



КУРСОВ ПРОЕКТ

Анотиране на понятия в текст чрез Уикипедия

Факултет по Математика и Информатика

Студент: Борислав Стоянов Марков

Факултетен номер: 0MI3400048

Учебен план: Изкуствен Интелект (редовно, магистър)

Курс: Курс 1; **Група:** Група 1

Активен период: 2021/2022 летен, магистри

Дисциплина: Откриване на знания в текст

Дата: 28.06.2022г.



1. Съдържание

1. Съдържание	2
2. Резюме.....	2
3. Въведение	2
4. Преглед на областта.....	3
5. Данни и характеристики.....	4
6. Методи	5
7 . Експерименти/Резултати/Дискусия	9
8 . Заключение	12
8. Библиография.....	12
Приложения.....	13
1. Сурс код (Source code).....	13

2. Резюме

В този проект съм се опитал да открия понятия в текст само с един пас през текста, определяйки дължината на споменаването автоматично. За обучителни данни с аотиран текст съм използвал част от Уикипедия на български език.

3. Въведение

В деншно време особено важно е да се аотира текста на новини, за да може да се търси по семантични връзки. За целта имайки за вход един текст, ние искаме да разберем какви споменавания има на обекти от реалния свят, да аотираме текста. Могат да се поставят и линкове съответно към Уикипедия. Аотирането искаме да стане на един пас, без разпознаване на части на речта а само на базата на близост между вектори. Тъй като аотирането е скъпа операция, Уикипедия е чудесен източник на безплатни ръчно аотирани данни.

За вход имаме текст на български език, на изхода имаме аотиран текст или разпознати словосъчетания и към какво сочат в българската Уикипедия. Разпознаването става по следния начин: първо за всяка дума се определя контекст (5 думи напред и 5 думи назад). Изваденият контекст се векторизира с модел на трансформър от сайта на HuggingFace [5], който поддържа и български език. След това векторът се подава на невронна мрежа от 3 линейни слоя и Softmax класификатор на изхода. Мрежата е предварително обучена за този проект и на изхода се получава само един клас, отговарящ на дадена страница от Уикипедия.

Обучението на данните е по идея на статия от Гугъл[1], но доста неща не беше ясно как са направени и съответно за имплементация реших да сложа по-лесно и реалистично решение. Избраното решение е само демонстративно и се нуждае от подобрения. В оригиналната статия имаме косинус близост, а тук съм заложил невронна мрежа, която ни дава съответствието на $M \rightarrow E$, където M е споменаването(mention), а E е обекта от Уикипедия (Entity). Съответно разчитам, че невронната мрежа ще генерализира и подобни споменавания ще сочат към подобни обекти.

4. Преглед на областта

Последните работи в областта са фокусирани върху трениране на невронни мрежи кандидати и определяне на ранк (Francis-Landau et al., 2016; Eshel et al., 2017; Yamada et al., 2017a; Gupta et al., 2017; Silet et al., 2018). Като цяло тези работи изследват полезни контекстни фийчъри и използват нови подходи и архитектури, комбинирайки страната на споменаването и страната на обекта. Разширенията включват съвместно разпознаване на всички обекти в документ (Ratinov et al., 2011; Globerson et al., 2016; Ganea and

