Икономически университет - Варна

Катедра "Информатика"

РЕФЕРАТ

по дисциплината "Езици за програмиране"

на тема: "Модел за автоматично сричкообразуване на български думи основан на ненадзиравано машинно самообучение"

Изготвил: докт. Красен Пенчев

Варна

2018

Съдържание

Въведение

Съществуват много дефиниции и дискусии за същността и значението на сричката в речта. Някои лингвисти отдават голямо значение на сричката, поставяйки я на централно място в теориите си. Изповядващите другата крайност я смятат за несъществена единица на говоримия език. Една от причините за различните гледни точки е предизвикателството точно да бъде определена същността на сричката.

За текущия реферат се приема определението, че сричката е единица на говоримия език, която е по-голяма от фонема, и се състои от един гласен звук, който може да бъде предшестван и, или последван от един или повече съгласни звуци[1] (Huang, Acero и Hon 2001). Дефиницията за сричка е подходяща, защото обхваща споменатите в предходния абзац общоприети признаци на сричката, без да засяга такива, за които учените не са единодушни.

Въпреки различните мнения на учените, хората използващи даден език са способни да броят сричките в думите единствено използвайки интуицията си. Имайки предвид това, може да се каже, че сричката, макар и не всепризнато от лингвистите, е структурна единица в изграждането на думата. Възможността за надеждно определяне на границите на сричката би имала широко приложение. Правилното “разбиване” на думите на срички би спомогнало за моделирането на думи в системите за автоматично преобразуване на реч в текст (Marchand, Adsett и Damper 2009). Някои от съществуващите индекси за оценка на четимост на текст, например Flesch-Kincaid, включват като задължителна стъпка анализ на сричките в думите съставящи текста. Сричките могат да бъдат използвани като индексирани термини при изграждането на текстови търсачки (full-text search).

Текущият реферат представя модел за автоматично сричкообразуване, основан на машинно самообучение. Целта е създаване на модел с общо предназначение, който да адресира споменатите в предходния абзац проблеми на автоматичното сричкообразуване. Разглежданият подход не е ограничен нито от количество тренировъчни данни, нито от предметната област на

[1] В литературата, съчетанието между ядрото и крайната част на сричката често се нарича „рима“.

данните. Напротив, колкото по-всеобхватна е тематиката на обучителните данни, толкова по-добри ще са възможностите на модела за идентифициране на границите на срички.

1 Подходи за автоматично сричкообразуване

Съществуват два основни подхода за автоматично сричкообразуване, единият от които базиран на предварително дефинирани правила, а другият на машинно самообучение (Marchand, Adsett и Damper 2007). Първият подход използва универсални правила за сричкообразуване, каквито бяха споменати във въведението. Основаният на машинно самообучение подход използва корпус, чиито думи са правилно „разбити“ на срички, за да обработва непознати думи. Доказано е, че подходът основан на предварително съставени правила показва по-слаби резултати от базирания на машинно самообучение (Marchand, Adsett и Damper 2007, 2009). По-слабите резултати се срещат в случаите на прилагане на моделите, основани на предварително дефинирани правила, върху данни с различна тематика. Обикновено подходите основани на предварително определени правила не са с универсално предназначение. Например модел, който е създаден за обработка на правни документи би се справил слабо в обработката на поетични текстове в сравнение с модел с общо предназначение.

1.1 Сричкообразуване основано на универсални правила

Съществуват три популярни метода за идентифициране на граници на срички, наречени съответно принцип за максимизиране на начална част (на сричка) (Kahn 1980), принцип на звуковата последователност (Selkirk 1984), и принцип на легалност (Goslin и Frauenfelder 2001).

При принципа на звуковата последователност, на всеки звук в сричката бива дадена числова стойност според скалата на звучност. Гласните звуци са с най-висок ранг, следвани от носовите съгласни, фрикативите и плозивите. Стойностите на последователните звуци нарастват в началната част на сричката и намаляват в крайната част, кодата. Основният проблем

на този подход е невъзможността за правилно определяне на границите на сричките, когато има повече от една възможност за поставяне на граница. Повече от една възможност за поставяне на граница има в случай на струпване на съгласни звуци. Предимство на подхода е независимостта му от тренировъчните данни.

