Aplikace pro doporučování podobných veřejných zakázek

## Historie dokumentu

|  |  |  |
| --- | --- | --- |
| Datum | Autor | Změna |
| - | Milan | Inicializace (abstrakt, data, techstack) |
| 8.1.2020 | Milan | Update (abstract, data, techstack) |
| 8.1.2020 | Milan | Změna struktury a přidání obsahu |
| 8.1.2020 | Milan | Přidání logického plánu |
| 8.1.2020 | Milan | Přidání časového plánu |
| 10.1.2020 | Milan | Doplnění textu k jednotlivým kapitolám |
| 15.1.2020 | Milan | Přidání Treexu u FEA04 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Autor: Milan Vancl

Vedoucí: Jaroslav Kuchař

Oponent: Marek Sušický

Obsah

[Historie dokumentu 1](#_Toc29571919)

[Abstract 4](#_Toc29571920)

[Úvod 4](#_Toc29571921)

[Data 4](#_Toc29571922)

[Technologie 4](#_Toc29571923)

[Plán 5](#_Toc29571924)

[Logický pohled 5](#_Toc29571925)

[Časově sousledný pohled 6](#_Toc29571926)

[STH00 – Stakeholders 7](#_Toc29571927)

[STH01 – Průzkum uživatelů a jejich požadavků 7](#_Toc29571928)

[DAT00 - Data 7](#_Toc29571929)

[DAT01 - Dokumenty veřejných zakázek 7](#_Toc29571930)

[DAT02 - Kategorie produktů a služeb 8](#_Toc29571931)

[DAT03 - Katalog produktů a služeb 8](#_Toc29571932)

[DAT04 – Data o lokacích 8](#_Toc29571933)

[FEA00 - Extrakce feature z dokumentů 8](#_Toc29571934)

[FEA01 - Klasifikace dokumentů 8](#_Toc29571935)

[FEA01a - Výběr metody a algoritmu 8](#_Toc29571936)

[FEA01b - Výběr technologie 8](#_Toc29571937)

[FEA01c - Návrh procesu učení 9](#_Toc29571938)

[FEA01d - Učení modelu 9](#_Toc29571939)

[FEA01e -Sestavení komponenty pro klasifikaci 9](#_Toc29571940)

[FEA02 - Document embedding 9](#_Toc29571941)

[FEA02a - Výběr metody a algoritmu 9](#_Toc29571942)

[FEA02b - Výběr technologie 9](#_Toc29571943)

[FEA02c - Sestavení komponenty pro extrakci embeddingu 9](#_Toc29571944)

[FEA03 - Metrika podobnosti dokumentů 9](#_Toc29571945)

[FEA03a - Výběr algoritmu 9](#_Toc29571946)

[FEA03b - Výběr technologie 9](#_Toc29571947)

[FEA03c - Sestavení komponenty pro porovnávání dokumentů 9](#_Toc29571948)

[FEA04 - Extrakce předmětu 10](#_Toc29571949)

[FEA04a - Výběr metody a algoritmu 10](#_Toc29571950)

[FEA04b - Výběr technologie 10](#_Toc29571951)

[FEA04c – Návrh procesu extrakce předmětu 10](#_Toc29571952)

[FEA04d - Sestavení komponenty pro extrakci předmětu 10](#_Toc29571953)

[REC00 - Doporučovací systém 10](#_Toc29571954)

[REC01 - Výběr doporučovací metody 10](#_Toc29571955)

[REC02 - Výběr technologie 10](#_Toc29571956)

[REC03 - Návrh systému 10](#_Toc29571957)

[REC04 – Sestavení 10](#_Toc29571958)

[APP00 - Aplikace 10](#_Toc29571959)

[APP01 - Výběr technologie 10](#_Toc29571960)

[APP02 - Sestavení prototypu 11](#_Toc29571961)

[APP03 - Integrace s ostatními komponentami 11](#_Toc29571962)

[APP04 - Nasazení? 11](#_Toc29571963)

[DIP00 - Psaní diplomového dokumentu 11](#_Toc29571964)

# Abstract

Návrh, sestavení a naučení modelu pro extrakci předmětu (produkt, služba) z textu dokumentace veřejných zakázek.

Vytvoření aplikace pro vyhledávání (doporučování) podobných zakázek podle předmětu, lokace, atd.

# Úvod

## Data

Data pro učení modelu – otevřená data veřejných zakázek v ČR. Implikuje český jazyk – nutnost naučení modelu NLP pro češtinu.

Dataset pro učení klasifikátoru – opendata VZ + olabelování (Hlídač Státu, @Michal Bláha)

## Technologie

NLP – python, spaCy?, deeppavlov

Download dat – java (projekt public-contracts)

Infrastruktura – vm-opendata (hlavní databáze, integrace), lokální dev. Prostředí, prof-gpu (výpočetní server)

Aplikace: web app (python(django/flask))

# Plán

## Logický pohled

1. STH00 – Stakeholders
   1. STH01 – Průzkum uživatelů a jejich požadavků
2. DAT00 - Data
   1. DAT01 - Dokumenty veřejných zakázek
   2. DAT02 - Kategorie produktů a služeb
   3. DAT03 - Katalog produktů a služeb
   4. DAT04 – Data o lokacích
3. FEA00 - Extrakce feature z dokumentů
   1. FEA01 - Klasifikace dokumentů
      1. FEA01a - Výběr metody a algoritmu
      2. FEA01b - Výběr technologie
      3. FEA01c - Návrh procesu učení
      4. FEA01d - Učení modelu
      5. FEA01e -Sestavení komponenty pro klasifikaci
   2. FEA02 - Document embedding
      1. FEA02a - Výběr metody a algoritmu
      2. FEA02b - Výběr technologie
      3. FEA02c - Sestavení komponenty pro extrakci embeddingu
   3. FEA03 - Metrika podobnosti dokumentů
      1. FEA03a - Výběr algoritmu
      2. FEA03b - Výběr technologie
      3. FEA03c - Sestavení komponenty pro porovnávání dokumentů
   4. FEA04 - Extrakce předmětu
      1. FEA04a - Výběr metody a algoritmu
      2. FEA04b - Výběr technologie
      3. FEA04c – Návrh procesu extrakce předmětu
      4. FEA04d - Sestavení komponenty pro extrakci předmětu
4. REC00 - Doporučovací systém
   1. REC01 - Výběr doporučovací metody
   2. REC02 - Výběr technologie
   3. REC03 - Návrh systému
   4. REC04 - Sestavení
5. APP00 - Aplikace
   1. APP01 - Výběr technologie
   2. APP02 - Sestavení prototypu
   3. APP03 - Integrace s ostatními komponentami
   4. APP04 - Nasazení?
6. DIP00 - Psaní diplomového dokumentu

## Časově sousledný pohled

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Leden | | | | Únor | | | | Březen | | | | Duben | | | | Květen 7. 5. deadline | | | |
| STH01 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DAT01 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DAT02 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DAT03 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DAT04 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FEA01 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FEA02 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FEA03 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FEA04 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| REC01 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| REC02 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| REC03 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| REC04 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| APP01 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| APP02 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| APP03 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| APP04 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DIP00 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

# STH00 – Stakeholders

## STH01 – Průzkum uživatelů a jejich požadavků

Pokusit se zkontaktovat nějaké zastupitele obcí ohledně jejich názor a požadavky na systém?

