Семинарска работа по предметот Линеарна алгебра и примени

Примената на Линеарната алгебра во Обработката на природните јазици

Филип Ставров, 183054

Абстракт

Линеарната алгебра претставува математичка дисциплина која се занимава со вектори и матрици, како и векторски простори и линеарни трансформации. За разлика од останатите делови на математиката кои честопати се поттикнати од нови идеи и нерешени проблеми, линеарната алгебра е многу добро и широко разбрана. Нејзината вредност лежи во нејзините многубројни примени, од математичка физика, до модерна алгебра и теорија на кодирање, а во поново време своја примена наоѓа и во машинското учење и обработката на природните јазици.

Обработката на природните јазици претставува гранка на Компјутерската наука, односно поконкретно, гранка на Вештачката интелигенција, чија главна цел е да ги зголеми можностите на компјутерите за разбирање на текст и изговорени зборови, како и релациите кои постојат помеѓу зборовите, на ист начин како и луѓето. Обработката на природните јазици комбинира компјутерска лингвистика - моделирање човечки јазик засновано на правила со статистички модели, модели на машинско учење како и модели на длабоко учење. Заедно, овие технологии им овозможуваат на компјутерите да го обработуваат човечкиот јазик во форма на текст или говорни податоци и да го "разберат" неговото целосно значење, земајќи ги во предвид намерата и чувствата на говорникот или писателот.

Вовед

Како што кажавме и претходно, Обработката на природните јазици претставува гранка од Вештачката интелигенција која е посветена на интеракцијата помеѓу компјутерите и луѓето користејќи природен јазик – во најголем дел од случаите станува збор за Англискиот јазик. Обработката на природните јазици наоѓа своја примена во апликации како што се Chat-Bots, препознавање на говор, анализа на текст, семантичка анализа и слично. Типичен пример за апликација базирана на концептите од Обработката на природните јазици е текст едиторот Grammarly.

Меѓутоа, на што всушност подлежат сите овие примени карактеристични за Обработката на природните јазици. Знаеме дека компјутерите не можат да разбираат текстуални податоци, па затоа пред да можеме да искористиме било која од техниките кои ни ги нуди Обработката на природните јазици, мораме текстуалните податоци да ги претставиме нумерички. Овде на помош доаѓа Линеарната алгебра, па во оваа семинарска ќе погледнеме на кој начин и како се одвива ваквиот процес. Покрај тоа, ќе погледнеме и како можеме да ја искористиме Линеарната алгебра со цел да ја намалиме димензионалноста на податоците со кои располагаме, имајќи предвид дека денес работиме со доста големи количини на податоци.

Вовед во Word embedding (Вградување на зборови)

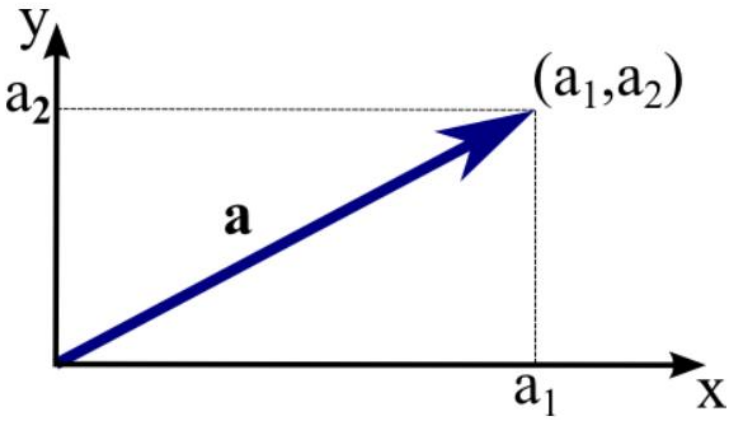
Пред да направиме детална анализа на процесот на Word embedding, да се потсетиме на термините од Линеарна алгебра кои се клучни во Обработката на природните јазици, како што се вектори и матрици.

Во Линеарна алгебра векторот претставува листа од артибутите на објект. Односно, да поедноставиме, векторот претставува листа од броеви. Овие броеви ни помагаат да ја идентификуваме положбата на дадена точка во просторот. Должината на оваа листа, односно бројот на броеви ни ја кажува

димензионалноста на просторот -> дводимензионален или повеќе-димензионален простор.

Векторите се карактеризираат со насока и големина, а ова е особено од корист кога се одредува положбата на една точка во просторот, во однос на друга.

Пример, на слика 1 можеме да погледнеме векторот a со координати (a1, a2).



Слика 1. “Вектор во дво-димензионален простор”

Кога станува збор за матрица, знаеме дека истата претставува сет од вектори. Пример доколку имаме два вектори во дво-димензионален простор во следнава форма:

2x+3y и **4x+5y**

Матрицата ќе изгледа вака

[2 3

4 5]

Сега, ако нештото има големина и правец, тоа ни отвора можност да правиме одредена математика со него. Тогаш, ако можеме да направиме одредена

математика со векторската форма на збор, тогаш имаме можност да направиме споредба на два збора во H-димензионален простор.

Алгоритмите за машинско, како и за длабоко учење не се во можност да примаат текстуални податоци. Поради тоа, ние мораме да најдеме начин да ги конвертираме овие текстуални податоци во нумерички. Па овде концептот на вектори настапува на сцена.

Word Embedding е начин на претставување на зборовите како вектори од броеви. Меѓутоа важно е да нагласиме дека во овој процес на конверзија, секој збор(или сега веќе вектор) го зачувува својот контекст во документот. Ваквите репрезентации се добиени со тренирање на различни невронски мрежи на големи количини на текстуални податоци наречени корпус. Word embedding всушност претставува јазична техника за моделирање со учење. Има повеќе техники на word embedding, а меѓу нив се издвојува Word2Vec.

Со цел да го поедноставиме ова, можеме да разгледаме пример со кој ќе ги демонстрираме можностите на математиката, кога зборовите ги разгледуваме како бројки, а не само како комбинација на букви за машинското учење.

