Инженерный вестник Дона, №11 (2024)
ivdon.ru/ru/magazine/archive/n11y2024/9603

Определение авторства литературных произведений с помощью нейросетей

Л.А. Метелькова, А.Н. Сак

Московский государственный строительный университет

Аннотация: В данной статье рассматриваются статистические методы, а также методы машинного обучения для выбора оптимального способа установить авторство по отрывку произведения. Создается датасет из отрывков соответствующих авторов, а также набор численных признаков, соответствующий каждому отрывку, применяются различные подходы для анализа авторства, такие как корреляция, сходство, t-test. Предпринимается попытка найти оптимальный метод для выходного слоя графовой сверточной нейросети, используемой для предварительной обработки данных. Осуществляется тренировка нейросети.
Ключевые слова: t-test, косинусное сходство, корреляция, графовые сверточные нейросети, анализ естественного языка

Введение

Очень часто возникает такая ситуация - вспоминаются знакомые строчки, но очень трудно вспомнить конкретного автора, хотя на ум приходят пара десятков поэтов, которые могли бы это написать. Бывает и так, что можно лишь приблизительно вспомнить слова строки стихотворения. Задачей данного исследования является выяснить, какой метод машинного обучения или статистического анализа наиболее эффективен для установления авторства произведения по его отдельному фрагменту или строке, схожей с оригиналом. Для того, чтобы понять, перу какого из выбранных поэтов принадлежит данная строка, мы должны создать датасет из строк их произведений. Данный датасет позволит применить методы машинного обучения для реализации обучения с “учителем”, а также оценить эффективность разных классификаторов с помощью метрики “accuracy” - соотношения правильно угаданного авторства строк к общему количеству вариантов. Accuracy=correct_predictions / total_predictions.

Сначала необходимо сформировать датасет из произведений поэтов, объединив их в один корпус. Для этого используем словарь.

```
R=[]
for k,v in F.items():
    R.append({'author':k,'text':v})
data=pd.DataFrame(R)
data.text=data.text.apply(lambda x: x.replace('.','').replace(',','').replace(';','').replace('-','').replace(':','').lower())
#data.text=data.text.apply(lambda x: x.replace(',','').replace(';','').replace('-','').replace(':','').lower())
```

```
data.head(len(data))
```

	author	text
0	boris vian	il s'est levé à mon approche debout il éta...
1	georges brassens	la veuve et l' orphelin quoi de plus émouva...
2	pierre perret	je suis vert vert vert je suis vert de co...
3	yannick noah	le ciment dans les plaines coule jusqu'aux...
4	renaud	qu'est ce que tu fais plantée petite fille ...
5	oldelaf	antoine a refusé ce soir de prendre un verre...
6	jean ferrat	la tristitude c'est tes plaies anticonstitut...
7	jacques dutronc	c'était un petit jardin qui sentait bon le...
8	juliette noureddine	il est 20 heures j'attends qu'il meure notr...
9	céliane	tu n'avais que vingt ans le regard amoureux ...
10	joe dassin	avec son marteau piqueur il creuse le sill...
11	bernard lavilliers	venues des hauts plateaux incendiées par ...
12	aznavour	depuis qu'avec l'homme sur terre elle fut mi...
13	denis lefdup	les réseaux sociau x qu'est ce qu'on ferai...

Рис.1.- Получение тренировочного датасета.

Затем разделяем произведения на контрольные фразы размером 100 элементов. В графе «автор» записываем соответствующего автора.

Creating a 100 elements control sequence

```
WWW=[]
AUT=[]
W=[]
for i in range(len(data)):
    str1=data.text.iloc[i]
    j=0
    p=0

    while j<len(str1):
        p=j
        if j<len(str1)-100:
            j+=100
            W.append({'author': data.author.iloc[i],'text':str1[p:j]})
        else:
            W.append({'author': data.author.iloc[i],'text':str1[p:]})
            break


data1=pd.DataFrame(W)
data1.tail()
```

