Реализация Наивного Байесовского классификатора. Мультиноминальная модель и модель Бернулли.

Иванов Р.А. 209 гр.

Теоретическая часть

Пункт a:

$$P(w_i|c_j) = \frac{1 + \sum_{d \in c_j} occurancesof w_i}{1 + \sum_{d \in c_j} allwords}$$

Пункт b:

$$P(d = (w_1, w_2, \dots, w_M)|d \in c_j) = \prod_{w_i \in d} P(w_i|c_j) \prod_{w_i \notin d} (1 - P(w_i|c_j))$$

Пункт c:

$$P(c_j|d) = \frac{P(d|c_j)P(c_j)}{P(d)} = \frac{\prod\limits_{w_i \in d} P(w_i|c_j) \prod\limits_{w_i \notin d} (1 - P(w_i|c_j))P(c_j)}{P(d)}$$

$$P(c_j) = \frac{cnt(c_j)}{|D_{train}|}$$

Пункт d:

$$c_k : k = argmax_j P(c_j|d) = argmax_j P(d|c_j)P(c_j) = argmax_j \prod_{w_i \in d} P(w_i|c_j) \prod_{w_i \notin d} (1 - P(w_i|c_j))P(c_j)$$

Практическая часть

Подготовка входных данных

Подключение необходимых библиотек:

In [1]:

```
from math import log
import pandas as pd   # -- используется только для красивого вывода таблиц в этом notebook
```

Загрузка обучающей выборки в 2 списка – позитивные (PosDocs) и негативные отзывы (NegDocs):

In [2]:

```
PosDocs = []
NegDocs = []
TrainT = "train.texts"
TrainL = "train.labels"
with open(TrainT, encoding='utf-8') as text:
    with open(TrainL, encoding='utf-8') as label:
        for line in text:
            lbl = label.readline()
            doc = line
            if lbl[:-1] == 'pos':
                PosDocs.append(doc)
            elif lbl[:-1] == 'neg':
                NegDocs.append(doc)
```

Минимальная, максимальная, средняя, медианная длина (в символах) позитивных / негативных отзывов:

In [3]:

```
def PrintLen(Docs):
    DataLen = 0
    MasLen = []
    for doc in Docs:
        LenDoc = len(doc)
        DataLen += LenDoc
        MasLen.append(LenDoc)
    MasLen.sort()
    m = len(MasLen)
    print('Наименьшая длина: ', MasLen[0])
    print('Наибольшая длина: ', MasLen[m - 1])
    print('Средняя длина: ', DataLen / m)
    if m % 2 == 0:
        print('Медианная длина: ', (MasLen[m // 2] + MasLen[m // 2 - 1]) / 2)
    else:
        print('Медианная длина: ', MasLen[m // 2])
```

In [4]:

```
print('pos:')
PrintLen(PosDocs)
print('neg:')
PrintLen(NegDocs)
```

```
pos:
Наименьшая длина:  71
Наибольшая длина:  10364
Средняя длина:  1361.8021276595746
Медианная длина:  997.5
neg:
Наименьшая длина:  53
Наибольшая длина:  8970
Средняя длина:  1317.3576203208556
Медианная длина:  982.0
```

Предобработки и токенизация:

In [5]:

```
def Preparing(doc):
    sym = '.,:;&()<>!?"-'
    for i in range(len(doc)):
        p = ''
        for j in range(len(doc[i])):
            if doc[i][j] in sym:
                doc.append(doc[i][j])
            else:
                p += doc[i][j]
        if p != '':
            doc[i] = p
    return doc
```

In [6]:

```
def CrList(text):
    Docs = []
    N = len(text)
    for sdoc in text:
        doc = sdoc
        doc = doc.lower()                  # -- перевод в нижний регистр
        doc = doc.replace('<br />', ' ')   # -- убираем символ <br />
        doc = doc.split()                  # -- разбиваем на слова
        doc = Preparing(doc)
        Docs.append(doc)

    return Docs
```