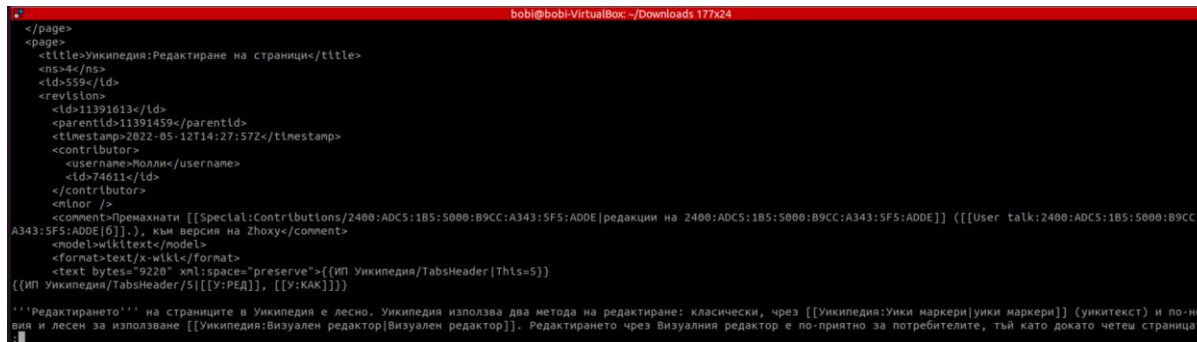
Hofmann, 2017), съвместно моделиране с релативни задачи като подобие на текст (Yamada et al., 2017b; Barrena et al., 2018) и междуезиково моделиране (Sil et al., 2018). Също в областта има и разработки с Reinforcement Learning, моделирайки Reinforcement агента като невронна мрежа с вход от контекста на споменаване, кандидат обекти и предишни решения (Joint Entity Linking with Deep Reinforcement Learning, Zheng Fang et al. 2019).

5. Данни и характеристики

Уикипедия е чудесен източник на анотирани данни, защото текста е изпълнен с вътрешни препратки и споменаванията са всъщност текста на връзката. Използвал съм част от българската уикипедия, свалена от <https://dumps.wikimedia.org/bgwiki/20220620/>. Архивът е **bgwiki-20220620-pages-articles.xml.bz2** 373.8 МБ. След разархивиране е около 3.5 ГБ. Като отправна точка за парсане на статиите съм ползвал собствен скрипт, който има заимствани методи от https://github.com/google-research/google-research/tree/master/dense_representations_for_entity_retrieval/. Идеята на скрипта „parse_wiki.py“ е да обходи статиите и за всяка страница от Уикипедия да намери споменаванията, а страницата да запамети като обект. Споменаванията са ограничени до 20 на страница. Извикването на скрипта става по следният начин:

```
python parse_wiki.py --bgwiki_archive=/home/bobi/Downloads/bgwiki-20220620-pages-articles.xml --max_pages=1000 --limit_mentions_per_page=20
```

Ето и суровият вид на файла на Уикипедия:

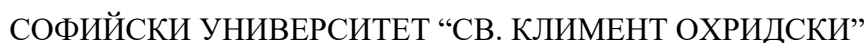


Фигура 5.1

Скриптът произвежда 2 csv файла, съответно **mentions.csv** и **entities.csv**.

mentions.csv					
idx	left_context	link_title	link_text	right_context	
6	календар от 15 октомври 1582	Файл:Christopher Clavius.jpg	мини Иезуитът [[Христ...	Григорианският календар е въведен в	
7	е въведен в употреба на	4 октомври	<null>	1582 г в съответствие с	
8	1582 г в съответствие с	була	<null>	от 24 февруари 1582 г	
9	в съответствие с була от	24 февруари	<null>	1582 г на папа Григорий	
10	с була от 24 февруари	1582	<null>	г на папа Григорий XIII	
11	24 февруари 1582 г на	папа	<null>	Григорий XIII чието име носи	
12	февруари 1582 г на папа	Григорий XIII	<null>	чието име носи и днес	
13	и днес Той поправя древноримския	Юлиански календар	<null>	като в него са нанесени	
14	отчете по точно дължината на	тропическа година	тропическата година	В Юлианския календар се приема	
15	времето между две последователни ...	Равноденствие	равноденствия	е 365 25 дни и	

Фигура 5.2



idx	title	text	url
0	Григориански календар	'Григорианският календар (понякога наричан и Грегори...	https://bg.wikipedia.org/wiki/%D0
1	GNU General Public License	GNU General Public License (на български преведан к...	https://bg.wikipedia.org/wiki/GNU
2	Лиценз за свободна документация на ГНУ	мини/Лого на GFDLЛицензът за свободна документация...	https://bg.wikipedia.org/wiki/%D0
3	GNU FDL	виж Лиценз за свобод	https://bg.wikipedia.org/wiki/GNU
4	GNU Free Documentation License	виж Лиценз_за_свобод	https://bg.wikipedia.org/wiki/GNU
5	България	Република България е държава в Югоизточна Европа	https://bg.wikipedia.org/wiki/%D0
6	История на България	Историята на българските земи може да бъде условно р...	https://bg.wikipedia.org/wiki/%D0
7	Септември	Септември е името на деветия месец от годината споре...	https://bg.wikipedia.org/wiki/%D0