14]:

	author	text
460	denis lefdup	out ce qui se dit il y en a qui se prennent l...
461	denis lefdup	ffit pour faire la fête d'appuyer sur le bouto...
462	denis lefdup	dra le jour de la grande panne si on veut enco...
463	denis lefdup	afin de réapprendre à marcher pour nous réapp...
464	denis lefdup	

Рис.2.- Разделение текста каждого автора на отрывки из 100 слов

После этого с помощью упрощенной версии метода TF-IDF [1] векторизируем каждую последовательность на основании частоты его употреблений во всем корпусе.

```
A={}
for i in range(len(data1)):
    for j in data1.text.iloc[i].split(' '):
        A[j]=A.get(j,0)+1

print(max(A,key=A.get))
len(A)
```

```
2516
```

```
B={}
w=sum(A.values())
for k,v in A.items():
    B[k]=v/w*100
```

Рис.3 Векторизация предложений

```
len(data1)
data1['vector']=data1['text'].apply(lambda x: [float(B.get(word,0)) for word in x.split(' ') if B.get(word,0)!=0 and B.get(word,0)<1])
```

Рис. 4.- Признаки со значениями <0 или >1 отбрасываются

В колонку vector попадают значения не равные 0 и меньше 1. Data1 будет выполнять роль набора контрольных фраз, которые мы будем тестировать, записывая ответы наших классификаторов и выбирая лучший из них. Принимая во внимание то, что распределение в каждом отрывке data1 не будет нормальным, мы приняли решение не бороться с выбросами, а решить эту проблему другим способом на более позднем этапе.

```
data1.head()
```

	author	text	vector
0	boris vian	il s'est levé à mon approche debout il éta...	[0.8356004663816555, 0.04858142246404975, 0.00...
1	boris vian	mignon là c'est pour mon lit! il m'arrivait...	[0.00971628449280995, 0.08744656043528955, 0.9...
2	boris vian	a suivie jusqu'à ma piaule et j'ai crié vas y ...	[0.6898561989895065, 0.00971628449280995, 0.05...
3	boris vian	moi au ciel zoum! faismoi mal johnny johnny...	[0.26233968130586865, 0.44694908666925764, 0.1...
4	boris vian	plus que ses chaussettes des belles jaunes a...	[0.582977069568597, 0.961912164788185, 0.14574...

Рис.5.- Текст и его численное представление

Теперь создаем датасет, векторы которого будут иметь относительно небольшой размер и будут использоваться для сравнения с контрольными примерами data1.

```
WWW=[]
AUT=[]
W=[]
for i in range(len(data)):
    str1=data.vector.iloc[i]
    p=0
    for j in range(25,len(str1),25):
        W.append({'author': data.author.iloc[i],'vector':str1[p:j]})
        p=j


data111=pd.DataFrame(W)
data111.head()
```

	author	vector
0	boris vian	[0.8356004663816555, 0.0485814224640497 5, 0.00...
1	boris vian	[0.00971628449280995, 0.0582977069568597, 0.00...
2	boris vian	[0.22347454333462882, 0.22347454333462882, 0.0...
3	boris vian	[0.0194325689856199, 0.02914885347842985, 0.23...
4	boris vian	[0.7481539059463661, 0.4372328021764477, 0.019...

Рис.6.- Формирование нового датасета из авторов и отрывков их произведений в виде векторов [2]

Теперь можно сравнивать модели. Первой моделью у нас будет простая корреляция [3] между векторами из data1 и контрольными векторами data111.

```
def corr(x,y):
    if len(x)>len(y):
        x=x[:len(y)]
    else:
        y=y[:len(x)]
    return np.corrcoef(x,y)[0,1]
```

```
A1=[]
B1=[]
C1=[]
for i in range(len(data1)):
    data111['corr']=data111['vector'].apply(lambda x:corr(x,data1['vector'].iloc[i]))
    data6=data111.groupby('author')[['corr']].sum()
    data7=data6.sort_values(by=['corr'],ascending=False)
    A1.append(data7.index[0])
    B1.append(data7.index[1])
    C1.append(data7.index[2])
data1['author1']=A1
data1['author2']=B1
data1['author3']=C1
```

Рис. 7.- Получение значений корреляции между контрольным предложением из data1 и тренировочным датасетом из data111