Marek má asi kontakt na starostu ve Slaném, se kterým se o projektu bavil, tak možná by se dalo s ním něco probrat.

# DAT00 - Data

## DAT01 - Dokumenty veřejných zakázek

Dataset dokumentů veřejných zakázek (dále VZ) vytvářím stahováním dokumentací VZ zadavatelů uveřejněných ve věstníku veřejných zakázek (dále VVZ). Používám k tomu open-source stahovátko *public-contracts* (dále PC)*,* vyvíjeného v rámci projektu *opendatalab*.

PC pracuje s centrální Postgres databází, kam ukládá veškeré záznamy.

PC nalezne všechny zadavatele VZ v systému VVZ scrappováním jejich webového vyhledávače profilů. Ke každému zadavateli ukládá url adresu jeho profilu, kde jsou zveřejňované zadavatelovy VZ.

Existuje několik portálů s profily zadavatelů. Mezi největší patří:

* [www.vhodne-uverejneni.cz](http://www.vhodne-uverejneni.cz)
* [www.e-zakazky.cz](http://www.e-zakazky.cz)
* [www.egordion.cz](http://www.egordion.cz)
* nen.nipez.cz
* [www.tenderarena.cz](http://www.tenderarena.cz)
* [www.profilzadavatele.cz](http://www.profilzadavatele.cz)

Každý z těchto portálů podle zákona podporuje XML API, přes které jsou dostupná základní data o všech zakázkách daného profilu. Mimo jiné jsou v těchto datech také odkazy na jednotlivé dokumenty k zakázkám.

Pro dokumenty již portály speciální API nemají, takže pro získání každého dokumentu je potřeba opět scrappovat jejich webová rozhraní.

Stahované dokumenty jsou ukládané v plain-text formátu do databáze. O extrakci textu se stará knihovna Apache Tika s TesseractOCR. Vzhledem k tomu, že Tika nepodporuje rozpoznávání českého jazyka, v některých případech špatně zpracovává PDF dokumenty. TesseractOCR češtinu podporuje a bez problému čte i dokumenty, se kterýma má Tika problém. Kvůli tomu převádím PDF dokumenty do obrazové reprezentace a zpracovávám samotným OCR.

*Aktuální stav: 9. 1. 2020*

|  |  |
| --- | --- |
| *Počet zadavatelů* | *22888* |
| *Celkový počet zakázek (období leden - červen 2019)* | *136k* |
| *Počet postahovaných zakázek* | *17k* |
| *Celkový počet dokumentů k zakázkám* | *900k* |
| *Počet postahovaných dokumentů* | *135k* |

*Stahování trvá dlouho (cca 1000/den). Systém tenderarena.cz pravidelně odmítá službu kvůli obraně proti zahlcení, čímž tvoří “bottleneck” procesu.*

## DAT02 - Kategorie produktů a služeb

Jako kategorie (třídy) zakázek a jejich dokumentů chci využít CPV kódy. CPV kódy jsou (zřejmě) uváděny u všech zakázek ve VVZ. Na profilech zadavatelů už z pravidla nejsou a v XSD schématu API profilů položku ani nemají.

Potřebuji tedy CPV kódy přiřadit k zakázkám dodatečně. Zároveň je zde ale problém s identifikací zakázek. Ve VVZ existuje evidenční číslo zakázky (dále ECZ), které ale také není uvedeno u všech zakázek na profilech.

Získat CPV kódy by asi mohlo jít vyscrapováním VVZ, nebo z databáze systému, kde je mají. Jedním z takových systémů je Hlídač Státu . Existuje sice API ke hlídači, ale neposkytuje ke smlouvám CPV kódy.

*Aktuální stav: 9. 1. 2020*

|  |  |
| --- | --- |
| *Počet zakázek s ECZ* | *30k* |
| *Počet postahovaných zakázek s ECZ* | *9k* |
| *Počet postahovaných dokumentů k zakázkám s ECZ* | *88k* |

*Snažím se vyjednat se správcem Hlídače státu získání CPV kódů.*

## DAT03 - Katalog produktů a služeb

Pro přesnější extrakci předmětu zakázky (produktu/služby) by mohlo jít vyhledávat položky podle „katalogu“.

*Aktuální stav: 9. 1. 2020*

*Marek říkal něco o katalogu z Alzy. Nevím, jestli to nějak dopadne.*

*Zkoušel jsem dělat scrapper na heureka.cz, která má ale asi 22 milionů produktů a hodně omezené možnosti stránkování. Za předpokladu, že by mě nechali bez postihu scrappovat by to podle mého odhadu trvalo asi 300 hodin.*

## DAT04 – Data o lokacích

Pro zeměpisnou identifikaci zakázek je potřeba získat data o adresách zadavatelů/dodavatelů a jejich místním zaměření.

# FEA00 - Extrakce feature z dokumentů

## FEA01 - Klasifikace dokumentů

### FEA01a - Výběr metody a algoritmu

V tuhle chvíli (od června 2019) drží SOTA v NLP metoda XLNet, která rozšiřuje předešlou úspěšnou metodu BERT.

BERT byl vydaný v říjnu 2018 a od té doby se stal SOTA v mnoha NLP problémech. Stal se tak velice populárním a vzniklo z něj mnoho upravených modelů. Jeho sílou je množství před-trénovaných modelů, které jdou ladit pro specifické NLP úlohy. Nyní existují modely naučené pro desítky světových jazyků, včetně češtiny.

BERT, kvůli jeho veliké podpoře, volím pro své účely i já.

### FEA01b - Výběr technologie

Existují dva frameworky poskytující model s podporou češtiny:

* Transformers (huggingface) - BERT-Base, Multilingual Cased (New, recommended)
* deeppavlov - SlavicBERT, Slavic (bg, cs, pl, ru)

Zatím jsem vyzkoušel klasifikaci s deeppavlov modelem na kaggle\_insults datasetu.

### FEA01c - Návrh procesu učení

Mám k dispozici Profinití stroj: 2 x Intel(R) Xeon(R) Silver 4114 CPU, 2 x Nvidia GTX 2080 Ti, 64 GB DDR4  
Jeho využití je umožněno za použití dockeru.