Пример – Сличност на животни и едноставна линеарна алгебра

Во овој пример нашето внимание ќе го насочиме кон конкретно подмножество од зборови, а тоа е дел од подмножеството од зборови за животни. Нашата задача е да ја најдеме сличноста помеѓу овие зборови и суштествата кои тие всушност ги отсликуваат. За да го направиме ова, ќе дадеме табеларен приказ на животните и на дел од нивните карактеристики.

	cuteness (0–100)	size (0–100)
kitten	95	15
hamster	80	8
tarantula	8	3
puppy	90	20
crocodile	5	40
dolphin	60	45
panda bear	75	40
lobster	2	15
capybara	70	30
elephant	65	90
mosquito	1	1
goldfish	25	2
horse	50	50
chicken	25	15

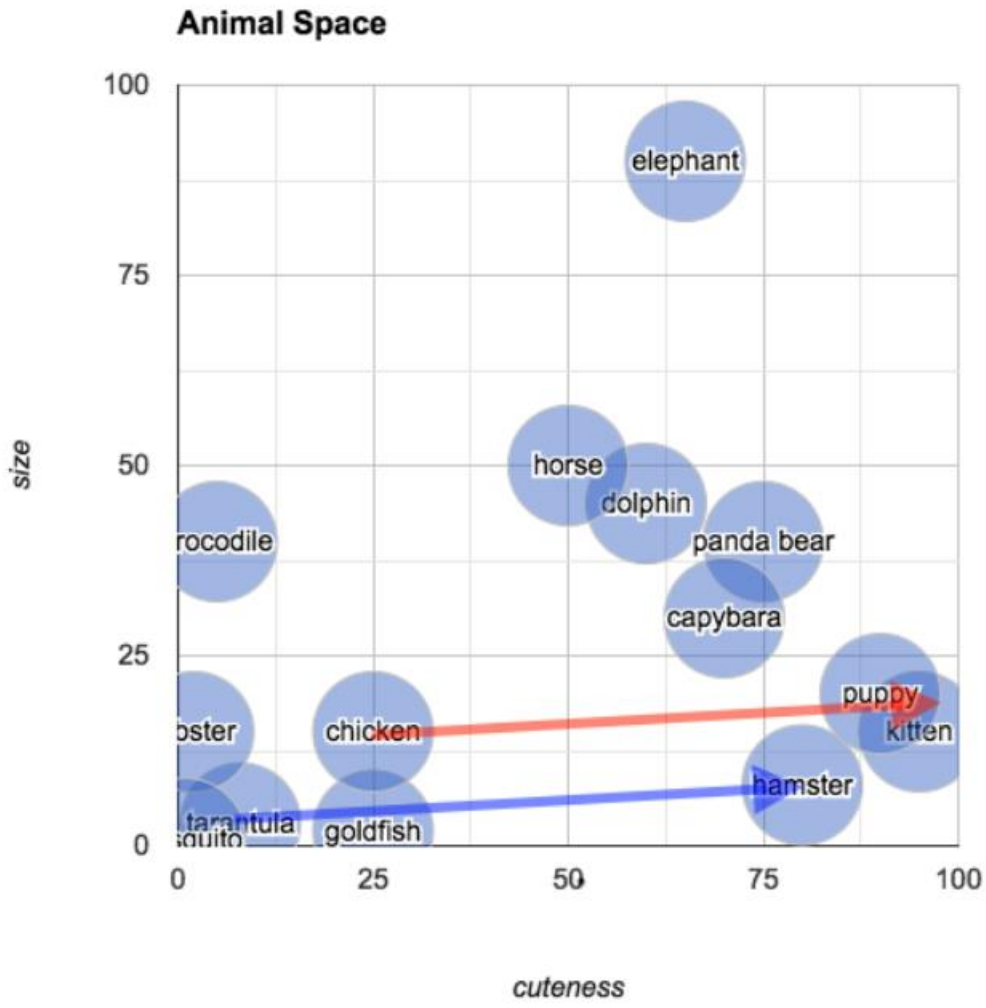
Слика 2. “Табеларен приказ на животни и нивни карактеристики”

Доколку ја разгледаме оваа табела, можеме да видиме дека секое животно го опишуваме со помош на два броја.

- Првиот број ни го дава нивото на слаткост на животното и истото прима вредност во интервалот од 0 до 100.
- Вториот број ни ја означува големината на животното, која исто така може да прима вредности во интервалот од 0 до 100.

*** вредностите на овие две карактеристики се земени случајно*

Имајќи ги овие две карактеристики со кои го опишавме секое од животните, лесно можеме секое од животните да го претставиме во дво-димензионален простор. Притоа, една од оските ќе ни ја означува големината на животното, а другата оска, неговата слаткост. Со поврзување на овие две бројки, ќе добиеме точка во просторот, која го ошишува секое животно, гледајќи од аспект на овие карактеристики.



Слика 3. “Приказ на животните во дводимензионален простор”

Овој начин на приказ на животните нуди можност лесно да ги испитуваме врските кои постојат меѓу нив.

- Пример, можеме да прашаме, кое животно е најслично до куче? Па доколку ја разгледаме сликата број 3, ќе забележиме дека одговорот на ова прашање е мече. Односно, корелацијата на големина и слаткост помеѓу куче и мече е многу слична. Еден од начините за да го пресметаме растојанието од едно до друго животно е да користиме Евклидово растојание.
- Следно, можеме да забележиме дека на дијаграмот има воспоставено две врски:

- Едната од кокошка кон мече
- Другата од тарантула кон хрчак

Овие стрелки ни опишуваат две животни со иста големина, меѓутоа едните од нив се послатки од другите. Па така животно кое е на слична големина со кокошка, но е многу послатко е.... мече.

Овој едноставен пример ни ја прикажа основната идеја, зошто е потребно да ги конвертираме зборовите во вектори, па следно можеме да преминеме кон како тоа добиваме вектор од збор во N-димензионален простор.

