**Що таке NLP?**

Natural Language Processing, як правило, скорочена як NLP - це галузь штучного інтелекту, яка займається взаємодією між комп'ютерами та людьми за допомогою природної мови.

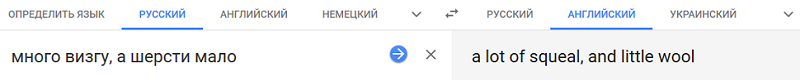
До області NLP можна віднести купу найрізноманітніших речей - від морфологічного аналізу слів до пошуку адрес в текстовому файлі.

Кінцева мета NLP - читати, розшифровувати, розуміти та осмислювати людські мови.

Більшість методик NLP покладаються на машинне навчання, щоб отримувати значення з людських мов.

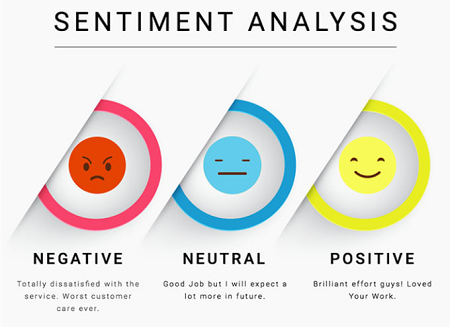
**Завдання NLP**

* Перше і найголовніше історично важливе завдання - це машинний переклад. Ним займаються дуже давно, і є величезний прогрес. Але завдання отримання повністю автоматичного перекладу високої якості (FAHQMT) так і залишається не вирішеним. Це в якомусь сенсі двигун NLP, одна з найбільших завдань, якій можна займатися.



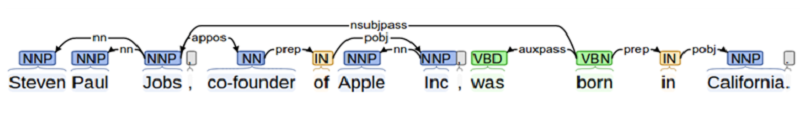
* Друге завдання - класифікація текстів. Дано набір текстів, і завдання - класифікувати ці тексти за категоріями.

1. Перший і один з найважливіших з практичної точки зору способів застосування - класифікація листів на спам і не спам.
2. Інший класичний варіант – класифікація новин за категоріями (рубрикація) - зовнішня політика, спорт, шапіто ... . Або, припустимо, вам приходять листи, і ви хочете відокремити замовлення з інтернет-магазину від авіаквитків і бронею готелів.
3. Третій класичний варіант застосування завдання текстової класифікації - сентіментний аналіз. Наприклад, класифікація відгуків на позитивні, негативні і нейтральні.



* ТРЕТЄ завдання - витяг іменованих сутностей, NER. Ми виділяємо в тексті ділянки, які відповідають заздалегідь обраному набору сутностей, наприклад, треба знайти в тексті всі локації, персони та організації. У тексті «Остап Бендер - директор контори" Рога і Копита "» ви повинні зрозуміти, що Остап Бендер - це персона, а "Рога і Копита" - це організація..
* З третім завданням пов'язане ЧЕТВЕРТЕ - завдання вилучення фактів і відносин (relation extraction). Наприклад, є відношення роботи (Occupation). З тексту «Остап Бендер - директор контори" Рога і Копита "» ясно, що наш герой пов'язаний професійними відносинами з "Рогами і Копитами". Те ж саме можна сказати безліччю інших способів: «Контору" Рога і Копита "очолює Остап Бендер», або «Остап Бендер пройшов шлях від простого сина лейтенанта Шмідта до глави контори" Рога і Копита "». Ці пропозиції відрізняються не тільки предикатом, а й структурою.