*Aktuální stav 9. 1. 2020*

*Mám připravený vzorový (kaggle\_insults) docker pro učení deeppavlov modelu na obou CPU i GPU.*

### FEA01d - Učení modelu

Vybranou technologií naučím výsledný klasifikátor.

### FEA01e -Sestavení komponenty pro klasifikaci

Aby bylo možné naučený klasifikátor použít v aplikaci, sestavím komponentu, která k němu bude tvořit rozhraní a bude zaintegrovaná do aplikace.

## FEA02 - Document embedding

### Motivace

Dnešní digitální výpočetní technologie spoléhají bez výjimky na číselnou reprezentaci dat. Některá data je možné přirozeným způsobem zobrazovat do spojitého prostoru, jako například obraz reprezentovaný dvourozměrným popisem barev, či zvuk popsaný průběhem signálu, zatímco jiná data takto přirozeně reprezentovat nelze.

Jedním z obtížně reprezentovatelných typů dat je text přirozeného jazyka, kde se pro reprezentaci slov při jeho zpracování tradičně užívá kód *1 z n*, který z principu vytváří velmi řídce obsazené prostory s vysokým počtem dimenzí. To je z mnoha důvodů nepraktické. Data jsou prostorově náročná a reprezentace nijak nezachycuje vztahy mezi slovy.

Žádoucí je tak převést všechna slova do spojitého prostoru pevně dané dimenze, která nebude závislá na velikosti slovníku.

V praxi se používají vektory v reálném n-dimenzionálním prostoru nabývajících desítky až stovky rozměrů. Vnoření jsou vytvářena automaticky, často s použitím strojového učení, čímž se význam jednotlivých rozměrů stává neinterpretovatelný. Vektory tak mají význam pouze ve vztahu k ostatním, samostatně ne.

### Embedding

Podle článku (Palachy, 2019), vnořování slov (dále angl. embedding), jako metoda reprezentace slov v číselném vektorovém prostoru, se stala nedílnou součástí dnešních řešení úloh strojového zpracování přirozeného jazyka (dále NLP z angl. natural language processing). Umožňuje různým modelům strojového učení závisejících na vektorové reprezentaci vstupu využít takové reprezentace jako bohatšího vyjádření samotného vstupního textu. Ve vektorovém prostoru je možné zachovat více sémantické i syntaktické informace, což napomáhá k dosažení lepších výsledků v téměř kterékoli úloze zpracování přirozeného jazyka, jakou si jen dokážeme představit.

Milníkem (Ruder, 2018) se stala v roce 2013 publikace práce skupiny Tomáše Mikolova. Jejich word2vec model s inovativním přístupem k získávání embeddingů spustil vlnu zájmu o obor.

Palachy dodává, že myšlenka embeddingu a její zásadně pozitivní dopady vedly vědce k úvaze o její aplikaci na větší textové celky, od vět až po knihy. Snaha mnoha lidí vyústila v řadu nových metod získávání vektorové reprezentace s různými inovativními řešeními a další významné průlomy v oboru.

Ve článku (Palachy, 2019) člení přístupy řešení text embeddingu do čtyř kategorií.

1. Sumarizace slovních vektorů

Klasický přístup, který zastupuje například algoritmus „bag-of-words“ v případě „one-hot“ slovních vektorů. Je možné aplikovat různá schémata vážení k sumarizaci vektorů.

1. Modelování témat

Přestože zde nelze mluvit o získávání embeddingů jako o hlavním účelu, modelovací techniky jako LDA („latent Dirichlet allocation“) nebo PLSI („probabilistic latent semantic indexing“) inherentně generují prostor pro embedding dokumentů. Tím modelují a vysvětlují distribuci slov v korpusu, kde jednotlivé dimenze mohou být viděny jako latentní sémantické struktury skryté v datech.

1. Enkodér-dekodér modely

Přístup využívající vnitřní reprezentaci dat v podobě číselných vektorů. Modely vznikají učením bez učitele s výhodou použití stále dostupnějších velkých korpusů s neoznačenými daty.

1. Učení s učitelem

Modely neuronových sítí mají schopnost naučení získávat obohacené reprezentace vstupních dat, které využívají k řešení úloh souvisejících s textem. Takto naučené sítě obsahují skryté vrstvy, kde jsou data reprezentovaná právě číselnými vektory.

### Klasické metody

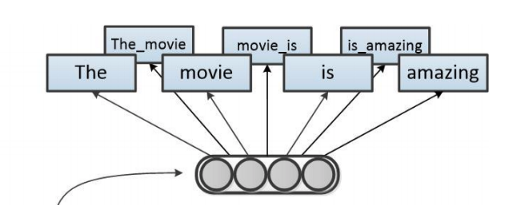
#### Bag-of-words

Metoda reprezentující text jako (multi)množiny, tzv. „bag-of-words“ (dále BOW), vyskytujících se slov. Každý dokument je zastoupen vektorem o délce velikosti slovníku, kde každá pozice zastupuje počet výskytů daného slova z něj. Z této podstaty metoda neuchovává žádnou gramatickou ani souslednou informaci o textu.



#### Bag-of-n-grams

Metoda se snaží oproti klasickému BOW omezit ztrátu informace o pořadí. Místo výskytu slov tak počítá výskyty n-slovných sousloví. Klasický BOW je tak případ této metody pro n = 1. Hlavní problém této metody je nelineární závislost velikosti slovníku na počtu unikátních slov. Proto se tak často užívají techniky ke snížení velikosti slovníku.



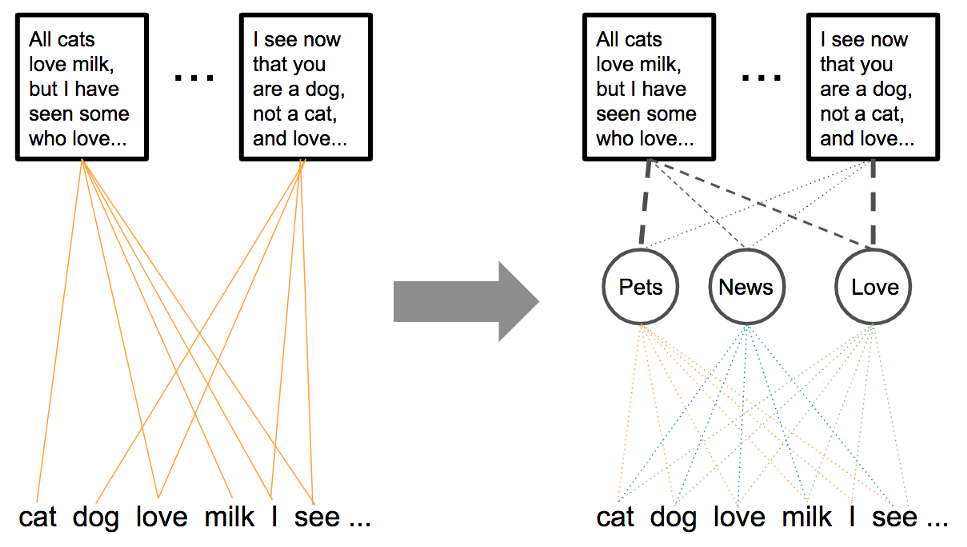
#### Tf-idf

Předešlé metody počítají jednotlivé výskyty, což není zcela objektivní metrika, protože některá slova se vyskytují obecně častěji, čímž se stávají méně důležitými. Alternativou je vážící schéma „term frequency – inverse document frequency“ (dále *tf-idf*). Toto schéma se skládá ze dvou složek: četnost slova v dokumentu (*tf*) a převrácená četnost slova ve všech dokumentech (*idf*). Pro výsledné ohodnocení se složky pronásobí s tím, že *tf* roste s počtem výskytů, zatímco *idf* je vysoké, když je slovo vzácné.