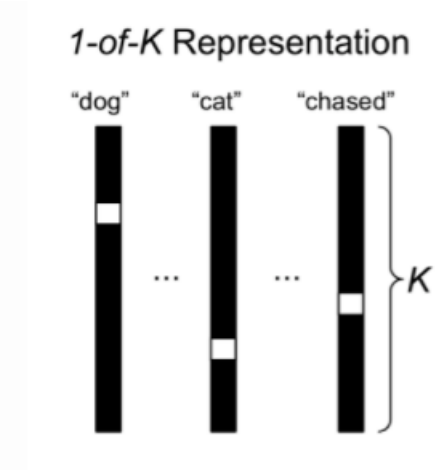
Мотивација за користењето на Word Embeddings

Веќе знаеме дека за компјутерите да можат да обработуваат зборови од природниот јазик, кои имаат своја семантика и свое значење, потребно е истите да бидат нумерички претставени.

Класичен начин за нумеричко претставување на зборот е користењето на "1-од-К" или таканаречената "One-hot" кодирање. Ова кодирање користи "sparse" вектор (вектор со многу нули) со должина К за претставување на секој од зборовите во вокабулар со истата големина. Секој од векторите е составен од "К-1" нули и една единствена единица, која се наоѓа на индексот кој е поврзан со дадениот збор.

Ваквиот начин на кодирање на зборовите е едноставен и обезбедува ортогонално можество на карактеристики за претставување на зборовите. Овој начин на кодирање претставувал столб на многу модели за Обработка на природните јазици со децении. Меѓутоа, ова "1-од-К" кодирање може да биде неефикасно, со оглед на тоа што кардиналноста на просторот на карактеристики може да биде многу голема за големи вокабулари, од каде следува така наречената "curse of dimensionality" односно проклетството на димензионалноста што предизвикува

лоши проценки на параметрите за многу алгоритми на машинско учење. Пример за ваквиот начин на кодирање имаме на сликата во продоложение.

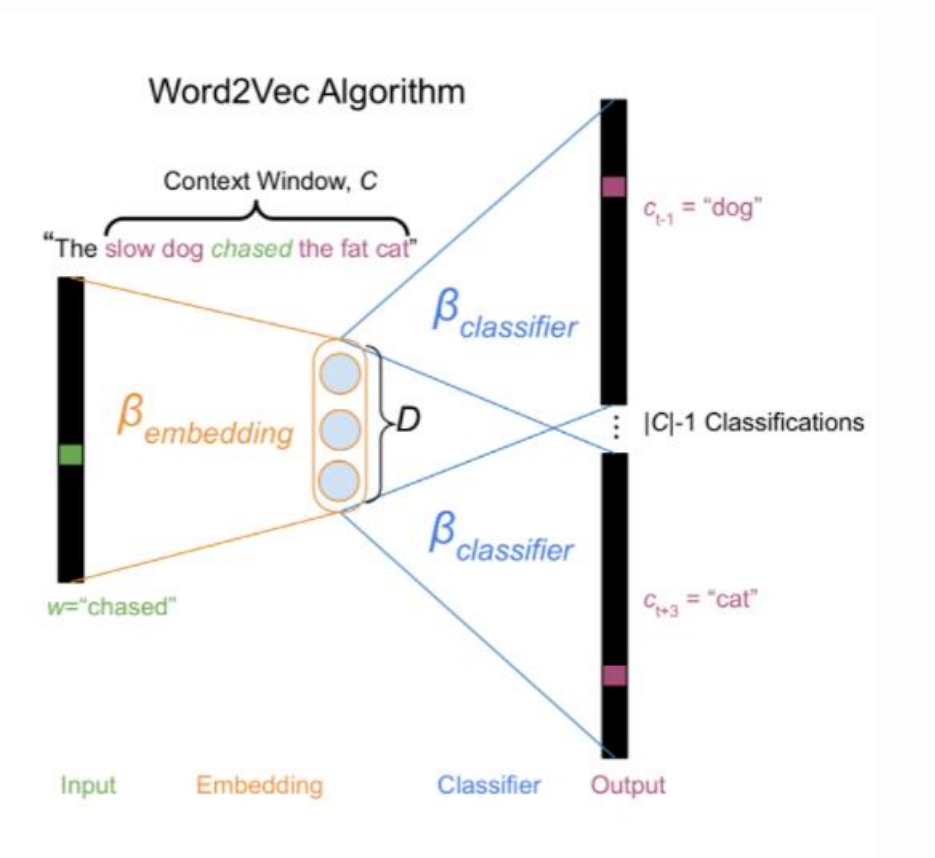


Слика 4. “One hot кодирање”

Во последниве години се почесто се користи кодирањето познато како "word embedding" (вградување зборови). Овој начин на кодирање носи повеќе семантички информации и за разлика од "one-hot" кодирањето кое има големи и ретки вектори, "word embedding" кодирањето се карактеризира со густи вектори чија должина е неколку пати стотици пати помала од векторите кои ги добиваме со "one-hot" кодирањето.

Постојат повеќе методи и алгоритми за ваквото вградување на зборови, меѓутоа најчесто употребуван е алгоритмот Word2Vec.

Основата на овој алгоритам лежи во тренирањето на невронска мрежа која е оптимизирана на корпус реченици. Па така, за даден збор, земен како примерок од една од реченициte во корпусот, задачата на невронската мрежа е да ги предвиди останатите зборови кои се лоцирани во рамки на контекстниот прозорец C, опкружувајќи го зборот избран како примерок.

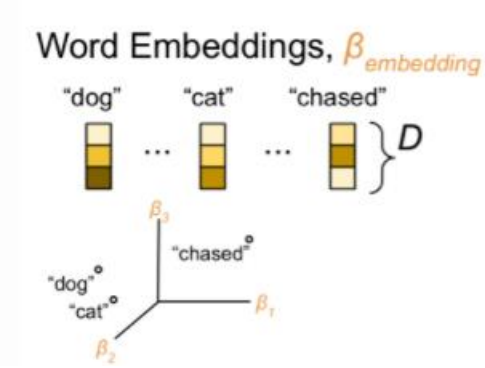


Слика 5. "Word2Vec алгоритам"

На слика број 5 можеме да погледнеме двослојна невронска мрежа за предвидување на зборовите во контекстниот прозоцер C, во зависност од избраниот збор, w.