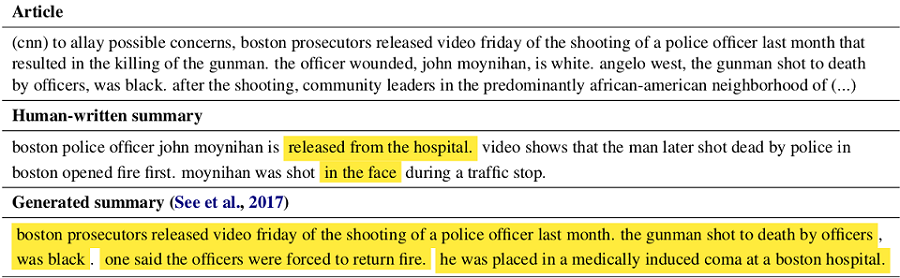
Прикладами інших часто виділяються відносин є відносини купівлі / продажу (Purchase and Sale), володіння (Ownership), факт народження з атрибутами - датою, місцем і т. Д. (Birth) і деякі інші.

Завдання здається не має очевидного практичного застосування, але, тим не менш, воно використовується при структуризації неструктурованою інформації. Крім того, це важливо в питально-відповідних і діалогових системах, в пошукових системах - завжди, коли вам потрібно аналізувати питання і розуміти, до якого типу він відноситься, а також, які обмеження є на відповідь.

* Два наступні завдання, можливо, ХАЙПОВІ. Це питально-відповідні і діалогові системи (чат-боти). Amazon Alexa, Аліса, Siri - це класичні приклади діалогових систем. Щоб вони нормально працювали, має бути вирішено багато завдань NLP. Наприклад, текстова класифікація допомагає визначити, потрапляємо ми в один із сценаріїв goal-oriented чат-бота. Припустимо, «питання про курсах валют». Relation extraction потрібно для визначення наповнювачів шаблону сценарію, а завдання ведення діалогу на загальні теми ("болталки") допоможе нам в ситуації, коли ми не потрапили ні в один із сценаріїв.

Питально-відповідні системи - теж зрозуміла й корисна річ. Ви задаєте машині питання, машина шукає відповідь на нього в базі даних або корпусі текстів.

* Ще один приклад класичної завдання NLP — підсумовування (summarization). Формулювання завдання проста - на вхід система приймає текст великого розміру, а виходом служить текст меншого розміру, якимось чином відображає зміст великого. Наприклад, від машини потрібно згенерувати переказ тексту, його назву або анотацію.



* Ще одна популярна завдання - argumentation mining, пошук обґрунтування в тексті. Вам наводять факт і текст, вам потрібно знайти обгрунтування даного факту в тексті.

**Де використовується NLP?**

NLP застосовується у:

* програмах перекладу мов, таких як Google Translate;
* пекстових процесорах, які використовують NLP для перевірки граматичної точності текстів. Наприклад Microsoft Word та Grammarly;
* програмах інтерактивної голосової відповіді (IVR), що використовуються в телефонних центрах для відповіді на запити певних користувачів.
* персональних програмах-помічниках, таких як OK Google, Siri, Cortana та Alexa.

**Чому вирішувати задачі NPL складно?**

Формулювання завдань не дуже складні, проте самі завдання зовсім не прості, тому що ми працюємо з природною мовою. Явища полісемії (багатозначні слова мають загальний вихідний сенс) і омонімії (різні за змістом слова вимовляються і пишуться однаково) характерні для будь-якого природної мови.

Полісемія: зупинка (процес або будівля), стіл (організація або об'єкт), дятел (птах або людина).

Омонімія: ключ, цибуля, замок, піч.

Іншим класичним прикладом складності мови є займенникова анафора. Наприклад, нехай нам дано текст «Двірник дві години прибирав сніг, він був незадоволений». Займенник «він» може ставитися як до двірника, так і до снігу. По контексту ми легко зрозуміємо, що він - це двірник, а не сніг. Але домогтися, щоб комп'ютер це теж легко розумів, непросто. Завдання займенникової анафори і зараз вирішено не дуже добре, тривають активні спроби поліпшити якість рішень.