In [7]:

```
PosDocs = CrList(PosDocs)
NegDocs = CrList(NegDocs)
```

Создание 2-ух словарей "слово→частота" с частотами каждого слов в позитивных и негативных отзывах:

In [8]:

```
def CrDicts(PosDocs, NegDocs):
    PosSet = set()
    NegSet = set()
    for doc in PosDocs:
        PosSet = PosSet.union(set(doc))
    for doc in NegDocs:
        NegSet = NegSet.union(set(doc))
    # print(PosSet)
    PosDict = {wrd: 0 for wrd in PosSet}
    NegDict = {wrd: 0 for wrd in NegSet}

    return PosDict, NegDict
```

In [9]:

```
PosDict, NegDict = CrDicts(PosDocs, NegDocs)
```

TRAIN: Заполнение словарей вероятностями

In [10]:

```
def FindProb(DictW, Docs):
    LenAllWord = 0

    for doc in Docs:
        LenAllWord += len(doc)

    counter = 0
    LenDocs = len(Docs)
    for doc in Docs:
        for wrd in doc:
            DictW[wrd] += 1
        counter += 1
        #if counter % 100 == 0:
            #print(counter, '/', LenDocs)

    print(LenDocs, '/', LenDocs)
    for wrd in DictW:
        DictW[wrd] = DictW[wrd] / LenAllWord
```

In [11]:

```
FindProb(PosDict, PosDocs)
FindProb(NegDict, NegDocs)
```

```
7520 / 7520
7480 / 7480
```

Наивный Байесовский Классификатор Бернулли

In [12]:

```
PosProb = len(PosDocs) / (len(PosDocs) + len(NegDocs))
NegProb = 1 - PosProb
AllPos = 0
AllNeg = 0
for doc in PosDocs:
    AllPos += len(doc)
for doc in NegDocs:
    AllNeg += len(doc)
```

In [13]:

```
def Binom(PosDict, NegDict, doc):
    posver = log(PosProb)
    smas = set(doc)

    for wrd in PosDict:
        posver += log(1 - PosDict[wrd])
    for wrd in smas:
        if wrd in PosDict:
            posver = posver - log(1 - PosDict[wrd]) + log(PosDict[wrd])
        else:
            posver += log(1 / (len(PosDict) + AllPos))

    negver = log(NegProb)

    for wrd in NegDict:
        negver += log(1 - NegDict[wrd])
    for wrd in smas:
        if wrd in NegDict:
            negver = negver - log(1 - NegDict[wrd]) + log(NegDict[wrd])
        else:
            negver += log(1 / (len(NegDict) + AllNeg))

    if negver > posver:
        return 'neg'
    else:
        return 'pos'
```

In [14]:

```
%%time
devtxt = "dev.texts"
devlbl = "dev.labels"
amount = 0
accur = 0
N = 0

with open(devtxt, encoding='utf-8') as text:
    for line in text:
        N += 1

print('N =',N)

with open(devtxt, encoding='utf-8') as text:
    with open(devlbl, encoding='utf-8') as label:
        for line in text:
            res = str()
            amount += 1
            num = label.readline()
            TMP = line
            TMP = TMP.replace('<br />', ' ')
            TMP = TMP.lower()
            TMP = TMP.split()
            TMP = Preparing(TMP)
            res = Binom(PosDict, NegDict, TMP)
            if num[:-1] == res:
                accur += 1
            if amount%2000 == 0:
                print(amount,'/',N)

print(accur/amount)
```

```
N = 10000
2000 / 10000
4000 / 10000
6000 / 10000
8000 / 10000
10000 / 10000
0.8552
Wall time: 7min 52s
```

Наивный Байесовский Мультиномиальный Классификатор

In [15]:

```
PosProb = len(PosDocs) / (len(PosDocs) + len(NegDocs))
NegProb = 1 - PosProb
AllPos = 0
AllNeg = 0
for doc in PosDocs:
    AllPos += len(doc)
for doc in NegDocs:
    AllNeg += len(doc)
```

