدوپ انجام داده بود و کاراییِ هدوپ را به عنوان محیطی برای مدیریتِ big data مورد ارزیابی قرار داد که در پیاده‌سازیِ ما نیز این بخش مشهود است.

علاوه بر مقالاتِ فوق از مقالاتِ مربوط به پیاده‌سازی HDFS و الگوریتم MapReduce نیز استفاده کردیم که در مراجع موجود در درفتِ مقاله ذکر نموده‌ایم.