Мы заполняем колонки ‘author1’, ‘author2’, ‘author3’ авторами, которые показывают 1-ый, 2-ой и 3-ий результат соответственно. Не логарифмируя векторы, получаем следующее значение метрики accuracy:

```
ta1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])]))/(len(data1
```

```
0.27741935483870966
```

Рис.8. - Значение метрики accuracy для трех авторов сразу

Значение accuracy=0.28 - не очень высокий результат. А данное значение для одного автора гораздо хуже:

```
len(data1[data1['author']==data1['author1']])/len(data1)
```

```
0.12473118279569892
```

Теперь используем метод косинусного сходства [4]:

```
import math
def cosine_sim(vec1,vec2):
    dot_prod=0
    for i,v in enumerate(vec1):
        dot_prod+=v*vec2[i]
        mag_1=math.sqrt(sum([x**2 for x in vec1]))
        mag_2=math.sqrt(sum([x**2 for x in vec2]))
        return dot_prod/(mag_1*mag_2)
```

```
A1=[]
B1=[]
C1=[]
R1=[]
for i in range(len(data1)):
    data111['cos']=data111['vector'].apply(lambda x: abs(cosine_sim(x[:len(data1['vector'].iloc[i])],data1['vector'].iloc[i])
    data6=data111.groupby('author')[['cos']].sum()
    data7=data6.sort_values(by=['cos'],ascending=False)
    A1.append(data7.index[0])
    B1.append(data7.index[1])
    C1.append(data7.index[2])
    j=np.argmax(data111['cos'])
    R1.append(data111['author'].iloc[j])
data1['author1']=A1
data1['author2']=B1
data1['author3']=C1
data1['author10']=R1
```

Рис. 9. – Использование метода косинусного сходства для определения трех вероятных кандидатов на авторство

И получаем следующие результаты:

```
len(data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])])/len(
#print(len(data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])
```

```
0.40948275862068967
```

```
len(data1[data1['author']==data1['author3']])/len(data1)
```

```
0.14224137931034483
```

Рис. 10. – Результаты использования метода косинусного сходства

Как мы видим, результаты лучше, чем у ковариации, особенно для троих кандидатов, но значение метрики “accuracy” для единственного победителя очень низкая. Еще одним недостатком данных методов является необходимость задания одинаковой размерности у векторов.

Как работает наш классификатор? В цикле пробегаем data1 и записываем результаты сравнения в колонку [‘cos’] или [‘corr’] в data111. Затем создаем новый датасет data6, группируя по авторам суммы значений колонки с результатом. Затем сортируем по убыванию данные значения и в data1 для каждого контрольного предложения записываем автора с самым высоким значением в колонку ‘author1’, со вторым по величине значением в ‘author2’ и ‘author3’ соответственно.

```
data1.head(10)
```

	author	text	vector	author1	author2	author3	author10
0	boris vian	il s'est levé à mon approche debout il éta...	[0.8356004663816555, 0.0485814224640497 5, 0.00...	renaud	yannick noah	juliette noureddine	renaud
1	boris vian	mignon là c'est pour mon lit! il m'arrivait...	[0.00971628449280995, 0.08744656043528955, 0.9...	renaud	yannick noah	juliette noureddine	renaud
2	boris vian	a suivie jusqu'à ma piaule et j'ai crié vas y ...	[0.6898561989895065, 0.00971628449280995, 0.05...	renaud	yannick noah	juliette noureddine	renaud
3	boris vian	moi au ciel zoum! faismoi mal johnny johnny...	[0.26233968130586865, 0.4469490866692576 4, 0.1...	renaud	yannick noah	pierre perret	renaud
4	boris vian	plus que ses chaussettes des belles jaunes a...	[0.582977069568597, 0.961912164788185, 0.14574...	renaud	yannick noah	juliette noureddine	renaud
5	boris vian	comprenait rien le malheureux et il m'a dit l...	[0.00971628449280995, 0.3400699572483482, 0.00...	renaud	pierre perret	yannick noah	renaud
6	boris vian	m'énervait je l'ai giflé et j'ai grincé d'un a...	[0.00971628449280995, 0.0582977069568597, 0.00...	renaud	yannick noah	pierre perret	renaud
7	boris vian	e suis pas une mouche zoum! faismoi mal johnn...	[0.2331908278274388, 0.194325689856199, 0.9424...	renaud	yannick noah	juliette noureddine	renaud