Правим множество за обучение, като обединим двата файла в Pandas DataFrame, в който имаме от една страна всички споменавания, от друга страна – съответните обекти(entities).

```
In [479]: 1 # Merge mentions and entities (inner merge)
2 merge_df = mentions_df.merge(entities_df, \
3                               left_on='link_title', \
4                               right_on='title', \
5                               how='inner',
6                               suffixes=['_mention', '_entity'])
7 print('Eligible mentions: ', len(merge_df))
8 merge_df.head(2)
```

Eligible mentions: 3739

Разделяме данните на обучаемо и тестово множество от данни в пропорция 80%/20%.

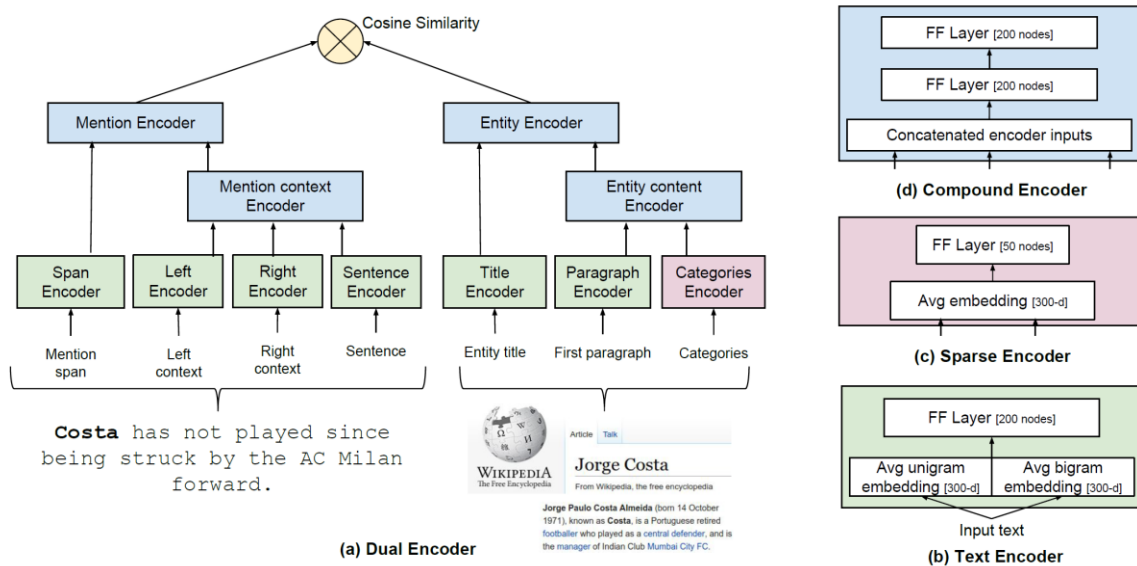
```
In [501]: 1 train_size = int(0.8 * len(dataset))
2 test_size = len(dataset) - train_size
3 train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, test_size])
4 print(f"len(train_dataset)={len(train_dataset)}")
5 print(f"len(test_dataset)={len(test_dataset)}")

len(train_dataset)=2991
len(test_dataset)=748
```

6. Методи

Първоначалната идея на проекта е според статия [1] използваща косинус близост между кодирано споменаване и кодиран обект. Двойният енкодер научава кодирането на споменаването – ϕ и научава кодирането на обекта – ψ . След това смята косинус близост.

$$s(m, e) = \cos(\phi(m), \psi(e))$$



Фигура 5.6 – Learning Dense Representations for Entity Retrieval [1]