На влез на невронската мрежа го имаме избраниот збор, w, претставен како вектор добиен со "one-hot" кодирање, како и зборовите во контекстниот прозорец, исто така претставени како "one-hot" вектори. Скриениот слој на оваа невронска мрежа е составен од D единици(делови), односно големината која ние сме ја одредиле за густите вектори кои сакаме да ги добиеме на излез. Овде ја добиваме матрицата на параметри $B_{embedding}$ KxD, која линеарно го мапира секој од ретките вектори со големина K, во густ вектор со големина D, каде D<<K. На крајот од конвергенцијата на оваа невронска мрежа на излез имаме густ вектор со големина D, за секој од зборовите во реченицата.

Докажано е дека ваквиот начин на кодирање на зборови во вектори е доста семантички богат. Притоа, зборовите кои се семантички блиски, имаат слични локации во D-димензионалниот простор. Ова можеме да го видеме на слика број 6, каде куче и мече се наоѓаат едно до друго бидејќи и двете се животни.



Слика 6. “Крајни вектори добиени за секој од зборовите”

Пресметување информациско-теоретски вградувања на зборови со Singular Value Decomposition (SVD) – Разделување по сопствени вредности

Пресметката на "word embeddings" користејќи го Word2Vec алгоритамот побарува градење и тренирање на невронска мрежа, што пак побарува големо знаење на калкулус, кој е неопходен за оптимизацијата на параметрите кои се базирани на градиент. Сепак, постои и поедноставен начин да се пресметаат еквивалентни вектори на зборови и тоа само со користење на теорија на информации и линеарна алгебра.

Пред да навлеземе во деталите на ваквиот метод, да се потсетиме на некои од основните концепти.

- **Маргинални и заеднички веројатности**

Веројатностите претставуваат основа за теоријата на информациите. Од големо значење за нас се моментално маргиналните и заедничките веројатности.

- Маргинална веројатност

Под маргинална веројатност на даден збор(w_i), p(w_i) во корпус од реченици, се подразбира бројот на појавувања на дадениот збор, N(w_i) -> познато и како “unigram frequency”, поделен со вкупниот број на појавувања на зборовите во корпусот од реченици $\sum_K N(w_K)$.

$$p(w_i) = \frac{N(w_i)}{\sum_k N(w_k)}$$

Слика 7. “Маргинална веројатност”

- Заедничка веројатност

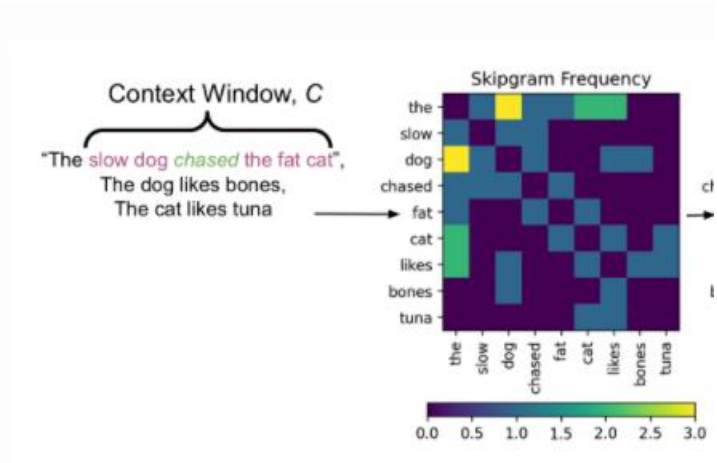
Заедничката веројатност на два збора w_i и w_j , каде i≠j, е бројот на пати каде овие два збора се појавуваат заедно (co-occurrence) $N(w_i,w_j)$, поделен со вкупниот број на појавувања на зборовите $\sum_K N(w_K)$.

$$p(w_i, w_j) = \frac{N(w_i, w_j)}{\sum_k N(w_k)}$$

Слика 8. “Заедничка веројатност”

Постојат повеќе дефиниции за “co-occurrence” (соседно појавување), меѓутоа во нашиот конкретен случај ќе се користиме

т.н. “skip-gram frequencies”. Skip-grams дефинираат заедничка функција на фреквенција, $N(w_i,w_j) = N(w_i, c_{t+-l})$, како бројот на пати на повторување на контекстниот збор c_{q+-l}, се појавува во контексниот прозоцер C, кој го опкружува тергетираниот/избраниот збор w_i. t овде ни го означува индексот на избраниот збор, а l бројот на чекори кои претходат или следат на следниот избран збор во контексниот прозорец. Ова е прикажано на сликата во продолжение.



Слика 9. “Skipgram Frequency”

- **Pointwise Mutual Information**

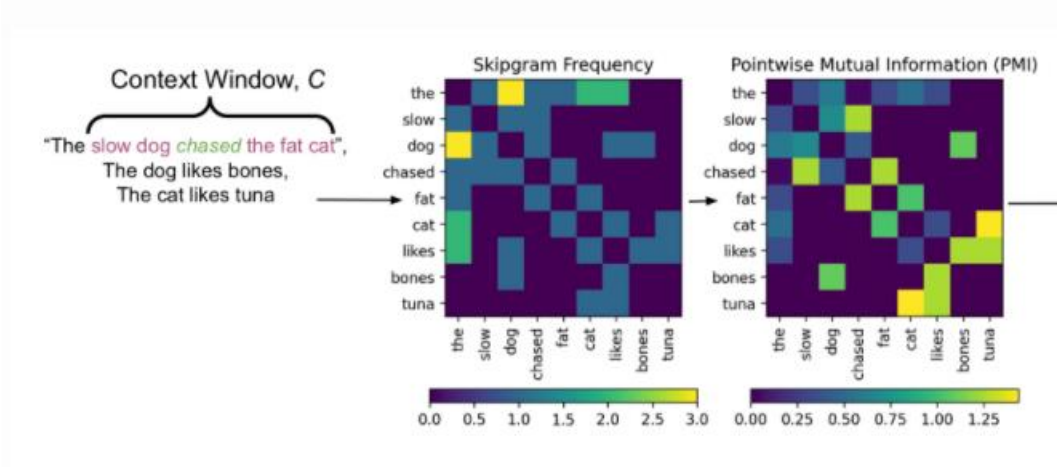
Имајќи ги маргиналните и заедничките веројатности, ние имаме можност да пресметаме голем број на моќни информатичко-теоретски квантитети. Меѓу нив се издвојува т.н. Pointwise Mutual Information. Равенката за истата е прикажана на сликата во продолжение.