Рис. 11.- Иллюстрация работы классификатора

Как мы видим, для векторной схожести мало совпадений. Теперь попробуем использовать этот же принцип, но для реализации t-теста. T-тест или t-критерий основывается на сравнении соотношения среднего значения выборки к стандартному отклонению и подразумевает две гипотезы:

H0: Распределение в двух векторах не отличается, т.е. два предложения принадлежат одному автору)

H1: Распределение двух векторов значительно отличается, т.е. два предложения принадлежат разным авторам.

Известно, что если p-value <0.05 , значит, можно опровергать гипотезу H0, а если превышает, значит нет оснований ее отвергать.

В Питоне есть две функции для t-критерия: stats.ttest_ind(a=a1,b=b1,equal_var=True) для нормального распределения и stats.mannwhitneyu(a1,b1,alternative='two-sided') для ненормального. Сначала подадим в функцию необработанные значения.

```
from scipy import stats
A1=[]
B1=[]
C1=[]
t=[]
T=[]
T1=[]
T2=[]
T3=[]
for i in range(len(data1)):
    data111['t-test']=data111['vector'].apply(lambda x:func1(x,data1['vector'].iloc[i]).pvalue)
    data6=data111.groupby('author')[['t-test']].mean()
    data7=data6.sort_values(by=['t-test'],ascending=False)
    A1.append(data7.index[0])
    B1.append(data7.index[1])
    C1.append(data7.index[2])
    T=[]
    for i in Z:
        a=len(data111[(data111['author']==i) & (data111['t-test']>0.9)])/len(data111[data111['author']==i])
        t.append(i)
        t.append(a)
        T.append(t)
        t=[]
    T=sorted(T,key=lambda x: x[1],reverse=True)
    T1.append(T[0][0])
    T2.append(T[1][0])
    T3.append(T[2][0])
data1['author1']=A1
data1['author2']=B1
data1['author3']=C1
data1['author4']=T1
data1['author5']=T2
data1['author6']=T3
```

Рис. 12.- Классификация с помощью функции stats.mannwhitneyu

Используя функцию stats.mannwhitneyu, получаем следующие результаты:

```
print(len(data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3']
print(len(data1[(data1['author']==data1['author4'])|(data1['author']==data1['author5'])|(data1['author']==data1['author6']
```

```
0.30603448275862066
0.2650862068965517
```

```
print(len(data1[data1['author']==data1['author3']])/len(data1))
print(len(data1[data1['author']==data1['author6']])/len(data1))
```

```
0.09051724137931035
0.09913793103448276
```

Рис. 13.- Результаты работы функции с точки зрения accuracy

Как мы видим, результат очень плохой. Мы использовали метрику на базе среднего значения полученного p-value. Мы исходим из той мысли, что чем больше сходство между векторами, тем больше значение p-value должно становиться. В колонках T1, T2, T3 находятся авторы, которые были отобраны за счет верно отгаданного количества авторов, соотношения p-значений которых больше 0.9. Результаты функции stats.ttest_ind также оставляют желать лучшего.

```
ata1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])])/len(data1)
```

```
0.2650862068965517
```

```
len(data1[data1['author']==data1['author3']])/len(data1)
```

```
0.08189655172413793
```

Рис. 14.- Результаты классификации с помощью функции stats.ttest_ind

Если метрика accuracy, основанная на среднем значении, работает плохо, можно применить сумму p - значений:

```
A1=[]
B1=[]
C1=[]
for i in range(len(data1)):
    data111['t-test']=data111['vector'].apply(lambda x:func1(x,data1['vector'].iloc[i]).pvalue)
    data6=data111.groupby('author')[['t-test']].sum()
    data7=data6.sort_values(by=['t-test'],ascending=False)
    A1.append(data7.index[0])
    B1.append(data7.index[1])
    C1.append(data7.index[2])
data1['author1']=A1
data1['author2']=B1
data1['author3']=C1
```

```
ata1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])])/len(data1
```

```
0.4676724137931034
```

```
len(data1[data1['author']==data1['author1']])/len(data1)
```

```
0.1961206896551724
```