Поради недостатъчно обяснения как работи оригиналният алгоритъм, реших да подменя някои компоненти и да направя собствена имплементация използвайки невронна мрежа и готово кодиране на споменаването „m“ (mention) и обекта „e“ (entity). Кодирането на вектори по даден текст правим с готови научени модели на трансформъри, поддържащи български език. Можете да намерите пример на адрес <https://huggingface.co/sentence-transformers/all-MiniLM-L12-v1>. Примерно кодиране:

```
from sentence_transformers import SentenceTransformer
sentences = ["This is an example sentence", "Each sentence is converted"]

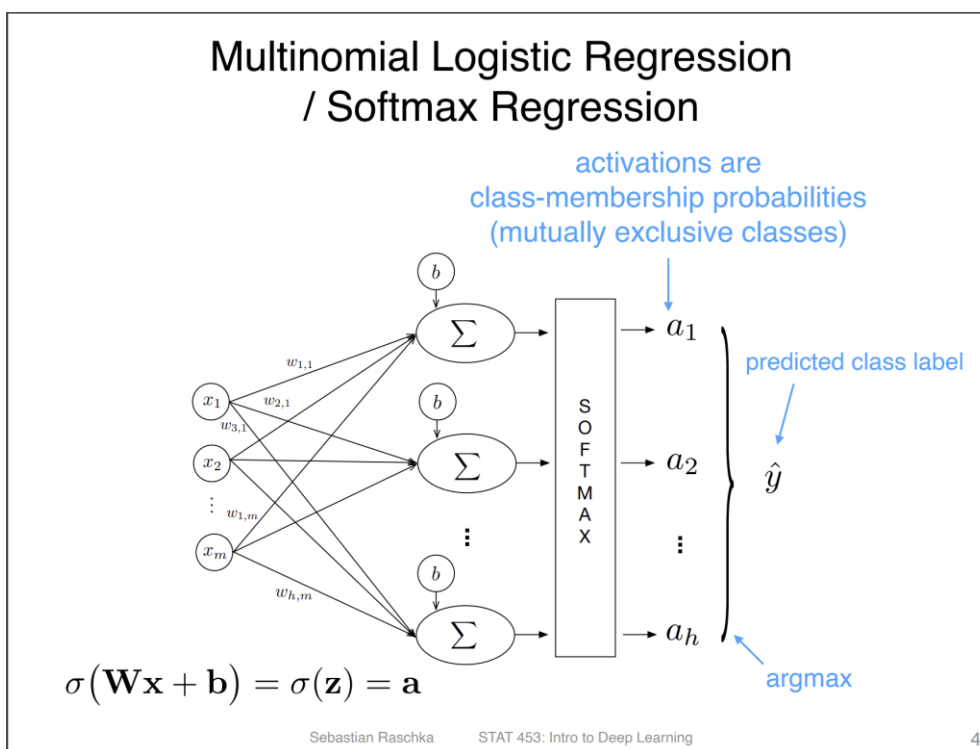
model = SentenceTransformer('sentence-transformers/all-MiniLM-L12-v1')
embeddings = model.encode(sentences)
print(embeddings)
```

И така задачата за аотиране на текст се свежда до следното. По зададено „m“, което включва контекст по 5 думи от ляво и 5 думи от дясно и самият текст на връзката, ние да намерим такава функция „f“, която да ни дава съответното „e“. Векторизацията на „m“ ще означим „hugging_face(.)“. По-формално имаме:

$$f(m) = \text{neural_net}(\text{hugging_face}(m)) = e$$

$$m = \text{лява_дума1} + „“ + \dots + \text{лява_дума5} + \text{текст_линк} + \text{дясна_дума1} + „“ + \dots + \text{дясна_дума5}$$

Невронната мрежа на изхода има функция на активация $\text{softmax}(\cdot)$, която ни осигурява избор на само един клас от много възможни. За softmax можете да научите повече от слайдовете на Себастиан Рашка [4]



Фигура 5.7. Sebastian Raschka, STAT 453: Intro to Deep Learning [4]

Softmax активационна функция изглежда по този начин, като горен индекс в квадратни скоби имаме поредността от група записи(batch number)