### Modelování témat

#### Latent Dirichlet allocation

LDA je generativní statistický model opírající se o myšlenku, že „každý dokument lze popsat distribucí témat a každé téma může být popsáno distribucí slov“. Při vytváření modelu z korpusu dokumentů vzniká skrytá (latentní) vrstva abstraktních témat. Každý dokument je poté reprezentován jako výběr z Dirichletova rozdělení nad tématy a každé téma jako výběr z Dirichletova rozdělení nad slovy. Hlavním užitím LDA je nekontrolované odhalování témat v dokumentech, tzv. modelování témat (z angl. topic modelling). Existují avšak i užití, kde latentní prostor témat se využívá jako prostor pro embedding dokumentů.



Obrázek 1 Ukázka přechodu z BOW na LDA

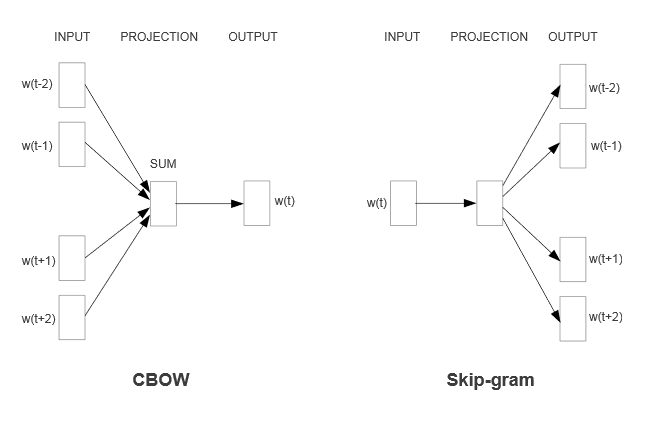
### Metody učení bez učitele

Většina modelů text embeddingu byla navržena podle myšlenky distribuční hypotézy (z angl. *The Distributional Hypothesis*), která říká, že slova používaná a vyskytující se ve stejných kontextech tíhnou k podobnému významu. Tuto myšlenku o slovech modely dále rozšiřuji různými přístupy i pro delší části textu.

#### word2vec

Jak už bylo řečeno v předešlé kapitole, word2vec (Mikolov, 2013) zaznamenal převrat v oboru zpracování přirozeného jazyka. Word2vec je mělká (dvouvrstvá) neuronová síť, která dokáže matematicky podchytit vazby mezi slovy z korpusu. Jako příklad úspěšného objevu se uvádí široce známá formule:

Tvůrci modelu přichází se dvěma architekturami. První je CBOW (z angl. continuous bag-of-words), která predikuje prostřední slovo na základě několika okolních slov, zatímco druhá, tzv. skip-gram architektura, v jistém smyslu dělá opačný proces. Predikuje okolní kontextová slova na základě jednoho vstupního. Dle autorů je mezi architekturami výkonnostní rozdíl. Zatímco CBOW je rychlejší, skip-gram dokáže lépe predikovat pro méně obvyklá slova.



Obrázek 2 Architektury CBOW a skip-gram

#### n-gram embeddingy

V práci (Mikolov, 2013b) rozšiřují skip-gram algoritmus modelu word2vec pro použití na krátkých frázích. Místo jednoslovných tokenů tak používají při učení několikaslovné fráze. Takový přístup je přirozeně nevhodný pro delší fráze kvůli rychlému růstu velikosti slovníku se zvyšující se délkou fráze. Metoda také negeneralizuje svou funkci na neznámé fráze jako jiné metody.

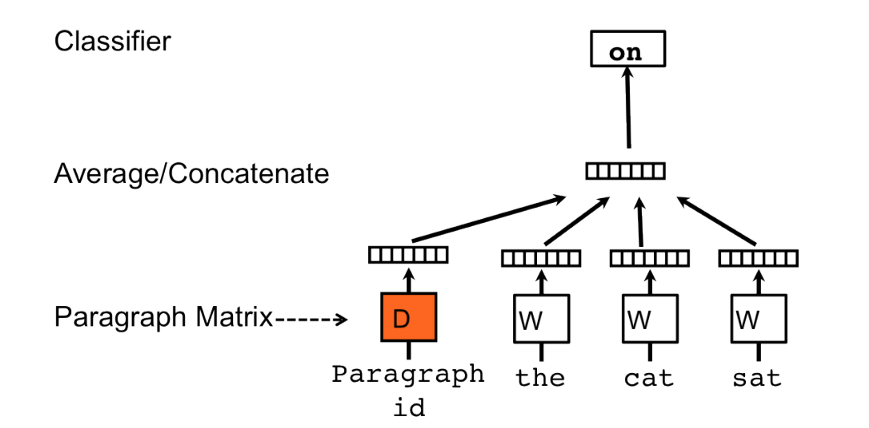
#### Agregace embeddingů slov

Velice intuitivní metoda pro získání embeddingu delšího textu je agregace embeddingů jednotlivých slov. Provedením nějaké vektorové operace na částečných vektorech získáme opět vektor v tom samém embeddingovém prostoru. Nabízí se například sčítání či průměrování vektorů.

Existují i o něco složitější řešení obsahující další kroky či vrstvy. Například (Kenter, 2016) navrhl jednoduchou neuronovou síť nad průměrovanými slovními embeddingy, čímž predikuje okolní věty. (Palachy, 2019) uvádí ve svém článku další příklady.

#### doc2vec

Model, zvaný jako „paragraph vectors“ (Le & Mikolov, 2014), je zřejmě první pokus zobecnění použití modelu word2vec na sekvence slov. Metoda je založená na rozšíření standardního modelu o paměťový vektor, který cílí na zachycení téma/kontextu ze vstupu. Každý odstavec je mapovaný na unikátní vektor (tzv. paragraph vector), podobně jako slova ze slovníku. Tento vektor je sdílený mezi všemi kontexty okna plovoucího přes odstavec. Pro predikované slovo se paměťový vektor přidává ke kontextu fixní velikosti. Vektory pro neznámé odstavce jsou náhodně inicializované.