Рис. 15.- Меняем принцип группировки на сумму

Как мы видим, результат улучшился в два раза для функции stats.mannwhitneyu, а для stats.ttest_ind

```
ata1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['autho
data1[(data1['author']==data1['author4'])|(data1['author']==data1['author5'])|(data1['author']==data1['auth
```

```
0.41594827586206895
```

```
print(len(data1[data1['author']==data1['author1']])/len(data1))
#print(len(data1[data1['author']==data1['author4']])/len(data1))
```

```
0.17456896551724138
```

Рис. 16.- Результаты работы классификатора с группировкой по сумме значений

Также улучшилось, но не так хорошо, как для предыдущей функции.

```
data111['vector']=data111['vector'].apply(lambda x:np.log(x))
```

```
data1['vector']=data1['vector'].apply(lambda x: np.log(x))
```

Рис.17.- Логарифмирование значений тренировочного и тестового датасетов

После логарифмирования значений векторов обоих датасетов, результаты функции stats.mannwhitneyu остались неизменными, а вот результаты работы функции stats.ttest_ind значительно улучшились:

```
ata1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==dat
data1[(data1['author']==data1['author4'])|(data1['author']==data1['author5'])|(data1['author']==da
```

```
0.4504310344827586
```

```
print(len(data1[data1['author']==data1['author1']])/len(data1))
#print(len(data1[data1['author']==data1['author4']])/len(data1))
```

```
0.2025862068965517
```

Рис.18.- Результаты работы функций stats.mannwhitneyu и stats.ttest_ind после логарифмирования.

Интересно также отметить, что после логарифмирования результат классификатора на основе ковариации немного снизился. Надо полагать, это связано с тем, что были минимизированы индивидуальные признаки, увеличивающие корреляцию [5]. А в случае с функцией на основе косинусного сходства, ситуация улучшилась.

```
len(data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])])/len(d
#print(len(data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])]
```

```
0.4224137931034483
```

```
len(data1[data1['author']==data1['author1']])/len(data1)
```

```
0.19181034482758622
```

Рис.19.- Результаты работы функции косинусного сходства после логарифмирования.

```
def hybrid_forward(A_hat, X, W):
    aggregate = np.dot(A_hat, X)
    propagate = relu(np.dot(aggregate, W))
    return propagate
def gcn_layer(A_hat, D_hat, X, W):
    return relu(D_hat * A_hat * X * W)
def relu (H):
    H[H<0]=0
    return H
def func(phrase1):
    if len(phrase1)>=4:
        A_hat=np.zeros((len(phrase1),len(phrase1)))
        for i in range(len(phrase1)):
            if i==0:
                for j in range(i,i+3):
                    A_hat[i,j]=1
            elif i==len(phrase1)-1:
                for j in range(i-2,i+1):
                    A_hat[i,j]=1
            else:
                for j in range(i-1,i+2):
                    A_hat[i,j]=1
        X=np.zeros((len(phrase1),2))
        X[:,0]=np.array(phrase1)
        X[:,1]=-np.array(phrase1)

        W_1=np.matrix([
          [-0.09987785,  0.44913911],
          [-0.27188145,  0.53559058]],
          dtype=float
            )
        W_2=np.matrix([
          [ 1.04352679,  1.32593575],
          [-1.64230931, -0.42889708]],
            dtype=float
          )
        D = np.sum(A_hat, axis=0)
        D_inv = D**-0.5
        D_inv = np.diag(D_inv)
        A_hat = D_inv * A_hat * D_inv
        H_1=hybrid_forward(A_hat, X, W_1)
        H_2=hybrid_forward(A_hat, H_1, W_2)
        return H_2[:,0]
    else:
        return [0]
```

Рис. 20.- Преобразование значений векторов с помощью CGN

Можно убедиться, что обработка значений векторов и их нормализация играют позитивную роль для сравнения предложений.