$$PMI(w_i, w_j) = \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)}$$

Слика 10. “Равенка за пресметка на PMI”

PMI матрицата ни дава можност за многу интиутивен и лесен начин за пресметка на асоцијациите и зависностите помеѓу зборовите во корпусот. Секој ред од матрицата ни ја дава количината на информации кои се споделени помеѓу избран збор и останатите зборови во корпусот. Да поедноставиме, оваа матрица ни ја дава количината на зависност/асоцијација помеѓу два збора. Доколку два збора се независни, односно не се поврзани, тогаш PMI има вредност 0.

Гледано од перспектива на пресметки, PMI претставува само логаритам од заедничката веројатност на двата збора, откако истата е нормализирана со маргиналните веројатности на зборовите. Ваквото нормализирање на заедничката веројатност со производот на маргиналните веројатности на зборовите придонесува за повеќе нијансирана претстава за нивната поврзаност, споредено со сировите фреквенции за поврзаност. Процесот на премин од Skipgram Frequency во PMI можеме да го видиме на слика број 11.



Слика 11. ”Премин од Skipgram Frequency во Pointwise Mutual Information”

- **Information-theoretic Word Embeddings**

PMI матрицата претставува квадратна матрица со големина KxK. Можеме да увидиме дека доколку имаме голем корпус, т.е вокабулар, големината на ваквата PMI матрица може да биде многу голема, а најверојатно е дека истата ќе биде и ретка. Затоа следен чекор во рамки на овој процес е Singular Value Decomposition (SVD), со цел да ја компресираме ваквата голема матрица. Доколку примениме SVD на оваа PMI матрица, користејќи приближување со низок ранг, каде D<<K, тогаш всушност можеме да пресметаме компактна репрезентација за информацијата за самиот збор која е заробена во PMI матрицата. Пред да продолжиме понатаму, да го разгледаме во детали процесот на Singular Value Decomposition.

Овде, покрај употребата на Линеарната алгебра во кодирањето на зборови, доаѓа и нејзината друга примена, а тоа е намалување на димензионалноста. Знаеме дека ние денес живееме во свет кој е опкружен со масовни количини на податоци, податоци кои е потребно да бидат обработени, анализирани и складирани. Податоци со голем број на карактеристики, пример податоци за слика, или видео се преведуваат во огромни матрици од броеви. Справувањето со вакви огромни матрици претставува предизвик дури и за суперкомпјутерите. Поради тоа, потребно е да ги намалиме оригиналните податоци во помало подмножество од истите, се со цел да ги издвоиме они кои се најрелевантни.

Најпознат пристап за намалување на димензионалноста е Singular Value Decomposition, односно разделување по сопствени вредности. SVD претставува метод за разделување на матрицата на нејзините основни делови, се со цел да се направат пресметките поедноставни. SVD претставува основниот коцепт на техниката на Latent Semantic Analysis (LSA), т.е. Латентна семантичка анализа, техника што се користи во т.н. моделирање на теми. LSA претставува ненабљудувана ML (Machine Learning - Машинско учење) техника, што се користи за да се совпаднат зборовите со теми низ различни текстуални документи. Во Обработката на природните јазици, темите ги претстатуваме

како групи од синоними - поврзани зборови. Ваквиот модел на теми, ги анализира различните теми и нивните појавувања и фреквенции, како и фреквенциите на различните зборови кои тие ги содржат.

Во продолежние ќе погледнеме како со процесот на разделување по сопствени вредности ќе постигнеме намалување на димензионалноста на матриците, односно нивно разделување на основни матрици, а воедно како и истото ќе го искористиме во процесот на кодирање на зборови.

- **Singular Value Decomposition (SVD)**

Процесот на Singular Value Decomposition всушност подлежи на процесот на дијагонализација на матриците. Знаеме дека процесот на дијагонализација на една матрица всушност претставува разделување на матрицата M на две посебни матрици, P и D, такви што:

$$M_{m\times m} = P_{m\times m} D_{m\times m} P^{-1}_{m\times m}$$

Слика 12. “Формула за дијагонализација на матрица”

Каде P е инверзибилна (квадратна матрица, која има своја инверзна матрица; Детерминантата на инверзибилна матрица е секогаш различна од 0), а D е дијагонална матрица (матрица каде сите елементи се нули, освен оние по главната дијагонала).

Најчест начин за дијагонализација на матрица е да се пресмета сопствената вредност на декомпозиција на матрицата и тоа како:

$$M = PDP^{-1}$$
$$M(P) = PDP^{-1}(P)$$
$$MP = PD.$$

Слика 13. “Процес на дијагонализација”

Равенката дадена на слика 13 е всушност еквивалентна на наоѓање на сопствените вектори a_i како и сопствените вредности λ_i на матрицата M, каде:

$$P = [a_1, a_2, \dots a_m]$$
$$D = \begin{bmatrix} \lambda_1 & 0 & \dots & \dots \\ 0 & \lambda_2 & 0 & \dots \\ \vdots & \dots & \ddots & \dots \\ 0 & \dots & 0 & \lambda_m \end{bmatrix}$$
$$PD = [\lambda_1 a_1, \lambda_2 a_2, \dots \lambda_m a_m]$$

Слика 14. “Сопствени вредности и сопствени вектори”

Па така, со решавање на сопствените вредности и вектори за M, ќе ги добиеме потребните компоненти за дијагонализација на матрицата. Меѓутоа, сеуште ни останува да ја пресметаме вредноста на P^{-1}.