#### GloVe

Model navržený v práci (Pennington, 2014) přichází s odlišnou myšlenkou získávání informace pro embedding slov, kterou předznamenává jeho název odvozený z anglického „Global Vectors“. Zatímco word2vec model je zaměřen pouze na lokální informaci v textu (slova z okolí), GloVe vedle lokální zachycuje i globální statistiky korpusu. Ve výpočtech k tomu zahrnuje matici společných výskytů slov (z angl. co-occurence matrix). Autorům se tak podařilo vytvořit další model, který předčil ostatní modely v úlohách rozpoznávání pojmenovaných entit, podobnosti či analogie slov.

Ve článku (Ganegedara, 2019) autor popisuje GloVe do podrobna.

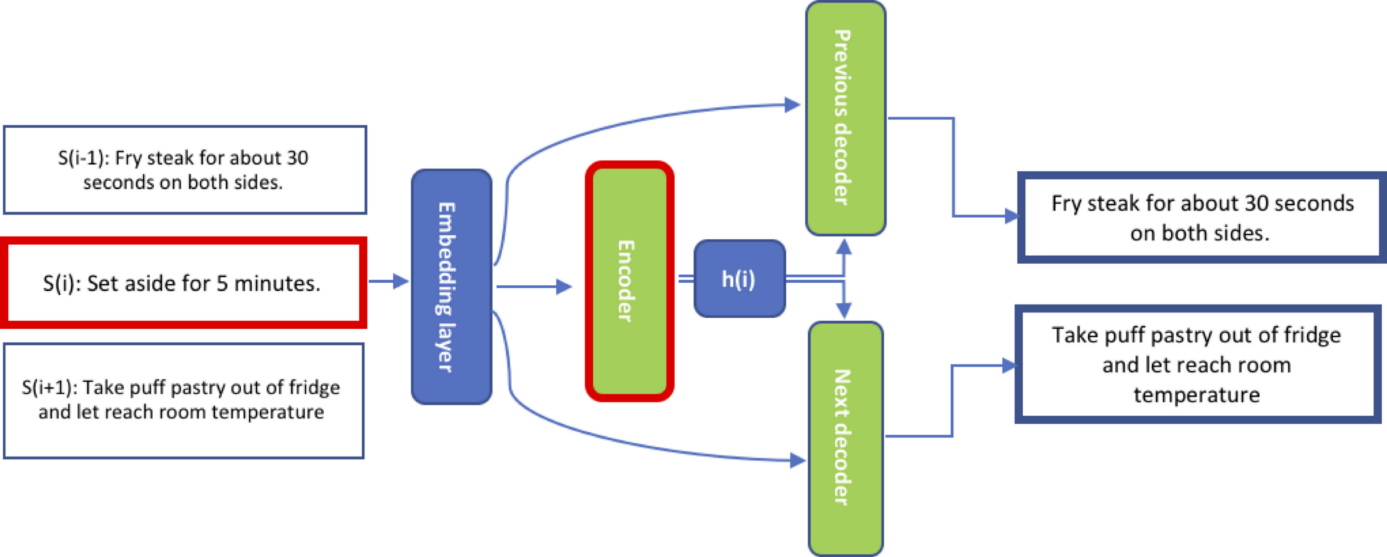
#### fastText

FastText je další rozšíření modelu word2vec v podobě knihovny (Facebook, 2016). Rozšíření spočívá v tom, že místo vkládání samotných slov do neuronové sítě, fastText slova dělí na znakové n-gramy. Například tri-gram dělení pro slovo jablko vychází na čtveřici [jab, abl, blk, lko]. Embedding pro celé slovo se poté získává agregací embeddingů každého n-gramu. Po natrénování modelu tak získáme embeddingy pro jakýkoli n-gram z korpusu, díky čemu tak mohou být embedovaná i vzácná či slova mimo něj. (Huang, 2018) Autoři na blogu (Bojanowski, 2016) dokonce zmiňují, že taková vlastnost napomáhá lépe zpracovat i morfologicky bohaté jazyky, jako je například i čeština.

Mimo to, fastText nabývá lepší výpočetní efektivity při řešení klasifikace oproti ostatním technologiím díky využití hierarchické struktury, za což knihovně přísluší její pojmenování.

#### Skip-thought vektory

Další pokus o zobecnění skip-gram architektury word2vec modelu rozšiřuje původní koncept ve smyslu použití celé věty jako základní jednotky pro predikci. Predikuje předcházející a následující větu. Autoři používají rekurentní neuronové sítě pro enkodér a dekodér, které využívají embedovací vrstvu provádějící embedding jednotlivých slov.



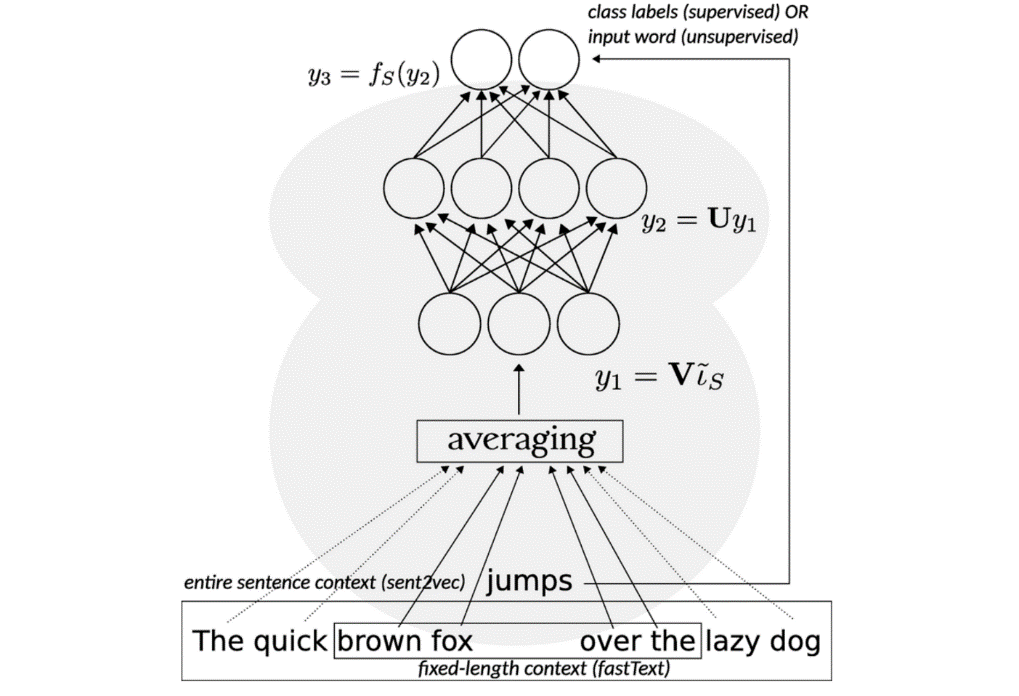
Obrázek 3 Věta s(i) je zakódována enkodérem do skryté reprezentace h(i), na základě dekodéry predikují předešlou s(i-1) a následující s(i+1) větu.