Принципът на легалност позволява струпване на съгласни звуци да бъде валидна начална или крайна част на сричка, само ако е било срещано като начално или крайно струпване на съгласни в дума. Това значи, че начална част на сричка може да бъде валидна единствено ако се е срещала като начална последователност от звуци на дума от тренировъчните данни. Този принцип има същия недостатък като принципа на звуковата последователност - когато има няколко начина за разбиване на струпване на съгласни звуци, границите между сричките са неясни.

При третия метод, принципа за максимизиране на началната част на сричката, когато има няколко варианта за разбиване на струпване на съгласни звуци се предпочита този вариант, при който началната част е с по-голяма дължина. Подходът е силно зависим от качеството и количеството на обучителните данни.

1.2 Сричкообразуване основано на машинно самообучение

Определянето на границите на сричката изцяло чрез предварително съставени правила често не постига желаните резултати. В някои ситуации, правилата са недостатъчни за ясно поставяне на границите на сричките, съставящи обработваната дума. Недостатъците на предварително съставени правила водят до нуждата от по-добри методи за автоматично сричкообразуване. Съществуват множество подходи, основани на машинно самообучение.

Müller (2006) предлага подход, който включва разработване на граматики за характеризиране на фонологичната структура на думите. Използвайки граматики, думата бива представена като последователност от срички. Всяка сричка, от своя страна е представена от съставните ѝ начална част и

рима. Римата се състои от ядрото и кодата на сричката. Всички граматики в предложения подход разграничават едносрични и многосрични думи и различни по брой струпвания на съгласни звуци.

При подхода на Bartlett, Kondrak и Cherry (2009) са постигнати едни от най-добрите резултати в автоматичното сричкообразуване. В подхода се използва комбинация от метода на опорния вектор и скрити модели на Марков. Всяка фонема бива класифицирана спрямо позицията си в думата, взимайки под внимание съвкупност от характеристики. Скритите модели на Марков преодоляват недостатъка на третирането на фонемите самостоятелно. При тренирането на модела, към всяка дума бива асоцииран клас, избран от съвкупността на всички възможни класове. Класовете представляват последователности от срички, представени като последователност от начална част, ядро и крайна част.

Представеният в текущия реферат модел стъпва на предложения от Mayer (2010) подход за независимо от езика автоматично, сричкообразуване. За всяка една дума в тренировъчните данни, моделът намира всички възможни комбинации от срички, използвайки универсалните правила за сричкообразуване. В същото време, честотите на срещане на възможните срички биват акумулирани. При обработка на непознати думи, моделът избира тази комбинация от срички, която има най-голям сбор на честотите на срещане на отделните срички.

Между подхода на Mayer и предлагания в този реферат метод има две съществени разлики. Първата разлика е, че текущият модел няма за цел да бъде независим от езика. Текущата разработка се отнася само за български език. Въпреки, че се отнася само за българския език, моделът има минимално “познание” за него, разликата между гласни и съгласни звуци, и основните правила за сричкообразуване, посочени в определението за сричка, предложено в началото на тази точка. Втората разлика е, че разработеният модел, в процеса на обучение, акумулира данни за характеристики на сричките, които методът на Mayer не взима под внимание.

2 Модел за автоматично сричкообразуване

Както името им подсказва, моделите основани на машинно самообучение трябва да бъдат *обучени* преди да бъдат използвани върху непознати данни. Освен процеса на обучение, който е втори по ред на изпълнение, има още два процеса, които също са важни и заслужават да им бъде обърнато внимание.

Първият процес, предшестващ обучението, включва събирането и подготовката на тренировъчни данни. Подготовката най-често включва "почистване" на данните, унифициране на формата им, и всички действия, които биха намалили "шума" в данните.

Обучението, което следва подготовката на данните, зависи пряко от качеството на тренировъчните данни. Въпреки, че зависи от качеството на обучителните данни, може да се каже, че процесът на обучение е също толкова важен, колкото и подготовката на данните.