Обратимся теперь к графовым сверточным сетям CGN [6]. Если представить, что предложение-линейный граф [7], в котором не только непосредственное окружение накладывает отпечаток на определенный узел, но и самые отдаленные узлы также. Применение CGN [8] позволяет учитывать это влияние и изменять признаки каждого слова соответствующим образом. Сначала возьмем длинные предложения в data1 и data111 без обработки.

Используем функцию stats.mannwhitneyu [9] для нахождения t-критерия:

```
A1=[]
B1=[]
C1=[]
for i in range(len(data1)):
    data111['t-test']=data111['vector1'].apply(lambda x:func1(x,data1['vector1'].iloc[i]).pvalue)
    data6=data111.groupby('author')[['t-test']].sum()
    data7=data6.sort_values(by=['t-test'],ascending=False)
    A1.append(data7.index[0])
    B1.append(data7.index[1])
    C1.append(data7.index[2])
data1['author1']=A1
data1['author2']=B1
data1['author3']=C1
```

Рис. 21.- Использование функции stats.mannwhitneyu для нахождения t-критерия после обработки ветора CGN

```
data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])])/len(data1)
```

```
0.5
```

Рис. 22.- Результат совместного применения stats.mannwhitneyu и CGN для 3-х авторов

```
len(data1[data1['author']==data1['author1']])/len(data1)
```

```
0.20043103448275862
```

Рис. 23.- Результат совместного применения stats.mannwhitneyu и CGN для одного автора

Используем функцию stats.ttest_ind для нахождения t-критерия

```
A1=[]
B1=[]
C1=[]
t=[]
T=[]
T1=[]
T2=[]
T3=[]
for i in range(len(data1)):
    data111['t-test']=data111['vector1'].apply(lambda x:func2(x,data1['vector1'].iloc[i]).pvalue)
    data6=data111.groupby('author')[['t-test']].sum()
    data7=data6.sort_values(by=['t-test'],ascending=False)
    A1.append(data7.index[0])
    B1.append(data7.index[1])
    C1.append(data7.index[2])
    T=[]
    for i in Z:
        a=len(data111[(data111['author']==i) & (data111['t-test']>0.95)])/len(data111[data111['author']==i])
        t.append(i)
        t.append(a)
        T.append(t)
        t=[]
    T=sorted(T,key=lambda x: x[1],reverse=True)
    T1.append(T[0][0])
    T2.append(T[1][0])
    T3.append(T[2][0])
data1['author1']=A1
data1['author2']=B1
data1['author3']=C1
data1['author4']=T1
data1['author5']=T2
data1['author6']=T3
```

Рис. 24.- Использование функции stats.ttest_ind для нахождения t-критерия после обработки CGN

```
print(len(data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])])
print(len(data1[(data1['author']==data1['author4'])|(data1['author']==data1['author5'])|(data1['author']==data1['author6'])])
```

```
0.4504310344827586
0.2607758620689655
```

```
print(len(data1[data1['author']==data1['author1']])/len(data1))
print(len(data1[data1['author']==data1['author6']])/len(data1))
```

```
0.1724137931034483
0.1206896551724138
```

Рис. 25.- Результаты работы функции stats.ttest_ind в качестве третьего слоя.

Теперь сократим длину предложения на 20. Для функции 1:

```
data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1['author3'])])/1
```

```
0.4978448275862069
```

```
len(data1[data1['author']==data1['author1']])/len(data1)
```

```
0.20043103448275862
```

Рис. 26.- Сокращаем длину вектора до 20 элементов для функции stats.mannwhitneyu

для функции 2:

```
print(len(data1[(data1['author']==data1['author1'])|(data1['author']==da
print(len(data1[(data1['author']==data1['author4'])|(data1['author']==da
```

```
0.4482758620689655
0.2672413793103448
```

```
print(len(data1[data1['author']==data1['author1']])/len(data1))
print(len(data1[data1['author']==data1['author6']])/len(data1))
```

```
0.1724137931034483
0.125
```