#### FastSent

(Hill, 2016) Zjednodušené řešení skip-thought modelu, kde místo enkodéru a dekodérů se používá klasický BOW. Díky tomu je dosaženo značného snížení výpočetní náročnosti.

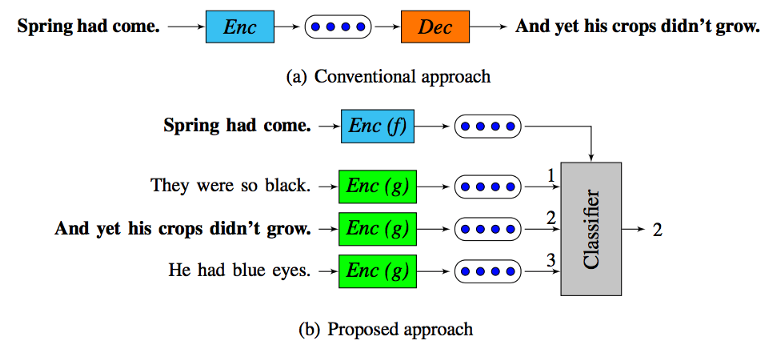
#### sent2vec

V práci (Pagliardini, 2017) představený model sent2vec kombinuje předešlé přístupy. Rozšiřují CBOW algoritmus modelu word2vec o schopnost zpracování celých vět. Pro predikci slova ve větě je místo kontextového okna o fixní velikosti použito jako kontext celý zbytek věty. Vektory slov z kontextu jsou opět průměrované jako vstup do neuronové sítě.



#### Quick-thought vektory

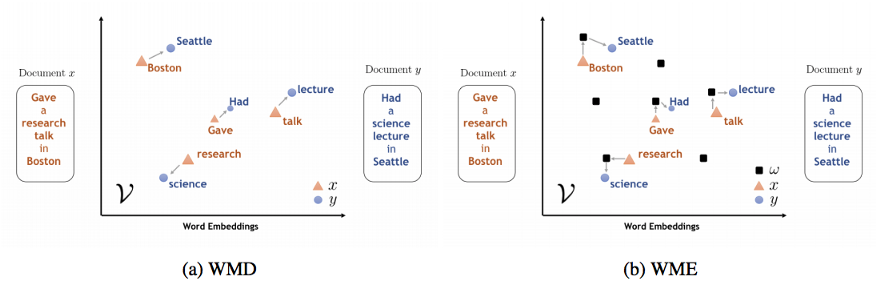
(Logenswaran & Lee, 2018) navrhli přeformulováním úlohy embeddingu dokumentů zcela nový přístup. Kontext, ve kterém se věta vyskytuje, začali predikovat metodou učením s učitelem.



Obrázek 4 Quick-thought (b) řešení, oproti klasickému Skip-thought (a) přístupu

#### Word Mover’s Embedding

Metoda založena na „Word Mover’s Distance“ (Kushner, 2015) - metriky pro podobnost mezi dvěma dokumenty jako minimální vzdálenost, kterou potřebují embedovaná slova jednoho dokumentu urazit ve vektorovém prostoru k dosažení embedovaných slov druhého dokumentu. K tomu bylo navrženo D2KE (z angl. distances to kernel and embeddings) (Wu, 2018a), jako metodologii pro odvození kernelu z dané vzdálenostní funkce.



Obrázek 5 Ukázka vztahu WMD (a) a WME (b) s kernelem odvozeným na základě množiny náhodných dokumentů

Více do detailu popisuje ve článku (Palachy, 2019).

#### ELMo

V práci (Peters, 2018) přichází s myšlenkou hluboce „kontextualizovaných“ slovních embeddingů. Místo přiřazování fixních embeddingů každému slovu, ELMo (z anlg. Embeddings from Language Model) používá oboustrannou LSTM (z angl. bi-directional long short-term memory) síť ke zpracování celé věty, čímž vytváří embedding pro slova na základě kontextu z obou stran. Vnitřní reprezentace slov je založená na jednotlivých znacích, čímž se model stává robustnějším pro slova mimo slovník použitý při trénování. Ve svém článku (Alammar, 2018) autor přehledně prezentuje celý proces ELMo embeddingu.

#### ULMFiT

Zde se nejedná přímo o metodu embeddingu, jako spíš o inovativní přístup k řešení úloh zpracování přirozeného jazyka. ULMFiT (z angl. Universal Language Model Fine-tuning) (Howard, 2018) přichází jako první s aplikací metody „transfer-learningu“ v NLP. Metoda spočívá ve třech krocích:

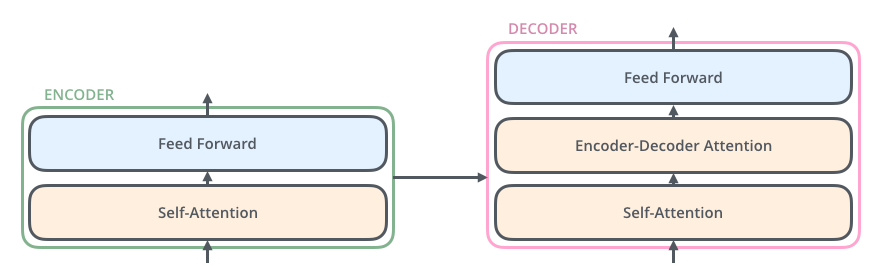
1. Základní před-trénování jazykového modelu (například na textu z Wikipedie)
2. Ladění (z angl. fine-tune) jazykového modelu na cílové doméně
3. Ladění klasifikátoru pro cílovou úlohu

Podle článku (Ghelani, 2019) autory použitá síť tímto způsobem výrazně překonala dosavadní state-of-the-art řešení na několika úlohách klasifikace textu.

#### Transformers

Práce „Attention Is All You Need“ (Vaswani, 2017) odstartovala novou éru v oboru zpracování přirozeného jazyka. Vznikla rodina modelů typu tzv. Transformer, který se vyhýbá konvolučnímu i rekurentnímu přístupu ke zpracování sekvence a místo toho používá inovativní přístup zvaný „attention“. Stále však uchovávají architekturu enkodér (čtení vstupu) – dekodér (provádění predikce).

Tyto modely generují kontextové embeddingy obsahující informaci o sousedních slovech, avšak jejich cílem není tvořit bohatý embedovací prostor pro vstupní text.