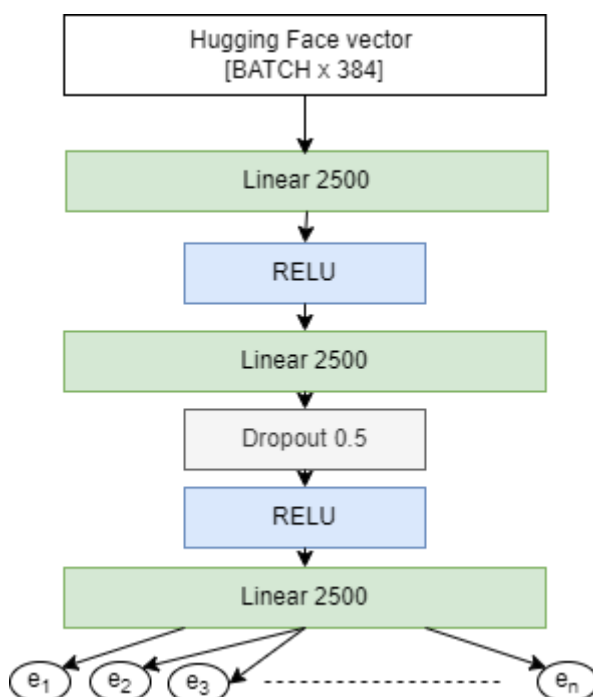
$$P(y = t \mid z_t^{[i]}) = \sigma_{\text{softmax}}(z_t^{[i]}) = \frac{e^{z_t^{[i]}}}{\sum_{j=1}^h e^{z_j^{[i]}}}$$

$t \in \{j \dots h\}$

h is the number of class labels

Фигура 5.8. Sebastian Raschka, STAT 453: Intro to Deep Learning [4]

Даваме структура на невронна мрежа, която ще ни представлява функцията $\text{neural_net}(\cdot)$ на следващата фигура.



Фигура 5.9

Следва и имплементацията на Python като код на горепосочената мрежа.

```

In [502]: 1 class MentionToEntityNet(nn.Module):
2     def __init__(self, out_size, in_size=384, dropout=0.5):
3         super(MentionToEntityNet, self).__init__()
4         #self.flatten = nn.Flatten()
5         self.linear_relu_stack = nn.Sequential(
6             nn.Linear(in_size, 2500),
7             nn.ReLU(),
8             nn.Linear(2500, 1500),
9             nn.Dropout(dropout),
10            nn.ReLU(),
11            nn.Linear(1500, out_size),
12        )
13        self.linear_relu_stack.apply(self._init_weights)
14    def _init_weights(self, m):
15        if isinstance(m, nn.Linear):
16            torch.nn.init.xavier_uniform_(m.weight)
17            m.bias.data.fill_(0.01)
18
19    def forward(self, x):
20        logits = self.linear_relu_stack(x)
21        # Softmax is used in evaluation later on,
22        # This method returns logits directly in order
23        # to call CrossEntropyLoss on logits
24        return logits
25
  
```

Фигура 5.10

Като функция на грешката използваме CrossEntropyLoss, която работи директно с логитите. Логити (logits) наричаме ненормализирани изходи от последния линеен слой.

По формално можем да запишем функцията на грешката като:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})} \cdot 1\{y_n \neq \text{ignore_index}\}$$

В случая това е за минибач(minibatch) и при редукция „mean“ се използва следната формула от PyTorch.

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{y_n} \cdot 1\{y_n \neq \text{ignore_index}\}} l_n, & \text{if reduction = 'mean';} \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

По-подробно може да се прочете в документацията на PyTorch [3] <https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html>.

7 . Експерименти/Резултати/Дискусия

След обучение на мрежата настройваме част от хиперпараметрите до постигане на оптимален резултат. За измерване сме избрали метриката точност(ассигасу), като ще следим и точността на обучаемото и на тестовото множество докато обучаваме мрежата.

```
# Training cycle
MAX_EPOCHS = 40
BATCH_SIZE = 32
DISPLAY_STEP = 4
LEARNING_RATE = 0.001
```

Таблица 7.1

Скоростта на обучение е параметър, който зависи от броя на обучаемите тегла в невронната мрежа и се избира евристично. При повече параметри на мрежата коефициента трябва да бъде все по-малък. В случая мрежата не е с много параметри и съответно няма нужда да бъде прекалено малък, защото обучението ще е много бавно.