In [16]:

```
def Multi(PosDict, NegDict, doc):
    posver = log(PosProb)
    for wrd in doc:
        if wrd in PosDict:
            posver += log(PosDict[wrd])
        else:
            posver += log(1 / (1 + AllPos))
    negver = log(NegProb)
    for wrd in doc:
        if wrd in NegDict:
            negver += log(NegDict[wrd])
        else:
            negver += log(1 / (1 + AllNeg))
    if negver > posver:
        return 'neg'
    else:
        return 'pos'
```

In [17]:

```
%%time
devtxt = "dev.texts"
devlbl = "dev.labels"
amount = 0
accur = 0
N = 0

with open(devtxt, encoding='utf-8') as text:
    for line in text:
        N += 1

print('N =',N)

with open(devtxt, encoding='utf-8') as text:
    with open(devlbl, encoding='utf-8') as label:
        for line in text:
            res = str()
            amount += 1
            num = label.readline()
            TMP = line
            TMP = TMP.replace('<br />', ' ')
            TMP = TMP.lower()
            TMP = TMP.split()
            TMP = Preparing(TMP)
            res = Multi(PosDict, NegDict, TMP)
            if num[:-1] == res:
                accur += 1
            #if amount%100 == 0:
                #print(amount,'/',10000)

text.close()
label.close()
print(accur/amount)
```

```
N = 10000
0.8456
Wall time: 4.9 s
```

Вывод: Мультиномиальный классификатор работает быстрее, но с меньшей точность, чем классификатор Бернулли.

Построение таблиц вероятностей

In [18]:

```
dictofpos = PosDict
dictofneg = NegDict
```

In [19]:

```
tabledata = []
for i in range(30):
    tabledata.append([''] * 2)

sorted_dict_pos = {}
sorted_keys_pos = sorted(PosDict, key=PosDict.get)

for w in sorted_keys_pos:
    sorted_dict_pos[w] = PosDict[w]

sorted_dict_neg = {}
sorted_keys_neg = sorted(NegDict, key=NegDict.get)

for w in sorted_keys_neg:
    sorted_dict_neg[w] = NegDict[w]

X = list(sorted_dict_pos)
Y = list(sorted_dict_neg)
```

Самые частые из pos

In [20]:

```
for i in range(30):
    tabledata[i][0] = X[-i-1]
    tabledata[i][1] = str(sorted_dict_pos[tabledata[i][0]])


pd.DataFrame(tabledata, columns=["Word", "Probability"])
```

Out[20]:

	Word	Probability
0	the	0.05113703074610977
1	.	0.047559410125551836
2	,	0.04284143210495006
3	and	0.02637155984846113
4	a	0.024670563417765114
5	of	0.02268729061648768
6	to	0.01974466952628448
7	is	0.01701338880365697
8	in	0.014866718588559442
9	it	0.011518050032875168
10	-	0.011136463258085726
11	i	0.010500485300103323
12	this	0.010221144212404897
13	that	0.010116452456244717
14	"	0.009369912176336141
15	as	0.007876342402705157
16	with	0.0068651374495131345
17	for	0.006664559785841761
18	was	0.0064879535990481855
19	but	0.0060457043113434986
20	film	0.0059390556999279875
21	)	0.005671944957575378
22	(	0.005632807852468768
23	movie	0.005539368014026738
24	his	0.005194961489088575
25	on	0.0049777505557468925
26	he	0.00444108300197251
27	are	0.00440732724881806
28	you	0.004276217946710918
29	not	0.004184245749710385

Самые редкие из pos

In [21]:

```
for i in range(30):
    tabledata[i][0] = X[i]
    tabledata[i][1] = str(sorted_dict_pos[tabledata[i][0]])


pd.DataFrame(tabledata, columns=["Word", "Probability"])
```

Out[21]:

	Word	Probability
0	pistilli	4.892138138326184e-07
1	evaporation	4.892138138326184e-07
2	opposes	4.892138138326184e-07
3	notsowelladjusted	4.892138138326184e-07
4	italy/canada	4.892138138326184e-07
5	truebut	4.892138138326184e-07
6	olsen's	4.892138138326184e-07
7	outtherefunny	4.892138138326184e-07
8	veronikathen	4.892138138326184e-07
9	395	4.892138138326184e-07
10	paradox	4.892138138326184e-07
11	"peeping	4.892138138326184e-07
12	dandelions	4.892138138326184e-07
13	waffles	4.892138138326184e-07
14	newsweek	4.892138138326184e-07
15	missionaries	4.892138138326184e-07
16	year´s	4.892138138326184e-07
17	mrdalton	4.892138138326184e-07
18	hoggish	4.892138138326184e-07
19	'deliverance'	4.892138138326184e-07
20	despotovich	4.892138138326184e-07
21	bettany's	4.892138138326184e-07
22	dawid	4.892138138326184e-07
23	austens	4.892138138326184e-07
24	outofnowhere	4.892138138326184e-07
25	'defeat'	4.892138138326184e-07
26	factoring	4.892138138326184e-07
27	rv	4.892138138326184e-07
28	soundalike	4.892138138326184e-07
29	kayeas	4.892138138326184e-07

Самые частые из neg

In [22]:

```
for i in range(30):
    tabledata[i][0] = Y[-i-1]
    tabledata[i][1] = str(sorted_dict_neg[tabledata[i][0]])


pd.DataFrame(tabledata, columns=["Word", "Probability"])
```

Out[22]:

	Word	Probability
0	.	0.05112460456738805
1	the	0.04943303521832304
2	,	0.04024276156613684
3	a	0.023929296925465884
4	and	0.02236289868127968
5	of	0.02091362841876722
6	to	0.020849786128327025
7	is	0.015204519028772765
8	in	0.013192230141905828
9	this	0.012246660470110497
10	-	0.012056138989190545
11	i	0.0119907886131494
12	it	0.011808812950634828
13	that	0.010657640941910051
14	"	0.01054151835063694
15	was	0.007888795778566786
16	movie	0.007214178819742047
17	for	0.0065214648337059145
18	but	0.006461644104868251
19	with	0.0063409972567922916
20	as	0.006216831542314117
21	film	0.005598516445924826
22	)	0.005241602853700113
23	(	0.005099340112010545
24	on	0.005088280817603582
25	not	0.004909321326290909
26	have	0.0045252621932491055
27	you	0.004506662470837395
28	be	0.004373448242753523
29	are	0.004364399729147826

Самые редкие из neg

In [23]:

```
for i in range(30): #самые редкие из neg
    tabledata[i][0] = Y[i]
    tabledata[i][1] = str(sorted_dict_neg[tabledata[i][0]])


pd.DataFrame(tabledata, columns=["Word", "Probability"])
```

Out[23]:

	Word	Probability
0	pointsome	5.026952003164969e-07
1	markbut	5.026952003164969e-07
2	aidsinfected	5.026952003164969e-07
3	fetishand	5.026952003164969e-07
4	brattiest	5.026952003164969e-07
5	bleibtreau	5.026952003164969e-07
6	truebut	5.026952003164969e-07
7	no2	5.026952003164969e-07
8	divas	5.026952003164969e-07
9	inscriptions	5.026952003164969e-07
10	60/10	5.026952003164969e-07
11	umbilical	5.026952003164969e-07
12	neurosurgery	5.026952003164969e-07
13	missionaries	5.026952003164969e-07
14	walkthru	5.026952003164969e-07
15	vulcan	5.026952003164969e-07
16	maniacturnedjokester	5.026952003164969e-07
17	butand	5.026952003164969e-07
18	1430	5.026952003164969e-07
19	congruity	5.026952003164969e-07
20	punksters	5.026952003164969e-07
21	outofnowhere	5.026952003164969e-07
22	co2	5.026952003164969e-07
23	credentialed	5.026952003164969e-07
24	idea/intentions	5.026952003164969e-07
25	neff's	5.026952003164969e-07
26	pollyanna	5.026952003164969e-07
27	smooch	5.026952003164969e-07
28	ignoramus	5.026952003164969e-07
29	bobbing	5.026952003164969e-07