Рис. 27.- Сокращаем длину вектора до 20 элементов для функции stats.ttest_ind

Практически без изменений. Теперь логарифмируем наши векторы. Получаем значение первой сокращенной функции

```
data1[(data1['author']==data1['author1'])|(data1['author']==data1['author2'])|(data1['author']==data1[
```

```
0.4978448275862069
```

```
len(data1[data1['author']==data1['author1']])/len(data1)
```

```
0.20043103448275862
```

Рис. 28.- Результат работы функции stats.mannwhitneyu после сокращения элементов до 20 и логарифмирования

Без изменений. Теперь результаты второй функции

```
print(len(data1[(data1['author']==data1['author1'])|(data1['author']==
print(len(data1[(data1['author']==data1['author4'])|(data1['author']==
```

```
0.4978448275862069
0.27370689655172414
```

```
print(len(data1[data1['author']==data1['author1']])/len(data1))
print(len(data1[data1['author']==data1['author6']])/len(data1))
```

```
0.20043103448275862
0.10344827586206896
```

Рис. 29.- Результат работы функции stats.ttest_ind после сокращения элементов до 20 и логарифмирования

Заметно определенное улучшение.

Результаты исследования

Максимальный уровень accuracy получают классификатор на основе CGN без логарифмирования для функции 1 и после логарфмирования для функции 2 после сокращения размера предложения data1 на 20.

В качестве входного слоя нашей графовой сверточной нейросети мы используем набор признаков, соответствующий матрице признаков X, полученных на основе статистического подсчета частотности слова в нашем корпусе. Каждому слову соответствует один признак. В больших моделях каждому слову могут соответствовать десятки или сотни признаков, что определяет большую точность LLM, таких как WordToVec [10] или еще более современных типа Bert. Предложение мы представляем как линейный граф, в котором каждое слово связано с предыдущим и последующим элементом.

Приходится принимать дополнительные меры для первого и последнего слова в предложении, так как оно замыкается на предыдущий или последующий элемент предложения и соответственно на себя. В качестве первого слоя мы используем произведение матрицы смежности A_hat на матрицу признаков X и на матрицу степеней вершин графа (degree matrix) D.

```
def hybrid_forward(A_hat, X, W):
    aggregate = np.dot(A_hat, X)
    propagate = relu(np.dot(aggregate, W))
    return propagate
```

Причем мы берем матрицу D в -0.5 , и на главной диагонали подсчитываем степень, соответствующую каждой вершине.

```
D = np.sum(A_hat, axis=0)
        D_inv = D**-0.5
        D_inv = np.diag(D_inv)
```

A_hat = D_inv * A_hat * D_inv

Это необходимо для того, чтобы нивелировать огромное влияние на значение вершины со стороны ее ближайших соседей и увеличить влияние дальних соседей. После получения первого скрытого слоя H_1 мы подаем его на второй скрытый слой вместо матрицы признаков:

H_1=hybrid_forward(A_hat, X, W_1)

H_2=hybrid_forward(A_hat, H_1, W_2)

Изначально мы следующие веса для двух слоев:

weights_0_1=np.matrix([[1, -1],
[-1, 1]], dtype=float)

weights_1_2=np.matrix([[1, -1],
[-1, 1]], dtype=float)

И сразу получаем результат:

Для множества из трех поэтов, набравших наибольшую сумму p-value, значение accuracy=0.5, а для одного поэта значение accuracy=0.2. Теперь потренируем нашу нейросеть:

```
alpha=0.01
prev=1e38
while True:
    for i in range(len(data1)):
        for j in range(len(data111)):
            if data1['author'].iloc[i]==data111['author'].iloc[j]:
                layer_0=data1['vector'].iloc[i]
                layerA=func(layer_0,weights_0_1, weights_1_2)
                if len(layerA)==2:
                    layer_1=layerA[0]
                    layer_2=layerA[1]
                    layer_0_0=data111['vector'].iloc[j]
                    layerB=func(layer_0_0,weights_0_1, weights_1_2)
                    if len(layerB)==2:
                        layer_1_1=layerB[0]
                        layer_2_2=layerB[1]
                        p_value=func1(layer_2[:,0],layer_2_2[:,0]).pvalue
                        error=1-p_value
                        if 0.5>error>0.01:
                            f=1
                        print('error=',error)
                        r+=1
                        if r==200:
                            k=1
                        if prev>error and f==1 and 0.5>=error>=0.01:
                            prev=error
                            print('r=',r,'k=',k)
                            layer_2_delta=(layer_2-error)
                            F=relu2deriv(layer_1)
                            layer_1_delta=layer_2_delta.dot(weights_1_2.T)
                            for i in range(layer_1_delta.shape[0]):
                                for j in range(layer_1_delta.shape[1]):
                                    layer_1_delta[i,j]=layer_1_delta[i,j]*F[i,j]
                            weights_1_2-=alpha*layer_1.T.dot(layer_2_delta)
                            weights_0_1-=alpha*np.array(layer_0).T.dot(layer_1_delta)
                            data111['vector1']=data111['vector'].apply(lambda x:[float(str(i).replace('[','').replace(']','')
                            data1['vector1']=data1['vector'].apply(lambda x:[float(str(i).replace('[','').replace(']','')) fo
```

Рис. 30.- Сверточная графовая нейросеть с корректировкой весов

Надо сказать, что для третьего выходного слоя мы используем функцию:

```
import scipy.stats as stats
def func1(a1,b1):
    return stats.mannwhitneyu(a1,b1,alternative='two-sided')
```

Эта функция рассчитана на ненормальное распределение в отличие от функции:

```
def func2(a1,b1):
    return stats.ttest_ind(a=a1,b=b1,equal_var=True),
```

рассчитанной на нормальное распределение.

Полное сходство дало бы единицу на выходе. Таким образом, мы можем рассчитать ошибку:

$$error=1-p\_value$$

Программа пробегает по датасету data1 и data111. Если автор совпадает, значения в соответствующих колонках ‘vector’ проходят через нейросеть, а затем отправляются в третий слой для получения p_value.

Так как образцы текста даже у одного автора могут сильно отличаться, приходится искусственно ограничивать диапазон ошибок для тренировки весов.

Выводы

Для определения принадлежности текста определенному автору хорошим решением является предварительная обработка признаков с помощью графовой сверточной нейросети. В качестве выходного слоя можно использовать сумму p-значений для анализируемого текста определенного поэта и всех остальных контрольных текстов, что позволяет найти наибольшую схожесть, сгруппировав поэтов по сумме p-значений. Эта функция очень подходит для анализа схожести из-за того, что нет

необходимости делать вектора одной длины, в отличие, например, от косинусной схожести или корреляции. Можно сделать заключение, что постановка одного признака, полученного с помощью Bag of Words, в соответствие каждому слову не является достаточной для реализации модели с высокой степенью accuracy. Тем не менее, эта модель показывает эффективность предварительной обработки признаков с помощью GCN.

Литература (References)

1. Hobson L., Cole H., Hannes H. Natural Language Processing in Action, Manning, Shelter Island, 575p.

2. Shorten C. Embedding Graphs with Deep Learning. URL: towardsdatascience.com/embedding-graphs-with-deep-learning-55e0c66d7752

3. Cohen M. Practical linear algebra for data science O'REILLY, 2023, 328 pp.

4. Sak A.N. XXIV International Scientific Conference "Construction the Formation of Living Environment" (FORM-2021) E3S Web Conf. Volume 263, 2021. URL: doi.org/10.1051/e3sconf/202126303013.

5. Trask A. Grokking Deep Learning -1st edition, Manning Publications, 2019, 352 pp.

6. Jepsen T. How to do Deep Learning on Graphs with Graphs Convolutional Networks. URL: towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d22507237805.

7. Hamilton W. Graph Representation Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, Vol. 14, No. 3, pp. 1-159.

8. Skiena S. The data science design manual.Springer, 2017, 512p.

9. Flawson Tong Graph Embedding for Deep Learning, URL: towardsdatascience.com/overview-of-deep-learning-on-graph-embeddings-

4305c10ad4a4.10. François Chollet Deep Learning with Python, Manning, 2021, 397 pp.

Дата поступления: 30.09.2024

Дата публикации: 14.11.2024