След успешното обучение на модела следва процеса на оценка на резултатите. Този процес дава реална информация за качеството на предходните процеси по събиране на тренировъчни данни, и обучение.

2.1 Подбор и подготовка на тренировъчни данни

Същността на разработения модел предполага, че той ще оперира върху отделни думи. Това в голяма степен опростява процеса по подготовка на тренировъчни данни. Извличането на отделни думи от текст е много по-проста операция от например анализ на изречения или параграфи, защото се избягват сложни алгоритми за идентифициране на препинателни знаци и техния контекст. В подготовката на тренировъчните данни могат да бъдат идентифицирани две стъпки, които са зависими помежду си.

Първата стъпка включва събирането на "груби" тренировъчни данни. В конкретния случай, тези данни представляват корпус от текстове. Важно е подбраните текстовете да бъдат на разнообразна тематика, за да може броят на уникалните думи да бъде възможно най-голям, представяйки същевременно богатството на българския език. Обучението на представяния подход за автоматично сричкообразуване би било повлияно благоприятно

ако тренировъчните данни съдържат думи от възможно най-много сфери на познание.

Следващата стъпка в подготовката на тренировъчните данни е трансформирането на събрания в предходната стъпка корпус до отделни думи. Този процес се нарича токенизация. Токенизацията, в текущия случай, може да бъде дефинирана като сегментация на символен низ до думите, които го изграждат. Важно е да се има в предвид, че токенизацията има различен смисъл ако се използва в контекста на преобразуване на програмен код до абстрактно синтактично дърво. Токенизацията има съвсем друг смисъл в областта на електронните финанси, където представлява метод за сигурно съхранение на номера на дебитни и кредитни карти (Díaz-Santiago, Maria Rodriguez-Henriquez и Chakraborty 2014).

Съществуват множество подходи за токенизация, основаващи се на предварително съставени правила, на машинно самообучение, но те не са предмет на анализ в текущия реферат. В трудовете на Habert и др. (1998) и He и Kayaalp (2006) се разглеждат множество подходи за токенизация, тяхната успеваемост и скорост. Въпреки изобилието на подходи за сегментация е важно да бъде избран най-подходящият за нуждите на представения в текущия реферат модел.

След токенизацията на корпуса следва да бъдат премахнати дублираните думи. Също като за сегментацията, така и за премахване на повтарящите се думи съществуват множество подходи. Изборът на подход за премахване на дубликатите не е предмет на текущия реферат. Тъй като моделът не взима предвид семантичното значение на думата, размерът на буквите може да бъде унифициран. Тази стъпка е важна, защото при евентуалното ѝ пропускане може да се очакват некоректни резултати.

Разполагайки със списък от неповтарящи се думи, с унифициран размер на буквите, може да се пристъпи към обучението на модела.

2.2 Обучение на модела за автоматично сричкообразуване

Също като процеса по подготовка на тренировъчни данни, разгледан в предходните редове, процесът на обучение на модела за идентифициране на границите на срички също може да бъде разделен на няколко стъпки.

Първата стъпка включва извличането на възможните комбинации между сричките на думата. Правилата за извличане се основават на предложеното определение за сричка във въведението на текущия реферат. Този процес се повтаря за всяка една дума в тренировъчните данни.

Да вземе за пример думата "червен". Тя може да бъде разделена на срички по няколко начина. В един случай се получава комбинацията (че, рвен), в друг (чер, вен), а в трети, (черв, ен). Всеки човек, говорещ български език, може да посочи, че втората комбинация съдържа сричките на думата "червен".

След като бъдат извлечени възможните комбинации между сричките за всички думи, резултатите трябва да бъдат акумулирани. Конкретната структура от данни в която да бъдат акумулирани обработените тренировъчни данни не е предмет на дискусия в текущия реферат. Въпреки това е добре да се спомене, че структурата от данни е тясно свързана с възможностите за съхранение на тренирания модел при бъдеща практическа имплементация.

При подходът на Mayer (2010) се изчислява честота на срещане на всяка една сричка от извлечените комбинации между срички за всяка една от думите в тренировъчните данни. Разработеният в текущия реферат метод също използва тази метрика, но е добавено още едно ниво на информация. Не само се изчисляват честотите на срещане за всяка сричка, а се пази и информация за местоположението ѝ в думата.