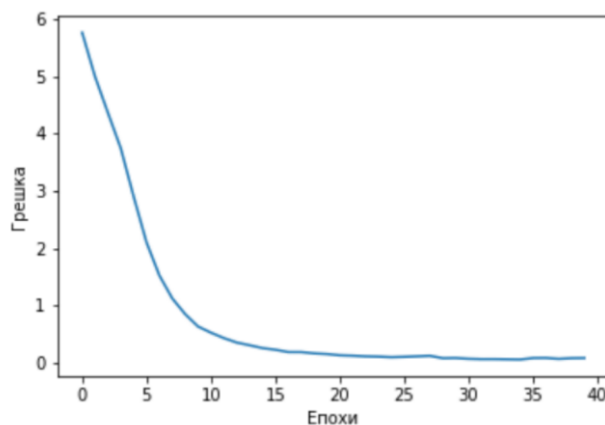
След всяка стъпка на обучение пресмятаме грешката, колко е точността върху тестовото и точността върху обучаемото множество. Формулата за точност изглежда формално така: $Acc = \frac{1}{N} \sum_i^N 1(y_i = \hat{y}_i)$, където y_i е правилният клас, \hat{y}_i е предвидения клас.

Това са броя правилно класифицирани класове разделени на всички записи от множеството. След обучение имаме следните резултати:

```
100% 40/40 [04:16<00:00, 7.23s/it]
Epoch: 0004 loss=3.74424, train_acc=0.38315, test_acc=0.26203
Epoch: 0008 loss=1.11976, train_acc=0.79472, test_acc=0.38770
Epoch: 0012 loss=0.42701, train_acc=0.92745, test_acc=0.39706
Epoch: 0016 loss=0.22309, train_acc=0.96122, test_acc=0.40508
Epoch: 0020 loss=0.14774, train_acc=0.97727, test_acc=0.40508
Epoch: 0024 loss=0.10322, train_acc=0.98061, test_acc=0.39706
Epoch: 0028 loss=0.11734, train_acc=0.98128, test_acc=0.39305
Epoch: 0032 loss=0.05949, train_acc=0.98830, test_acc=0.38770
Epoch: 0036 loss=0.07779, train_acc=0.98195, test_acc=0.38369
Epoch: 0040 loss=0.07814, train_acc=0.98429, test_acc=0.39572
Optimization Finished!
```

Фигура 7.1

Тук принтираме резултат на всяка четвърта епоха. Графиката на грешката спрямо поредния номер епоха даваме на следващата графика.



Фигура 7.2

Грешката пада с течение на епохите. Виждам, че точността над обучаемото множество нараства, което е очаквано, но точността над тестовото множество не може да нарасне над 0.40. Това означава, че модела не може да генерализира добре. Тоест по дадените думи не може да предвиди кое е ентитето. В случая считам, че не всяко споменаване (mention) е с еднакво качество и е еднакво близко до обекта (entity). Освен това не използвахме косинус близостта между “m” и “e”, както е в оригиналната статия. Това именно е довело до ниската точност на модела. Сега нека да видим и реален текст и дали наистина ще разпознаем някакви споменавания (очакваме до 40% да бъдат разпознати).

```
text = ""
Дни, използвани от слънчевите календари
При слънчевите календари датата отговаря
на слънчевия ден. Денят може да се състои
от периода между изгрев и залез слънце,
последван от нощта, или може да представлява
времето между повтарящи се събития, например
```

два залеза. Дължината може да се променя малко през годината или може да се използва среден слънчев ден. Други типове календари също могат да използват слънчевия ден.

Юлиански и Григориански календар

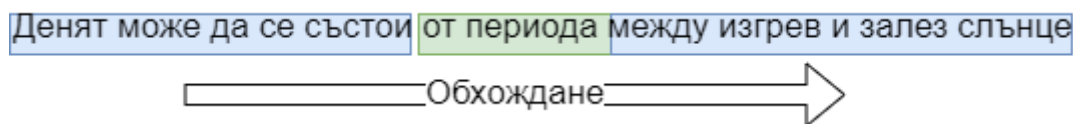
Юлианският календар е въведен от римския диктатор Юлий Цезар през 46 г. пр.н.е. При него месеците са по-дълги от лунния цикъл и затова той не е удобен за следене на лунните фази, за сметка на това много точно показва сезоните. Обикновените години имат 365 дена, а всяка четвърта година е високосна, което означава, че има 366 дни. Така продължителността на средната година е 365,25 дни.