Ще одна додаткова складність - це еліпсис. Наприклад, «Петрик з'їв зелене яблуко, а Маша - червоне». Ми розуміємо, що Маша з'їла червоне яблуко. Проте, домогтися, щоб машина теж це зрозуміла, непросто. Зараз завдання відновлення вирішується на крихітних корпусах (кілька сотень пропозицій), і на них якість повного відновлення відверто слабка (близько 0.5). Зрозуміло, що для практичних застосувань така якість нікуди не годиться.

**Як працює обробка природних мов?**

NLP тягне за собою застосування алгоритмів для ідентифікації та вилучення правил природної мови таким чином, що неструктуровані мовні дані перетворювались у форму, зрозумілу комп'ютерам.

Сирий текст не підходить для навчання машин. Потрібно конвертувати текст у вектори – набори цифр. Цей процес називають виділенням ознак.

Мішок слів – техніка виділення ознак, яка описує входження слів у текст.

Для використання моделі необхідно визначити словник відомих слів, які називають токенами, а також обрати ступінь їхньої присутності.

**Приклад**

Розглянемо створення моделі, використовуючи приклад для наочності.

1. **Загрузка даних**

У нас є дані:

I like this movie, it's funny.

I hate this movie.

This was awesome! I like it.

Nice one. I love it.

Загружаємо їх у вигляді масиву:

["I like this movie, it's funny.", 'I hate this movie.', 'This was awesome! I like it.', 'Nice one. I love it.']

1. **Визначаємо словник**

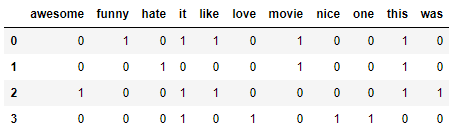
Збираємо усі унікальні слова, ігноруючи реєстр, пунктуацію, токени з одного символу.

1. **Створюємо вектори документа**

Перетворюємо сирий текст у набір цифр, щоб використовувати їх як вхідні дані для моделі машинного навчання.

Відмічаємо наявність слів (1 – є, 0 – немає)

Результат:



Із словником росте вектор документу. У прикладі довжина вектора дорівнює кількості відомих слів. Якщо вектор складається з тисяч або мільйонів елементів, кожен документ може містити лише меншу частину слів їх словника. Виходить вектор з багатьма нулями, який вимагає багато пам’яті та ресурсів.

Можна зменшити кількість слів, щоб зменшити вимоги до обчислювальних ресурсів.

Наприклад, викинути стоп-слова, привести слова до базових форм і виправити неправильно написані слова.

Інший спосіб – використання згрупованих слів, інакше кажучи, N-грам (N – кількість згрупованих слів). У модель потрапляють лише ті, що фігурують у корпусі.

**Приклад:**

Речення:

The office building is open today

Його біграми (сполучення двох слів):

* the office
* office building
* building is
* is open
* open today

Ми оцінили наявність слів, але також можна рахувати їх кількість у документі та частоту відносно загальної кількості.

Коли текст буде надано, комп'ютер буде використовувати алгоритми, щоб витягнути значення, пов'язане з кожним реченням, та зібрати з них суттєві дані.

Іноді комп'ютер може не зрозуміти значення речення, що призводить до незрозумілих результатів.

**Які методи використовуються в NLP?**

Синтаксичний аналіз та семантичний аналіз - основні прийоми, що використовуються для виконання завдань з обробки природних мов.

**1. Синтаксис**

Синтаксис позначає розташування слів у реченні таким чином, що вони мають граматичний зміст.

У NLP синтаксичний аналіз використовується для оцінки того, як природна мова узгоджується з граматичними правилами.

Комп'ютерні алгоритми використовуються для застосування граматичних правил до групи слів та отримання значень від них.