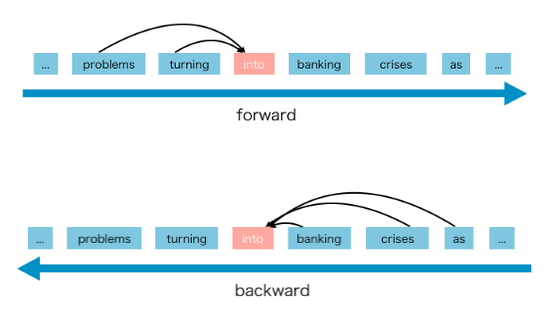


Obrázek 6 Ilustrace enkodér-dekodér architektury Transformer

#### GPT

Jako jednou z prvních adaptací Transformer architektury je GPT (z angl. Generative Pre-Training Transformer) (Radford, 2018). Oproti základní architektuře používá však jinou strukturu. GPT se skládá pouze z na sebe napojených bloků dekodéru. Vyškálováním modelu autoři později dosáhli lepšího modelu, pojmenovaného jako GPT-2 (Radford, 2019).

Ve článku (Liang, 2019) uvádí, že GPT se řadí mezi tzv. autoregresivní jazykové modely (dále LM z angl. language model). Autoregresivní LM při učení používají kontextových slov k predikci následujícího slova. Takový postup je ovšem omezující na dva směry, buď dopředný či zpětný.



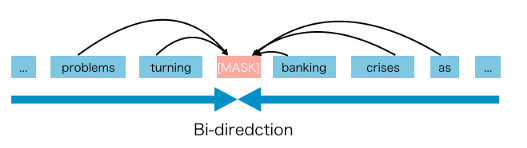
Obrázek 7 Směrově orientované predikce autoregresivního LM

#### BERT

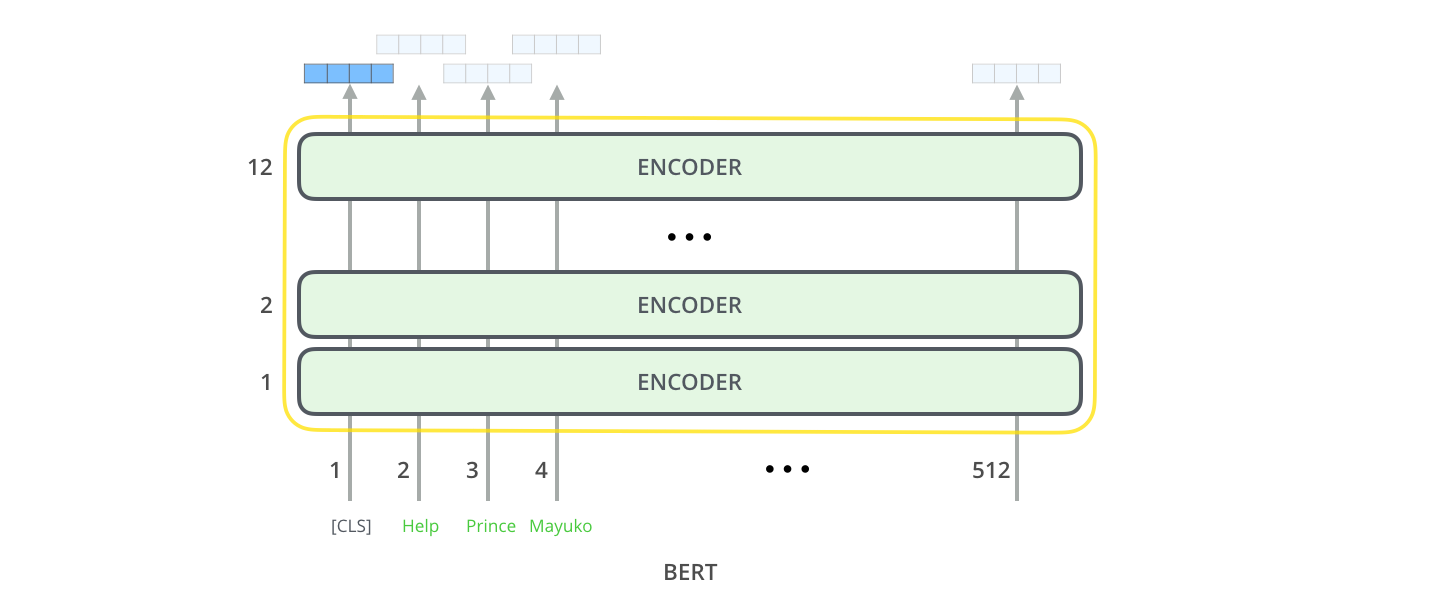
Dalším z úspěšných modelů Transformer rodiny je BERT (z ang. Bidirectional Encoder Representations from Transformers) (Devlin, 2018), který způsobil rozruch v komunitě, kvůli dosažení state-of-the-art výsledků v mnoha různých úlohách zpracování přirozeného jazyka (Horev, 2018).

BERT je navržený pro modelování jazyka a následné jednoduché ladění. Kvůli tomu potřebuje oproti klasickému Transformer modelu pouze mechanizmus enkodéru.

Autoři modelu přišli na nový způsob řešení predikce slova na základě kontextu. BERT nepredikuje následující slovo, ale slovo zamaskované speciální značkou. Takovému přístupu se říká autoenkódovací jazykový model (z angl. autoencoder language model).



Této inovace BERT využívá pro komplexnější využití kontextové informace. Narozdíl od autoregresivních modelů tak enkodér modelu BERT může číst celou sekvenci slov najednou. Konkrétně tedy nejde o dvousměrné čtení, přestože je jeho název modelu tak usouzený.

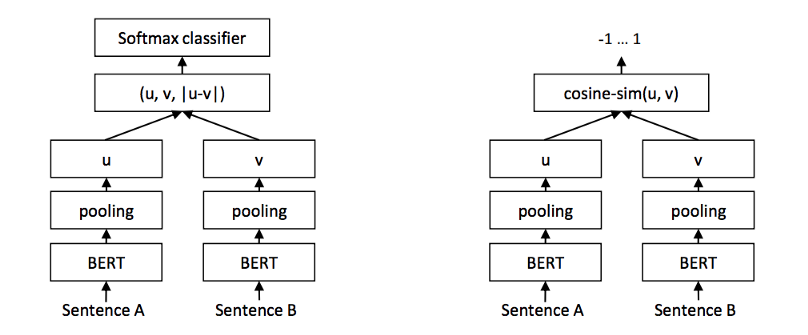


Obrázek 8 Ilustrace architektury modelu BERT

Mnohem víc do detailu o architektuře Transformer a BERT popisuje ve svých ilustrovaných článcích Jay Alammar (Alammar, 2018).