Mayer (2010) достига до извода, че съществуват три основни местоположения, които сричката може да заема в една дума, начално (initial), междинно (medial), и крайно (final). Ако бъде взета за пример думата "параван", резултатът от тренирането на модела върху нея би бил: (па$_i$ ра$_m$ ван$_f$), (пар$_i$ ав$_m$ ан$_f$), (па$_i$ рав$_m$ ан$_f$), където i, m, f обозначават местоположението на сричката в думата. В случай, че думата е с повече от три срички ще има

повече от една сричка в междинна позиция.

След края на обучението на модела, след обработката на всяка дума поотделно и акумулирането на резултатите, следва фазата на оценка на модела. Във фазата на оценка може да бъде получена реална мярка за качеството на предходните два процеса, събирането на обучителни данни и самото обучение на модела.

2.3 Идентифициране на границите на срички в думите

Преди моделът да "вземе решение" за правилната комбинация от срички в думата, която обработва, всички възможни комбинации от срички трябва да бъдат извлечени. Тази стъпка се повтаря и в процеса на обучение на модела. След извличането на всички възможни комбинации от срички, данните за тях, честотите им на срещане и позициите им се използват за намиране на правилната комбинация.

Ако се върнем на примера с думата "червен", след обучението на модела е възможно да разполагаме със следните примерни данни за думата:

$$
\begin{aligned}
\text{че} &= (\text{че}_{[i,290]}, \text{че}_{[f,185]}) \\
\text{рвен} &= (\text{рвен}_{[i,146]}, \text{рвен}_{[f,125]}) \\
\text{чер} &= (\text{чер}_{[i,273]}, \text{чер}_{[f,254]}) \\
\text{вен} &= (\text{вен}_{[i,233]}, \text{вен}_{[f,192]}) \\
\text{черв} &= (\text{черв}_{[i,122]}, \text{черв}_{[f,108]}) \\
\text{ен} &= (\text{ен}_{[i,157]}, \text{ен}_{[f,231]})
\end{aligned}
$$

За целите на примера умишлено е избрана кратка дума, тъй като колкото по-дълга е една дума, толкова повече са възможните ѝ комбинации от срички. За най-дългата дума в българския език "непротивоконституционалителствувайте", възможните комбинации от срички са над 1500, а самите срички участващи в комбинациите са много повече.

Моделът проверява сумите на честотите на сричките, взимайки предвид и позициите за които са записани. По този начин, автоматичното

сричкообразуване се свежда до решаване на следното уравнение:

$$\text{че}_{[i,290]} + \text{рвен}_{[f,125]} = 415$$
$$\text{чер}_{[i,273]} + \text{вен}_{[f,192]} = 465$$
$$\text{черв}_{[i,122]} + \text{ен}_{[f,231]} = 353$$

Както се вижда от резултатите на трите уравнения, сричките “чер” и “вен” са най-вероятните градивни елементи на думата “червен”. От примера става ясно каква е причината за добавяне на още едно ниво на информация, използване на контекста на вече срещаните срички за по-прецизно предсказване на сричките в непознати думи.

В реална обстановка е възможно да се случи така, че да има уравнения с еднакви резултати. В такъв случай се използва сумата на честотите на сричките, без позициите да бъдат взимани под внимание. Ако отново се стигне до ситуация на равни суми, се пристъпва до използването на принципа за максимизиране на началната част на сричката.

Въпреки че начина на работа на модела за идентифициране на границите на срички е ясен, съществуват трудности пред процеса по оценка на успеваемостта. Макар оценяването на успеваемостта на разработения модел да не е предмет на текущия реферат, нужно е да се споменат някои от трудностите пред адекватната оценка на модела.

Една от причините за трудностите пред оценката на успеваемостта на модела е съставянето на златен стандарт[2]. Според Duanmu (2009), дори в английски език, език с ясно дефинирани лингвистични правила, съществуват разногласия между учените за правилното сричкообразуване на дума като “happy”. Имайки предвид неединодушието относно значението на сричката в речта, съществуването на множество верни комбинации между срички за една и съща дума не е учудващо.