Ось кілька синтаксичних прийомів:

* Стемінг: скорочення слова до [основи](https://uk.wikipedia.org/wiki/Основа_слова) шляхом відкидання допоміжних частин, таких як закінчення чи суфікс.
* Морфологічна сегментація: передбачає поділ слів на окремі одиниці, що називаються морфемами.
* Сегментація слова: передбачає поділ великого фрагмента безперервного тексту на окремі одиниці.
* Позначення частин мови: передбачає визначення частини мови для кожного слова.
* Розбір: Він передбачає проведення граматичного аналізу поданого речення.
* Позначення кінця речення: передбачає розміщення меж речення на великому фрагменті тексту.
* Стовбування: Це передбачає вирізання схильних слів до їх кореневої форми.

**2. Семантика**

Семантика позначає значення, яке передається текстом. Семантичний аналіз - один із складних аспектів обробки природних мов, який ще не був повністю вирішений.

Він передбачає застосування комп’ютерних алгоритмів для розуміння значення та інтерпретації слів та структури речень.

Ось деякі методи семантичного аналізу:

* Розбіжність сенсу у слові: передбачає надання значення слову на основі контексту.
* Генерація природних мов: передбачає використання баз даних для отримання смислових намірів та перетворення їх на людську мову.
* Витяг іменованих сутностей (NER).

**NER**

Витяг іменованих сутностей (Named-entity recognition, NER) - одна з найпопулярніших завдань NLP.

Завдання NER - визначити частини тексту, які можна ідентифікувати та класифікувати за попередньо заданими групами. Приклади таких груп включають імена людей та назви місць.

Припустимо, є новинний текст, і ми хочемо виділити в ньому сутності (деякий заздалегідь зафіксований набір - наприклад, персони, локації, організації, дати і так далі). Завдання NER - зрозуміти, що фрагмент тексту «1 січня 1997 року" є датою, "Кофі Аннан" - персоною, а "ООН" — організацією.

**Зведення задачі NER до задачі класифікації**

Незважаючи на те що сутності часто бувають багатослівними, зазвичай завдання NER зводиться до задачі класифікації на рівні токенов, т. Е. Кожен токен відноситься до одного з декількох можливих класів. Є кілька стандартних способів зробити це, але самий загальний з них називається BIOES-схемою. Схема полягає в тому, щоб до мітки суті (наприклад, PER для персон або ORG для організацій) додати деякий префікс, який позначає позицію токена в спання суті. Більш детально:

B - від слова beginning - перший токен в спання суті, який складається з більше ніж 1 слова.

I - від словами inside - це те, що знаходиться в середині.

E - від слова ending, це останній токен суті, яка складається більше ніж з 1 елемента.

S - single. Ми додаємо цей префікс, якщо сутність складається з одного слова.

Таким чином, до кожного типу сутності додаємо один з 4 можливих префіксів. Якщо токен не відноситься ні до якої сутності, він позначається спеціальною міткою, зазвичай має позначення OUT або O.

Зрозуміло, що за такої розмітки ми однозначно можемо встановити межі всіх анотацій сутностей. Дійсно, про кожен токен ми знаємо, чи вірно, що сутність починається з цього токена або закінчується на ньому, а значить, закінчити чи анотацію суті на даному токені, або розширювати її на наступні маркери.

Переважна більшість дослідників використовує цей спосіб (або його варіації з меншою кількістю міток - BIOE або BIO), але у нього є кілька суттєвих недоліків. Головний з них полягає в тому, що схема не дозволяє працювати з вкладеними або пересічними сутностями. Наприклад, сутність "МГУ імені М.В. Ломоносова "- це одна організація. Але Ломоносов сам по собі - це персона, і це теж було б непогано поставити в розмітці. За допомогою описаного вище способу розмітки ми ніколи не зможемо передати обидва ці факти одночасно (бо у одного токена можемо зробити тільки одну позначку). Відповідно, токен "Ломоносова" може бути або частиною анотації організації, або частиною анотації персони, але ніколи не тим і іншим одночасно.