Таблица с баесовскими весами

In [24]:

```
Bweignt = {}
for wrd in PosDict:
    if wrd in NegDict:
        c = PosDict[wrd]/NegDict[wrd]
        c = log(c)
        Bweignt[wrd] = c

sorted_Bweignt = {}
sorted_keys_Bweignt = sorted(Bweignt, key=Bweignt.get)

for w in sorted_keys_Bweignt:
    sorted_Bweignt[w] = Bweignt[w]

tabledata = []
for i in range(30):
    tabledata.append([''] * 4)

X = list(sorted_Bweignt)

for i in range(30):
    tabledata[i][0] = X[-i-1]
    tabledata[i][1] = str(sorted_Bweignt[tabledata[i][0]])
    tabledata[i][2] = PosDict[tabledata[i][0]]
    tabledata[i][3] = NegDict[tabledata[i][0]]


pd.DataFrame(tabledata, columns=["Word", "Weight", "Positive probability", "Negative probab
```

Out[24]:

	Word	Weight	Positive probability	Negative probability
0	sox	3.980148803400044	0.000027	5.026952e-07
1	kolchak	3.7794781079378925	0.000022	5.026952e-07
2	adele	3.757005252085834	0.000022	5.026952e-07
3	7/10	3.6944848951045	0.000061	1.508086e-06
4	mathieu	3.636377264297219	0.000019	5.026952e-07
5	daylewis	3.636377264297219	0.000019	5.026952e-07
6	chavez	3.4991761427837345	0.000017	5.026952e-07
7	clara	3.469323179634053	0.000016	5.026952e-07
8	tony's	3.4385515209672994	0.000016	5.026952e-07
9	firstrate	3.4385515209672994	0.000016	5.026952e-07
10	philo	3.406802822652719	0.000015	5.026952e-07
11	astaire	3.390542301780939	0.000030	1.005390e-06
12	kelly's	3.3740129998297284	0.000015	5.026952e-07
13	delightfully	3.3050201283427767	0.000014	5.026952e-07
14	anton	3.3050201283427767	0.000014	5.026952e-07
15	vertigo	3.3050201283427767	0.000014	5.026952e-07
16	haines	3.3050201283427767	0.000014	5.026952e-07

	Word	Weight	Positive probability	Negative probability
17	stardust	3.268652484171902	0.000013	5.026952e-07
18	giovanna	3.268652484171902	0.000013	5.026952e-07
19	bourne	3.2686524841719016	0.000040	1.508086e-06
20	anchors	3.230912156189055	0.000013	5.026952e-07
21	lindy	3.230912156189055	0.000013	5.026952e-07
22	luzhin	3.1916914430357735	0.000012	5.026952e-07
23	cheung	3.1508694485155186	0.000012	5.026952e-07
24	chan's	3.1508694485155186	0.000012	5.026952e-07
25	melbourne	3.1508694485155186	0.000012	5.026952e-07
26	8/10	3.142501198845002	0.000058	2.513476e-06
27	georges	3.1083098340967226	0.000023	1.005390e-06
28	itchy	3.1083098340967226	0.000011	5.026952e-07
29	sinatra's	3.063858071525889	0.000011	5.026952e-07

Биграммы

Создание биграмм:

In [25]:

```
def Bigr(docs):
    doc = []
    sym = '.,:;&()<>!?"-'
    for A in docs:
        TMP = []
        tmpstr = ''
        for i in range(1, len(A)):
            if A[i] not in sym:
                tmpstr = A[i-1] + ' ' + A[i]
                TMP.append(tmpstr)
        doc.append(TMP)
    doc += docs
    return doc
```