При тестването на методите им за автоматично сричкообразуване, Goldwater и Johnson (2005) правят разграничение между думи изградени

[2]Прието е данните, които служат за оценка на успеваемост на алгоритми и модели да се наричат златен стандарт.

от поне две срички и думи, изградени от какъвто и да е брой срички. При някои от тестваните методи се забелязва разлика в резултатите от няколко процента до почти 30% успеваемост. Авторите доказват, че е по-лесно да се постигне по-висока успеваемост на идентифициране на границите на сричката при тестване върху данни, съдържащи голям брой едносрични думи и редки струпвания на съгласни звуци.

Съдържанието на тестовите данни е важен аспект при оценката на подходи за автоматично сричкообразуване. Съществуват различни виждания по въпроса дали данните трябва да съдържат произволна извадка думи или е нужно да има някакви критерии при подбора на тестовите данни. Ако тестовите данни съдържат голям брой едносрични думи, успеваемостта на модела е много по-висока. Това е така, защото при едносричните думи няма нужда от разбиване на струпани съгласни звуци (Goldwater и Johnson 2005).

3 Архитектура на софтуерната реализация

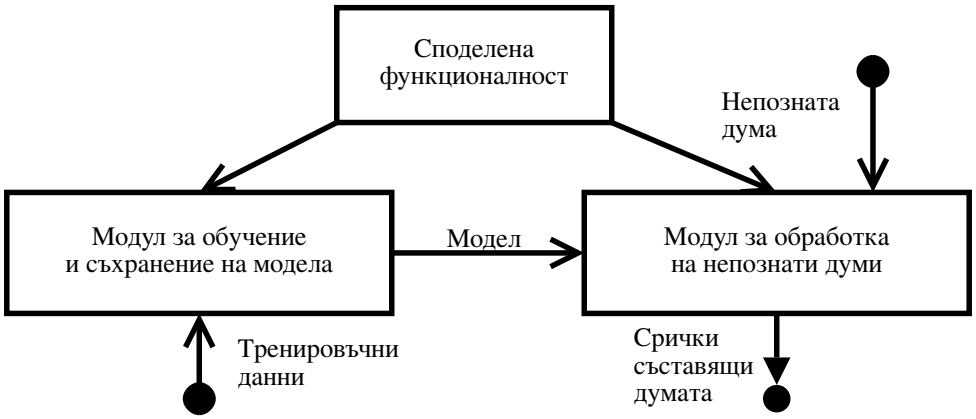
Всяка софтуерна разработка трябва да има ясна и последователна архитектура. Софтуерната архитектурата е интересен и важен аспект на всяка една приложна система. Интересен аспект, защото архитектурата няма всеобщо признато определение. Освен това, архитектурата е една от тези части на софтуера, които не може да бъде видяна, но която присъства неизменно. Архитектурата е важен аспект, защото от нея зависи колко лесно софтуерната разработка ще бъде поддържана и развивана.

Съществуват множество дефиниции за софтуерна архитектура, но може да се каже, че специалистите са съгласни с твърдението, че софтуерната архитектура представлява най-абстрактното разбиране за една система. До тук два пъти беше спомената думата “система” в контекста на софтуерната разработка, но не става ясно каква е разликата между софтуерна система и програма и в кой от двата термина се вписва имплементацията на представения в текущия реферат модел.

Софтуерната система може да бъде създадена от по-малки компоненти и обикновено може да бъде преизползвана. Например една програмна библиотека, написана на даден програмен език, представлява софтуерна

система, или накратко софтуер. Програмната библиотека няма начало и край, тя е компонент на по-голяма система.

Наличието на двата процеса, по обучение на модела и обработка на непознати думи, зависещи един от друг, подсказва, че имплементацията на модела би представлявала софтуерна система. На пръв поглед практическата реализация би съдържала поне два модула, съдържащи функционалности присъщи за двата процеса. Програмна библиотека е точното название за практическата реализация на представения модел за автоматично сричкообразуване.



Фигура 1: Диаграма на модулите, съставящи имплементацията на разработения модел.