Інший приклад вкладених сутностей: "Кафедра математичної логіки і теорії алгоритмів механіко-математичного факультету МДУ". Тут в ідеалі хотілося б виділяти 3 вкладених організації, але наведений вище спосіб розмітки дозволяє виділити або 3 непересічні суті, або одну сутність, що має анотацією весь наведений фрагмент.

Крім стандартного способу звести задачу до класифікації на рівні токенов, є і стандартний формат даних, в якому зручно зберігати розмітку для завдання NER (а також для багатьох інших завдань NLP). Цей формат називається CoNLL-U.

**Як вирішити задачу NER за допомогою нейромереж?**

Автори вводять два різновиди архітектури, що відповідають двом різним способам врахувати контекст токена:

* або використовувати «вікно» заданої ширини (window based approach),
* або вважати контекстом все пропозицію (sentence based approach).

Ми отримали на вхід список слів нашої пропозиції: наприклад, "The cat sat on the mat".

Нехай все мається K різних ознак для одного токена (наприклад, такими ознаками можуть виступати словоформа, частина мови, капіталізація, чи є наш токен першим або останнім в реченні і т. П.). Всі ці ознаки ми можемо вважати категоріальним (наприклад, словоформи відповідає логічний вектор довжини розмірності словника, де 1 стоїть тільки на координаті відповідної індексу слова в словнику). Нехай image - логічний вектор, відповідний значенням i-ї ознаки j-го токена в реченні.

Важливо відзначити, що в sentence based approach крім категоріальних ознак, що визначаються за словами, використовується ознака - зрушення щодо токена, мітку якого ми намагаємося визначити. Значення цієї ознаки для токена номер i буде i-core, де core - номер токена, мітку якого ми намагаємося визначити в даний момент (ця ознака теж вважається категоріальним, і вектора для нього обчислюються точно так же, як і для інших).

Наступний етап знаходження ознак токена - множення кожного image на матрицю image, яка називається Lookup Table (таким чином булеві вектора "перетворюються" в безперервні). Нагадаємо, що кожен з image - логічний вектор, в якому на одному місці стоїть 1, а на інших місцях - 0. Таким чином при множенні image на image, відбувається вибір однієї з рядків в нашій матриці. Цей рядок і є ембеддінгом відповідної ознаки токена. Матриці image (де i може набувати значень від 1 до K) - це параметри нашої мережі, які ми навчаємо разом з іншими верствами нейромережі.

Відмінність описаного способу роботи з категоріальним ознаками від появи пізніше word2vec в тому, що тут матриці ініціалізуються випадковим чином, а в word2vec матриці перевчатються на великому корпусі на задачі визначення слова по контексту (або контексту по слову).

Таким чином, для кожного токена отриманий безперервний вектор ознак, що є конкатенацией результатів перемноження всіляких image на image.

Тепер розберемося з тим, як ці ознаки використовуються в sentence based approach (window based ідейно простіше). Важливо, що ми будемо запускати нашу архітектуру окремо для кожного токена (т. Е. Для пропозиції "The cat sat on the mat" ми запустимо нашу мережу 6 разів). Ознаки в кожному запуску збираються однакові, за винятком ознаки, що відповідає за позицію токена, мітку якого ми намагаємося визначити - токена core.

Беремо отримані безперервні вектора кожного токена і пропускаємо їх через одновимірну згортку з фільтрами не дуже великий розмірності: 3-5. Розмірність фільтра відповідає розміру контексту, який мережу одночасно враховує, а кількість каналів відповідає розмірності вихідних безперервних векторів (сумі розмірностей ембеддінгов всіх ознак). Після застосування згортки одержуємо матрицю розмірності m на f, де m - кількість способів, якими фільтр можна прикласти до нашими даними (т. Е. Довжина пропозиції мінус довжина фільтра плюс один), а f - кількість використовуваних фільтрів.