""

```
stop_words = {"и", "на", "в", ".", "се", "да", \
              "от", "или", "през", "а", "може", \
              "например", "той", "не", "че"}
```

Таблица 7.2

Разпознаването на текста става като пуснем един брояч от началната дума и премине до края и съответно вземаме 5 думи назад и 5 думи напред. В допълнение пробваме с прозорец с една и две думи (ако споменаването е една или две думи). Тук е момента да спомена, че нямаме клас за да кажем, че дадена дума или думи не са никакво споменаване. Липсва негативно свързване (negative mining). В момента единствения начин да избегнем категоризация е да гледаме колко е вероятността текущата дума/думи да попадат в някакъв клас(т.е. са обект(entity)). За да избегнем да категоризираме всяка дума като споменаване слагаме филтър “if max_prob > 0.98 “ и показваме като избрани думи само тези, за които модела дава висока вероятност.



Фигура 7.3

Ето и резултата от обхождането, дадена е фразата, вероятността и обекта.

```
между изгрев==0.99325=>Варна
изгрев==0.99325=>Варна
използва среден==0.99609=>Европа
среден==0.99609=>Европа
слънчев ден==0.99627=>Григориански календар
ден==0.99627=>Григориански календар
```



ден Други==0.98896=>Григориански календар
Други==0.98896=>Григориански календар
римския диктатор==0.99753=>България
диктатор==0.99753=>България
удобен==0.98141=>Рим
сметка==0.98468=>1 април
365 дена==0.98328=>Обикновена година, започваща във вторник
дена==0.98328=>Обикновена година, започваща във вторник
високосна==0.99687=>Варна
дни==0.99435=>Григориански календар

Таблица 7.3

Тук пропускаме стоп думите, тъй като те не могат да са смислено споменаване. Виждаме, че на места имаме попадения като „дни“=> “Григориански календар“, „слънчев ден“=>“Григориански календар“, но имаме и много грешни попадения, например „Други“=>“Григориански календар“, „удобен“=>“Рим“ и т.н. Като цяло считам, че този модел трябва да се подобри и да се вкара косинус близост във функцията на грешката и да се промени кодирането на изреченията.

8 . Заключение

В заключение ще спомена, че изложеният модел не може да генерализира добре кое е споменаване и кое не за обекти от реалния свят. Трябва да се ползва косинус близост във функцията на грешката на невронната мрежа и всяко споменаване да има своя качествена оценка до колко е близко до реалния споменаван обект. Т.е. ние не ползвахме никъде първото изречение на обекта (entity) за да преценим колко е близко до споменаванията, сочещи към него. Друг проблем е, че ако увеличим броя на Уики страниците които слагаме в множеството обучаеми данни, класовете на невронната мрежа биха нарастнали много и времето за разпознаване ще нарастне.

8. Библиография

[1] Learning Dense Representations for Entity Retrieval, Daniel Gillick et. al., 2019 [PDF]

<https://aclanthology.org/K19-1049.pdf?fbclid=IwAR0vPi8AQrc5rPeAF2sLjZ9lg5K5WUTSwmhFPuxx436kmi-SN0RZ8OtdLLo>

[2] End-to-End Retrieval in Continuous Space, Daniel Gillick et al. 2018



[3] Deep Learning for Coders with fastai & PyTorch, Jeremy Howard & Sylvain Gugger, O'Reilly, 2020

[3] Pytorch documentation, <https://pytorch.org/docs/stable/torch.html>

[4] Logistic Regression and Multi-class classification, Lecture 08, Sebastian Raschka, https://sebastianraschka.com/pdf/lecture-notes/stat453ss21/L08_logistic_slides.pdf

[5] Sentence Transformers, <https://huggingface.co/sentence-transformers/all-MiniLM-L12-v1>

Приложения

1. Сопс код (Source code)

<https://github.com/borkox/uni-sofia-nlp-wiki-entity-linking>