Качествената имплементация на една програмна библиотека изисква определена дисциплина и спазването на изпитани във времето конвенции. В следващите подточки са разгледани споменатите модули за обучение на модела и този за обработка на непознати думи. Обърнато е внимание на това как модулите взаимодействат помежду си и как е структуриран приложният програмен интерфейс.

3.1 Модул за обучение и съхранение на модела

Целта на модула се припокрива с целта на процеса, който модулът реализира - изграждане на модел, който да бъде използван за “разбиване” на непознати за модела български думи до съставящите ги срички. Това значи, че модулът създава някаква същност, която впоследствие бива използвана за обработката на непознати думи. Също като процеса, така и модулът получава на входа тренировъчни данни и създава статистически модел.

Спазвайки характеристиката на *UNIX* философията за използване на чист текст като формат на входните и изходните данни, модулът приема списък от думи, всяка на нов ред, като тренировъчни данни. По този начин се улеснява както набавянето на тренировъчните данни, така и самото обучение. Съществуват няколко изисквания към думите в обучителния списък, всички думи трябва да бъдат съставени от малки букви и да бъдат премахнати думите, които се повтарят.

Структурата от данни, която е най-подходяща за съхранение на данните на разработеният модел, е така наречения речник, на английски *dictionary*. Тази структура от данни представлява съвкупност от двойки ключ и стойност. Някои от другите имена на структурата са *hashmap*, *hashtable*, *hashdict*. В трите имена се среща думата *hash* и това не е случайно. Едно от основните свойства на речника е уникалността на ключовете, които той съдържа.

Обучението и структурите от данни които се използват за съхранение на модела в паметта са важни, но ако един модел трябва да бъде използваем, то той трябва да бъде записван на диска на компютъра. Ако не бъде записван, би се наложило обучението да протича всеки път в който потребителят на програмната библиотека пожелае да използва софтуера. Модулът трябва да предоставя възможност за зареждане на обучен модел от файловата система. Зареденият модел може да се използва от модула за "предсказване" на сричките, образуващи дадена дума.

След казаното до тук, може да се изведе примерен приложен програмен интерфейс на модула. Уточнението, че интерфейсът е примерен е важно, защото неговата цел е да щрихова начина, по който трябва да изглежда модула при реализация, имайки предвид, че детайлите на реализацията не са предмет на текущия реферат. В таблица 1 нагледно е показано какви операции трябва да поддържа модулът.

Публичен интерфейс	
`train_from_file`	Обучава модел използвайки тренировъчни данни от файл и връща обучения модел.
`train_from_list`	Обучава модел използвайки списък от думи, подаден като аргумент на функцията.

`save_to_file`	Записва обучения модел в посочен от потребителя файл.
`load_from_file`	Зарежда обучен модел от посочен от потребителя файл.
Типове от данни	
`Model`	Представлява обучения модел. В зависимост от структурата от данни и парадигмата която е избрана, същността на типа може да е различна.
`SyllableMetadata`	Съдържа всички данни свързани с честотите на срещане на сричка.
Помощни функции	
`merge_metadata`	Слива две стойности от тип `SyllableMetadata` и връща като резултат стойността от операцията, която също е `SyllableMetadata`.

Таблица 1: Примерен приложен програмен интерфейс на модула за обучение на модела за автоматично сричкообразуване.

3.2 Модул за обработка на непознати думи

Модулът за обработка на непознати думи изисква наличието на обучен модел. Няма значение как модулът е възникнал, дали е зареден от файловата система, обучен и съхраняван в RAM паметта, дали е обучен от текущия потребител или е получен по електронната поща. Единственото, което свързва модула за обучение и съхранение на модела и този за обработка на непознати думи е структурата от данни, в която се съхранява моделът. Благодарение на тази независимост, отделните компоненти са лесни за подмяна и обновяване.

Приложният програмен интерфейс на модула за обработка на непознати думи трябва да предоставя една основна функционалност - “разбиване” на дадена дума до сричките, които я съставят.