Процесот на дијагонализација има голем број на нумерички и пресметливи погодности. Меѓу нив се лесната пресеметка на инверзна матрица, повторување на математички проблеми во нов, канонски систем, каде што одредени операции или структури имаат погодни толкувања.

Меѓутоа, гледајќи ја дефиницијата за дијагонализација дадена погоре, можеме да заклучиме дека за една матрица да може да се дијагонализира, истата мора да биде квадратна и инверзибилна. Иако, има голем број на интересни проблеми кои обработуваат само квадратни матрици, има уште повеќе сценарија каде се сретнуваат и матрици кои не се квадратни. Овде

на сцена настапува Singular Value Decomposition – разделување по сопствени вредности.

Singular Value Decomposition можеме да го гледаме како генерализација на дијагонализацијата за неквадратни матрици. Од ова следува дека всушност сите матрици имаат SVD решение.

Основата на Singular Value Decomposition лежи воа тоа дека за секоја матрица M, матриците M^TM и MM^T се симетрични. Ова можеме да го докажеме и испишеме на следниот начин:

$$\mathbf{(M^TM)^T = (M^T)^TM^T = MM^T}$$

Истиот принцип е и ако тргнеме од обратната страна, т.е:

$$\mathbf{(MM^T) = M^T(M^T)^T = M^TM}$$

Покрај тоа, SVD ја користи предноста дека сите симетрични матрици, како што се M^TM и MM^T имаат сопствени вредности кои формираат ортонормална основа. Со овие два поими во предвид, најпрво можеме да го дефинираме SVD, а потоа и да ги извлечеме неговите компоненти од матриците M^TM и MM^T.

Целта на SVD е да ја раздели матрицата M_{mxn} во три различни матрици и тоа:

$$M_{m\times n} = U_{m\times m}S_{m\times n}V^T_{n\times n}$$

Слика 15. “SVD дефиниција за mxn матрица M”

Каде:

- U -> е ортогонална матрица -> $(UU^T) = I$ – единечна матрица
- V -> е ортогонална матрица -> $(UU^T) = I$ – единечна матрица
- S -> е дијагонална матрица

За да ја изведеме U, ќе направиме анализа на симетричната матрица M^TM, користејќи ја дефиницијата за SVD дадена на слика број 15.

$$\mathbf{M^TM = (USV^T)^T(USV^T)}$$

$$\mathbf{M^TM = (VS^TU^T)(USV^T)}$$

$$\mathbf{M^TM = VS^TU^TUSV^T = VS^TISV^T, \text{ бидејќи U е ортогонална}}$$

$$\mathbf{M^TM = VS^TSV^T}$$

$$\mathbf{\underline{M^TM = VS^TSV^{-1}}, \text{ бидејќи V е ортогонална}}$$

Оваа равенка е во суштина друга операција за дијагонализација како онаа дефинирана на слика број 12 (равенката за дијагонализација на квадратна матрица). Разликата е во тоа што овде правиме дијагонализација на матрицата MTM, наместо M, и имаме дијагонализирачка матрица STS, наместо D.

Како што прикажавме во равенката прикажана на слика 15, ваквата дијагонализација може да се реши преку разложување до сопствени вредности, што ги сугерира следниве две својства на SVD:

- *Колоните на матрицата V се сопствените вектори на матрицата M^TM.*
- *Бидејќи матрицата S^TS ги дава сопствените вредности на матрицата M^TM, дијагоналата на матрицата S ги содржи корените на овие сопствени вредности.*

Ги пронајдовме V и S, но што е со U?

За да го изведеме U, правиме слични пресметки како и за V, но сега наместо M^TM, овие пресметки ќе ги правиме на MM^T.

$$\mathbf{MM^T = (USV^T)(USV^T)^T}$$

$$\mathbf{MM^T = (USV^T)(VS^TU^T)}$$

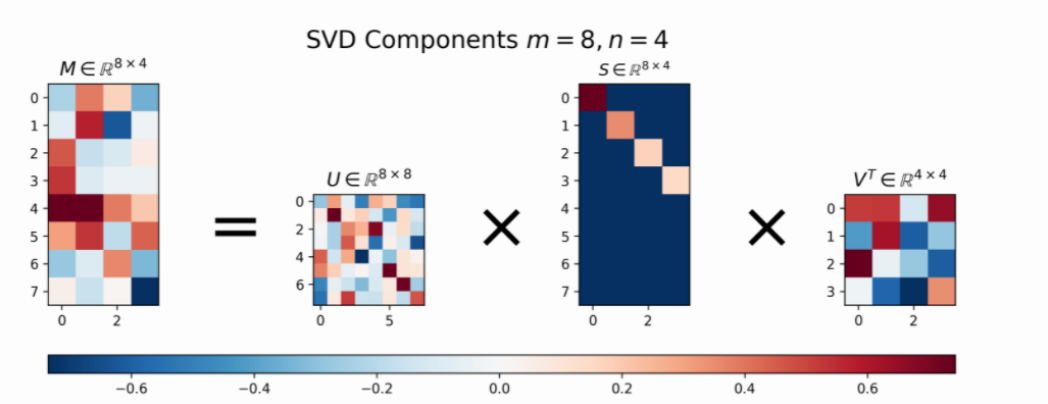
$MM^T = USV^TVS^TU^T = USIS^TU^T$, бидејќи V е ортогонална

$MM^T = USS^TU^T$

$\underline{MM^T = USS^TU^{-1}}$, бидејќи U е ортогонална

Оваа равенка ни сугерира и трето својство на SVD, а тоа е дека колоните на матрицата U се сопствени вектори на матрицата MM^m. Значењето на матрицата S, останува исто како и во горната равенка.