#### Sentence-BERT

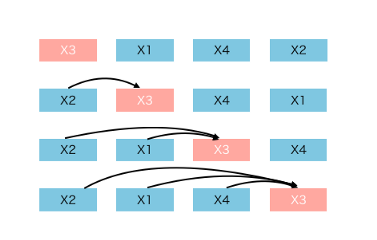
Jak je vidět na obrázku 6, BERT doplňuje sekvenci tokenů o speciální token [CLS], kde jeho výstup lze používat pro klasifikační úlohy. Autoři modelu Sentence-BERT však ukazují (Reimers & Gurevych, 2019), že poskytuje pouze slabý embedding pro další úlohy. Navrhli proto architekturu využívající BERT jako základ pro tzv. „siamese“ a „triplet“ síťovou strukturu, kterou by bylo možné zachycovat sémanticky smysluplné embeddingy vět. Tím se jim podařilo překonat dosavadní state-of-the-art metody pro embedding vět.



Obrázek 9 Architektura SBERT pro trénování klasifikátoru a počítání kosinové podobnosti

#### XLNet

XLNet (Yang, 2019) je model odvozený od modelu BERT. Jak autor vysvětluje ve svém článku (Liang, 2019), princip autoenkodéru v modelu BERT přinesl i svá omezení. Proto autoři modelu XLNet navrhli tzv. permutační modelování jazyku (z angl. permutation LM). To umožňuje modelu využít kontextu z obou stran, přestože zůstává být autoregresivní. Tyto inovace se opět projevily pozitivně na vlastnosti modelu, díky čemu zaujal state-of-the-art výsledky pro mnoho NLP úloh.



Obrázek 10 Permutační modelování jazyka

### Metody učení s učitelem

Palachy (2019) ve svém článku dále uvádí několik přístupů řešení embeddingu pomocí metod učení s učitelem. Pro účely mé práce tyto metody však nejsou důležité, proto je konkrétně nezmiňuji.

### FEA02a - Výběr metody a algoritmu

* FastText (jeví se nejlépe)
* Doc2vec,
* BERT (huggingface),
* Sentence-BERT,
  + Zřejmě SOTA, ale nepodporuje češtinu
* Word Mover’s Embedding,
* Universal Sentence Embedidng (nepodporuje češtinu),
* GloVe (nepodporuje češtinu)

Pravděpodobně bude nejlepší použít BERT embedding, když ho používám jako jazykový model.

### FEA02b - Výběr technologie

Viz. FEA01b

### FEA02c - Sestavení komponenty pro extrakci embeddingu

Obdobně jako FEA01e

## FEA03 - Metrika podobnosti dokumentů

### FEA03a - Výběr algoritmu

Nejlepší bude pravděpodobně metrika na základě embeddingu, kvůli zachycení sémantických podobností. Reálně si ale nejsem jistý, zdali to bude dostatečně fungovat. Kosinová vzdálenost by mohla dobře měřit podobnost dvou embeddingů.

Další zajímavou alternativou je WMD.

### FEA03b - Výběr technologie

Podle výběru algoritmu.

### FEA03c - Sestavení komponenty pro porovnávání dokumentů

Komponenta by měla sloužit jako metrika pro doporučovací engine.

## FEA04 - Extrakce předmětu

### FEA04a - Výběr metody a algoritmu

Potřeba zprocesovat text:

1. Tokenizace
2. Lemmatizace
3. POS tagování
4. Analýza závislostí slov

Podle Davida Šenkýře je nejspolehlivější explicitní návrh patternů větného rozboru, které se matchují v textu.

Alternativně by mohlo jít vyzkoušet sumarizaci dokumentu, ale pochybuji o výsledcích na extrakci předmětu.

#### Větný rozklad

4 úrovně jazykového rozkladu:

1. Slovní úroveň – prostý text (tokeny)
2. Morfologická úroveň – lemma, POS tagy
3. Analytická úroveň – syntaktická analýza (větná struktura)
4. Tektogramatická úroveň - sémantická anotace (vztahy mezi důležitými slovy)

*Aktuální stav: 15. 1.*

*Také bych se o tom chtěl pobavit na schůzce na ZČU.*

### FEA04b - Výběr technologie

Pro větný rozbor lze použít:

* Treex
  + má český model
  + poskytuje až tektogramatickou úroveň rozkladu, čímž se jeví být nejlepší
  + implementace v Perlu
    - slabší podpora
    - komplikovaná instalace
* UDPipe/Udapi
  + Používá Universal Dependencies anotační modely
  + Neposkytuje tektogramatickou úroveň
  + Krom jiných užití existují i jako knihovny pro python
    - Dokonce i api pro spacy, ovšem model na pozadí funguje stále stejně
* MorphoDiTa - má český model, poskytuje pouze morfologickou úroveň rozkladu
* Stanford NLP
* Možná TreeTagger, RFTagger

*Aktuální stav: 16. 1.*

### FEA04c – Návrh procesu extrakce předmětu

Podle vybrané metody a technologie.

### FEA04d - Sestavení komponenty pro extrakci předmětu

Komponenta by měla extrahovat z dokumentace zakázek předmět (produkt, službu) pro vyhledávání/feature extraction.

# REC00 - Doporučovací systém

Systém by měl doporučovat na základě vyhledávání (zadaná klíčová slova, dokument) podobné zakázky (kategorie, předmět, doba, lokalita).

Ještě je potřeba domyslet do jaké míry by mělo doporučování fungovat.

## REC01 - Výběr doporučovací metody

Pravděpodobně content-base algoritmus, využívající množinu feature zakázek.

Na collaborative-filtering nebude mít systém uživatelské vstupy.

## REC02 - Výběr technologie

Nutno prozkoumat.

## REC03 - Návrh systému

Závisí na REC01 a REC02.

## REC04 – Sestavení

Doporučovací systém bude jako další komponenta, která na vstupu bude očekávat feature extrahované z uživatelského vstupu. Na základě feature systém vypočte doporučené položky.

# APP00 - Aplikace

## APP01 - Výběr technologie

Pravděpodobně některý z Python web frameworků.

* Django,
* Flask

## APP02 - Sestavení prototypu

Jednoduché webové rozhraní s možností zadání vstupu pro vyhledávání a oblastí se zobrazením výstupu. Prototyp bude mít mockované komponenty.

## APP03 - Integrace s ostatními komponentami

Do prototypu aplikace zaintegrovat všechny funkční komponenty, aby byl celkový systém schopný fungovat.

## APP04 - Nasazení?

Bude-li to potřeba, připravit a alokovat infrastrukturu pro nasazení celého systému.

# DIP00 - Psaní diplomového dokumentu

Průzkum dosavadních systémů a řešení doporučování veřejných zakázek.

* Tenderman.cz- pricing až 10k Kč/měsíc
* Vhodne-uverejneni.cz