Важно свойство на функционалността за “разбиване” на думите на

съставящите ги срички е възможността за лесно паралелизиране. В наши дни, компютърните процесори имат поне по две ядра и законът на *Gordon Moore* вече не е валиден (Sulov 2014), а софтуерните разработчици все по често посягат към възможностите за паралелизация. Макар подходите за паралелизация на софтуер да не са предмет на текущия реферат, важно е да се спомене, че модуларизацията, която бива разглеждана, спомага за лесната бъдещата паралелизация на представяния подход за автоматично сричкообразуване.

След казаното до тук, също както при предходния разгледан модул, може да се изведе примерен приложен програмен интерфейс на модула. От таблица 2 нагледно става ясно, че приложният програмен интерфейс покрива споменатите изисквания, които модулът покрива, и функционалността, която предоставя.

Публичен интерфейс	
`syllabify`	Използва обучен модел, подаден като аргумент, за да определи комбинацията от срички, съставящи дадена дума. Резултатът е списък от данни от тип `Syllable`.

Таблица 2: Примерен приложен програмен интерфейс на модула за разбиване на думите на съставящите ги срички.

Макар модулите да са в състояние да бъдат използвани поотделно, съществуват няколко функции, които биват използвани и от двата модула. Тези функции могат да бъдат обособени в собствен модул, но той не е предназначен за използване от потребителите на софтуерната библиотека и затова са представени по-долу, в таблица 3.

Публичен интерфейс	
`candidates`	Резултатът е списък от комбинациите срички (`CandidateList`), които е възможно да изграждат дадена дума.

`with_positions`	Приема `CandidateList` като аргумент и за всяка сричка в списъка връща двойка стойности, включваща сричката (`Syllable`) и нейната позиция (`Position`).
Типове данни	
`Syllable`	Псевдоним на тип от данни `String`.
`Position`	Изброен тип, който включва три полета - `Initial`, `Middle`, `Final`. Представя позициите, които една сричка може да заема в думата.
`CandidateList`	Псевдоним на списък от `Syllable`. Представя списък от срички, които може да съставят дума.
Помощни функции	
`substrings`	Резултатът е списък с всички възможни символни низове в рамките на дадена дума.
`filter_to_candidates`	Използвайки универсалните правила за сричкообразуване филтрира списък със срички до списък само с възможни срички, за дадена дума.
`determine_position`	Приема като аргументи сричка и индекс, където се намира в списъка с възможни срички. Като резултат връща данни от тип `Position`.

Таблица 3: Примерен приложен програмен интерфейс на модул, съдържащ функции, споделени между двата основни модула.

След казаното до тук става ясно, че както модула за обучение и съхранение на модела, така и модулът за идентифициране на сричките съставящи дадена дума предоставят на потребителите ясен приложен програмен интерфейс. Разглеждайки цялостната софтуерна система става ясно, че входните и изходните данни представляват чист текст. Чистият текст е много подходящ формат входни и изходни данни за системи, които трябва да взаимодействат с други системи. Използвайки предложената структура на модулите, както те, така и цялата система, са достатъчно самостоятелни,

за да бъдат използвани като компонент на по-големи софтуерни системи.

Заключение

Разработеният модел за автоматично сричкообразуване е все още в процес на активна разработка. Макар детайлите за начина на работа на модела да са достатъчни за базова имплементация, остават някои въпроси, които трябва да бъдат изяснени преди модела да придобие завършен вид.

Един от въпросите, които трябва да бъдат решени преди моделът да бъде определен като завършен, е свързан с подхода за третиране на *дифтонги*. Дифтонгът е комбинация от два съседни гласни звука в една сричка, където единият гласен звук е сричкообразуващ. Това определение не противоречи на дефиницията за сричка, дадено във въведението на текущия реферат. Дифтонгите са трудни за овладяване, защото идентифицирането на сричкообразуващия гласен звук е трудно. Въпреки трудностите, съществуват множество подходи за идентифициране на дифтонги, които подходи предстои да бъдат анализирани и интегрирани в разработения в текущия реферат модел.