Важно е да нагласиме дека, кога матрицата не е квадратна, дијагонализирачката матрица S нема да биде квадратна, како што беше тоа случајот со D. Наместо тоа, матрицата S ќе биде зголемена во зависност од поголемата димензија на редици или колони.

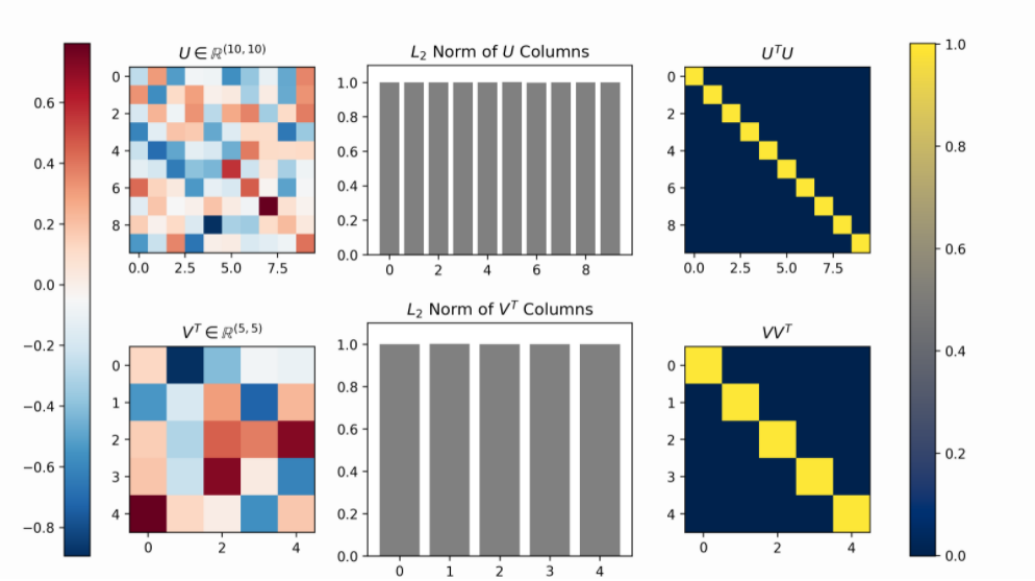


Слика 16. “Визуелизација на U, S и V за матрица која не е квадратна”

На сликата погоре можеме да ги погледнеме резултатите од SVD на матрица каде бројот на редици е поголем од бројот на колони. Можеме да видиме дека матрицата S, покрај тоа што има дијагонални опаѓачки елементи, истата не е квадратна. Наместо тоа, поради поголемиот број на редови на матрицата M, истата е дополнета со дополнителни редови исполнети со нули.

Формулацијата на SVD гарантита дека колоните на матриците U и V формираат ортонормална основа. Ова значи дека сите вектори на колони во една матрица се ортогонални и имаат иста должина. Ова е еквивалентно на тврдењето дека

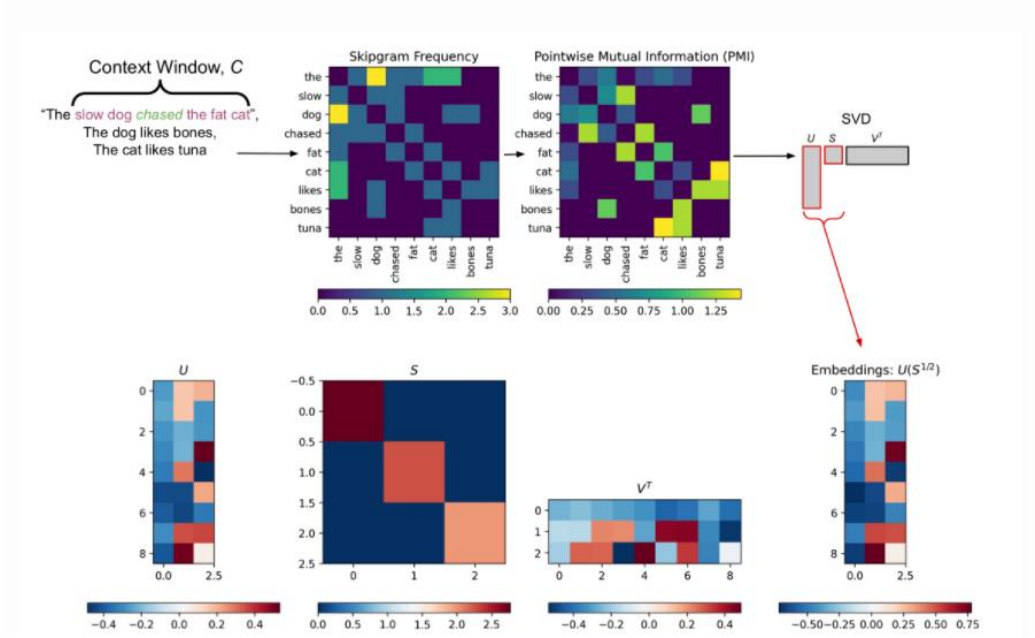
внатрешниот производ на секоја од матриците а со својата транспонирана матрица ќе формира идентична(единечна) матрица, како и дека L_2 нормата на секоја колона ќе биде еднаква на еден. Овие две тврдења се прикажани на сликата во продолжение, а истите се прикажани преку SVD на матрица со димензии 10x5.



Слика 17. “Приказ на матрици по извршен SVD на матрица со димензии 10x5.”

Можеме да видиме дека навистина нормите на сите вектори на колони на матриците U и V се еднакви на 1, како и дека внатрешниот произовод на овие матрици со нивните транспонирани води до соодветно 10x10 и 5x5 единечни матрици, што индицира дека матриците U и V претставуваат ортонормална база.

Сега да се навратиме на преминот од Pointwise Mutual Information кон Singular Value Decomposition и да го погледнеме истиот во детали. Целосниот процес е прикажан на сликата во продолжение.