In [26]:

```
PosDocs = Bigr(PosDocs)
NegDocs = Bigr(NegDocs)
```

Создание 2-ух словарей "слово→частота" с частотами каждого слов в позитивных и негативных отзывах:

In [29]:

```
def CrDicts(PosDocs, NegDocs):
    PosSet = set()
    NegSet = set()
    counter = 0
    for doc in PosDocs:
        PosSet = PosSet.union(set(doc))
        counter += 1
        if counter % 2000 == 0:
            print(counter,'/',len(PosDocs))
    counter = 0
    for doc in NegDocs:
        NegSet = NegSet.union(set(doc))
        counter += 1
        if counter % 2000 == 0:
            print(counter,'/',len(PosDocs))
    # print(PosSet)
    PosDict = {wrd: 0 for wrd in PosSet}
    NegDict = {wrd: 0 for wrd in NegSet}

    return PosDict, NegDict
```

In [30]:

```
PosDict, NegDict = CrDicts(PosDocs, NegDocs)
```

```
2000 / 15040
4000 / 15040
6000 / 15040
8000 / 15040
10000 / 15040
12000 / 15040
14000 / 15040
2000 / 15040
4000 / 15040
6000 / 15040
8000 / 15040
10000 / 15040
12000 / 15040
14000 / 15040
```

TRAIN: Заполнение словарей вероятностями

In [31]:

```
def FindProb(DictW, Docs):
    LenAllWord = 0

    for doc in Docs:
        LenAllWord += len(doc)

    counter = 0
    LenDocs = len(Docs)
    for doc in Docs:
        for wrd in doc:
            DictW[wrd] += 1
        counter += 1
        #if counter % 100 == 0:
            #print(counter, '/', LenDocs)

    print(LenDocs, '/', LenDocs)
    for wrd in DictW:
        DictW[wrd] = DictW[wrd] / LenAllWord
```

In [32]:

```
FindProb(PosDict, PosDocs)
FindProb(NegDict, NegDocs)
```

```
15040 / 15040
14960 / 14960
```

Наивный Байесовский Мультиномиальный Классификатор

In [33]:

```
PosProb = len(PosDocs) / (len(PosDocs) + len(NegDocs))
NegProb = 1 - PosProb
AllPos = 0
AllNeg = 0
for doc in PosDocs:
    AllPos += len(doc)
for doc in NegDocs:
    AllNeg += len(doc)
```

In [34]:

```
def Multi(PosDict, NegDict, doc):
    posver = log(PosProb)
    for wrd in doc:
        if wrd in PosDict:
            posver += log(PosDict[wrd])
        else:
            posver += log(1 / (1 + AllPos))
    negver = log(NegProb)
    for wrd in doc:
        if wrd in NegDict:
            negver += log(NegDict[wrd])
        else:
            negver += log(1 / (1 + AllNeg))
    if negver > posver:
        return 'neg'
    else:
        return 'pos'
```

In [35]:

```
%%time
devtxt = "dev.texts"
devlbl = "dev.labels"
amount = 0
accur = 0
N = 0

with open(devtxt, encoding='utf-8') as text:
    for line in text:
        N += 1

print('N =',N)

with open(devtxt, encoding='utf-8') as text:
    with open(devlbl, encoding='utf-8') as label:
        for line in text:
            res = str()
            amount += 1
            num = label.readline()
            TMP = line
            TMP = TMP.replace('<br />', ' ')
            TMP = TMP.lower()
            TMP = TMP.split()
            TMP = Preparing(TMP)
            res = Multi(PosDict, NegDict, TMP)
            if num[:-1] == res:
                accur += 1
            #if amount%100 == 0:
                #print(amount,'/',10000)

text.close()
label.close()
print(accur/amount)
```

```
N = 10000
0.8441
Wall time: 5.31 s
```

Вывод: на практике оказалось, что классификатор, работающий с биграммами, показывает результат чуть хуже, нежели работающий только со словами.

In [ ]:

```
```