Събирането на "груби" тренировъчни данни е друг проблем, който предстои да бъде решен. Качеството на тренировъчните данни има много голямо значение за обучението на модела. Както беше споменато в точка 2, има характеристики, които корпусът от тренировъчни данни трябва да притежава. В литературата са описани множество случаи в които се стига до създаване на модели, основани на машинно обучение, които служат за набавяне на тренировъчни данни. Тъй като разработеният модел използва честоти, при недостатъчно добре подбрани тренировъчни данни е възможно да се наблюдават големи разлики в честотите на срещане на някои срички. Това явление би повлияло негативно на качеството на обработката на непознати думи.

Макар да има няколко проблема по които предстои сериозна работа, избраният дизайн на практическата реализация на разработения модела позволява той да бъде усъвършенстван итеративно. Може да се каже, че итеративният подход за софтуерна разработка е подходящ за реализаци-

ята на научно подкрепени програмни проекти изобщо, защото позволява софтуерът да бъде актуализиран спрямо научния прогрес. Основните изисквания за започване на разработка на една софтуерна система са наличие на характеристика на входните и изходните данни, и ясна архитектура на системата.

Литература

Bartlett, S., G. Kondrak и C. Cherry (2009). „On the Syllabification of Phonemes“. В: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, с. 308—316.

Díaz-Santiago, S., L. Maria Rodriguez-Henriquez и D. Chakraborty (2014). „A Cryptographic Study of Tokenization Systems“. В: *Proceedings of the 11th International Joint Conference on e-Business and Telecommunications*. Т. 4. ICETE 2014. Vienna, Austria: SCITEPRESS - Science и Technology Publications, Lda, с. 6.

Duanmu, S. (2009). *Syllable Structure: The Limits of Variation*. Oxford linguistics. OUP Oxford.

Fowler, M. (2003). „Design - Who needs an architect?“ В: *IEEE Software* 20.5, с. 11—13.

Goldwater, S. и M. Johnson (2005). „Representational Bias in Unsupervised Learning of Syllable Structure“. В: *Proceedings of the Ninth Conference on Computational Natural Language Learning*, с. 112—119.

Goslin, J. и U. Frauenfelder (2001). „A Comparison of Theoretical and Human Syllabification“. В: *Language and Speech* 44.4, с. 409—436.

Habert, B. и др. (1998). „Towards tokenization evaluation“. В: *Proceedings of LREC*. Т. 98, с. 427—431.

He, Y. и M. Kayaalp (дек. 2006). *A Comparison of 13 Tokenizers on MEDLINE*. Техн. докл. The Lister Hill National Center for Biomedical Communications.

Huang, X., A. Acero и H.-W. Hon (2001). *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Kahn, D. (1980). *Syllable-based Generalization in English Phonology*. Outstanding dissertations in linguistics. Garland.

Marchand, Y., C. Adsett и R. Damper (2007). „Evaluating automatic syllabification algorithms for English“. В: *Proceedings of SSW6*.

— (февр. 2009). „Automatic Syllabification in English: A Comparison of Different Algorithms“. В: *Language and speech* 52, с. 1—27.

Mayer, T. (юли 2010). „Toward a Totally Unsupervised, Language-Independent Method for the Syllabification of Written Texts“. В: *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology, SIGMORPHON 2010, Uppsala, Sweden, July 15, 2010*, с. 63—71.

Müller, K. (2006). „Improving Syllabification Models with Phonotactic Knowledge“. В: *Proceedings of the Eighth Meeting of the ACL Special Interest Group on Computational Phonology and Morphology*, с. 11—20.

Reitermanov, Z. (ян. 2010). „Data splitting“. В: *WDS'10 Proceedings of Contributed Papers*, с. 31—36.

Rogova, K., K. Demuynck и D. Van Compernolle (2013). „Automatic syllabification using segmental conditional random fields“. В: *COMPUTATIONAL LINGUISTICS IN THE NETHERLANDS JOURNAL* 3, с. 34—48.

Salus, P. H. (1994). *A Quarter Century of UNIX*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Selkirk, E. (1984). „On the major class features and syllable theory“. В: *Language Sound Structure*.

Sulov, V. (май 2014). „On the Essence of Hardware Performance“. В: *Research Journal of Economics, Business and ICT* 9, с. 13—18.