Слика 18. “Information-theoretic Word Embedding со PMI и SVD”

Униграм фреквенциите, како и KxK матрицата на скипграм фреквенции се пресметави врз база на корпусот на реченици и предефинираниот контекстен прозорец C. Во овој конкретен пример, вредноста на K=9, и тоа ја означува големината на вокабуларот во корпусот. Овие фреквенции се користени за да се пресмета PMI матрицата користејќи ја равенката во продолжение која ја објаснавме претходно.

$$PMI(w_i, w_j) = \log \frac{p(w_i, w_j)}{p(w_i)p(w_j)}$$

Слика 19. “Равенка за пресметка на PMI”

На матрицата добиена со PMI применуваме SVD, каде D<<K и вредноста на D во конкретниот случај е 3. По ваквиот чекор добиваме сопствени вектори од низок ранг U, како и сопствени вредности S. Сопствените вектори од низок ранг се рескалираат со квадратниот корен на сопствените вредности, со цел да добиеме копмресирана репрезентација на PMI матрицата чии димензии ќе бидат KxD.

Секој ред од оваа матрица од низок ранг ни дава вектор на кодирање (embedding vector) за секој од K-те зборови во вокабуларот (Ова е матрицата доле десно).

Овој информатичко-теоретски, или уште наречен и линеарно-алгебарски метод обезбедува "word embeddings" аналогни на оние кои се добиваат со употреба на Word2Vec. Ваквиот начин на кодирање е исто така богат со семантички информации, а воедно и сличните зборови имаат слични локации во т.н. "embedding space", а нивната насоченост во просторот го пренесува семантичкото значење.

Ваквите пристапи со употреба на SVD, директно на т.н. "co-occurrence" матрици биле користени уште во 1990тите и тоа во алгоритми како Latent Semantic Indexing а да обезбедат word ebmeddings. Меѓутоа со појавата на длабокото учење, забележано е намалување на употребана на ваквите методи.

```python
#Import needed library
import spacy
from sklearn import svm
#Loading NLP dictionary from Spacy
nlp = spacy.load("en_core_web_md")

#Build a class for categories
class Category:
    BOOKS = "BOOKS"
    CANDY = "CANDY"
#Train data
train_x = ["i love the book", "this is a great book", "this tastes great", "i love the chocolate
train_y = [Category.BOOKS, Category.BOOKS, Category.CANDY, Category.CANDY]

#Creat words vectors
docs = [nlp(text) for text in train_x]
train_x_word_vectors = [x.vector for x in docs]

#Match vectors to training data
clf_svm_wv = svm.SVC(kernel='linear')
clf_svm_wv.fit(train_x_word_vectors, train_y)

#Test new data
test_x = ["I went to the store and bought gummies", "let me check that out"]
test_docs = [nlp(text) for text in test_x]
test_x_word_vectors =  [x.vector for x in test_docs]
clf_svm_wv.predict(test_x_word_vectors)
```

Слика 20. “Краток код за word embeddings”

На слика број 20 е прикажан едноставен код во Python, кој вклучува Word2Vec кодирање на зборови. Притоа имаме вклучување на библиотеките Spacy и Sklearn.

Заклучок

По направената анализа на поврзаноста помеѓу Обработката на природните јазици и Линеарната алгебра можеме да кажеме дека дека Линеарната алгебра претставува солидна основа за градење на голем број од алгоритмите употребувани во Обработката на природните јазици, но и во многу друго обласни опфатени од Машинското и Длабокото учење.

Најголема примена на термини, теореми и докази од Линеарната алгебра се среќаваат во процесот на кодирање на зборови, т.н. word embeddings, како и во процесот на намалување на димензионалноста.

Со помош на знаењата од Линеарната алгебра, ние можеме полесно да ги претставиме дадените зборови како вектори, а потоа и да ги претставиме истите во просторот. Можеме лесно да ја толкуваме нивната поврзаност, како и лесно да ги извршуваме операциите кои се дозволени со нив – бидејќи сепак, тоа претставуваат операции со вектори, кои се детално опфатени во Линеарната алгебра.

Исто така, знаењето за матрици кое го имаме од Линеарната алгебра, ни помага во процесот на намалување на димензионалноста, како и во самиот процес на word embeddings. Знаејќи ги разните видови на матрици, како и правилата, законите и операциите кои важат за нив, без проблем ќе успееме да се снајдеме и со доста сложени, големи и ретки матрици.

Од направената анализа на самите процеси на кодирање на зборовите, можеме да установиме дека математичката основа добиена од Линеарната алгебра во големо го олеснува и убрзува овој процес на кодирање, а покрај тоа, ни помага и

во полесно претставување на векторите(зборовите) визуелно во просторот, независно од неговата димензионалност.

Користена литература

[1] 5 Applications of Linear Algebra in Data Science - https://towardsdatascience.com/5-applications-of-linear-algebra-in-data-science-81dfc5eb9d4

[2] 10 Powerful Applications of Linear Algebra in Data Science (with Multiple Resources) - https://www.analyticsvidhya.com/blog/2019/07/10-applications-linear-algebra-data-science/

[3] Understanding the role of vectors in natural language processing - https://medium.com/analytics-vidhya/understanding-the-role-of-vectors-in-natural-language-processing-55f1632e17ff

[4] An implementation guide to Word2Vec using NumPy and Google Sheets - https://towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281

[5] Linear Algebra for Natural Language Processing - https://towardsdatascience.com/from-linear-algebra-to-text-representation-for-natural-language-processing-239cd3ccb12f

[6] Deep Dive Into Word2Vec - https://medium.com/analytics-vidhya/deep-dive-into-word2vec-7fcefa765c17

[7] Introduction to Word Embedding and Word2Vec - https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

[8] Who Needs Backpropagation? Computing Word Embeddings with Linear Algebra - https://dustinstansbury.github.io/theclevermachine/info-theory-word-embeddings

[9] Singular Value Decomposition: The Swiss Army Knife of Linear Algebra - https://dustinstansbury.github.io/theclevermachine/singular-value-decomposition