Як і в більшості випадків при роботі з пакунками, після згортки ми використовуємо пулінг - в даному випадку max pooling (т. Е. Для кожного фільтра беремо максимум його значення на всьому реченні), після чого отримуємо вектор розмірності f. Таким чином, вся інформація, що міститься в реченні, яка може нам знадобитися при визначенні мітки токена core, стискається в один вектор (max pooling був обраний тому, що нам важлива не інформація в середньому за пропозицією, а значення ознак на його найважливіших ділянках) . Такий "сплюснутий контекст" дозволяє нам збирати ознаки нашого токена по всьому пропозицією і використовувати цю інформацію, щоб визначити, яку мітку повинен отримати токен core.

Далі пропускаємо вектор через багатошаровий персептрон з якимись функціями активації (в статті - HardTanh), а в якості останнього шару використовуємо повнозв'язну з softmax розмірності d, де d - кількість можливих міток токена.

Таким чином сверточних шар дозволяє нам зібрати інформацію, що міститься у вікні розмірності фільтра, пулінг - виділити саму характерну інформацію в реченні (стиснувши її в один вектор), а шар з softmax - дозволяє визначити, яку ж мітку має токен номер core.

**Векторне представлення слів (word embeddings)**

почну з особливо цікавої під області глибинного навчання- з векторного

представлення слів. По-моєму, зараз векторне представлення- одна з

самих крутих тем для вивчення deep learning, хочу вперше воно було

запропоноване більше десяти років тому.

Векторне подання слова - параметризованих функцій, що відображає слова з деякої природної мови в вектори великої розмірності (припустимо, від 200 до 500 вимірювань). Наприклад, це може виглядати так:

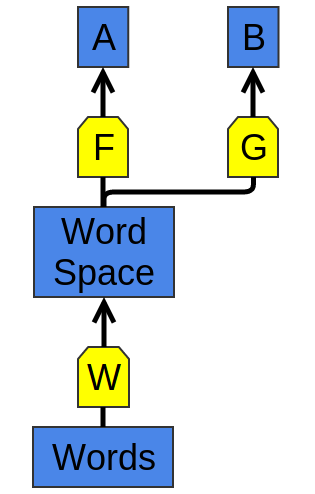




(Як правило, ця функція задається таблицею пошуку, що визначається матрицею, в якій кожному слову відповідає рядок).

W ініціалізується випадковими векторами для кожного слова. Вона навчатиметься, щоб видавати осмислені значення для вирішення деякої задачі.

Властивості векторних уявлень, звичайно, цікаві, але чи можемо ми з їх допомогою зробити щось корисне? Крім дурних дрібниць на кшталт перевірки, «коректна» та чи інша 5-грами.



**W** і F навчають, підганяючи під задачу **A**. Потім **G** зможе вчитися вирішувати задачу **B**, використовуючи **W**.

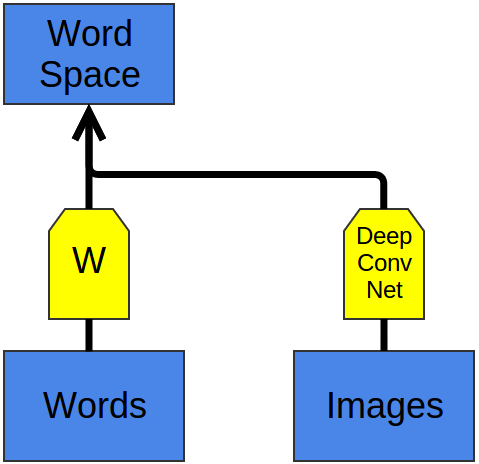
Ми навчили векторні уявлення слів, щоб добре справлятися з простими завданнями, але, знаючи їх чудові властивості, які ми вже спостерігали, можна вважати, що вони стануть в нагоді і для більш загальних проблем. Справді, векторні уявлення на кшталт цих жахливо важливі:

Загальна стратегія - навчити гарне уявлення для завдання **A** і використовувати його для вирішення завдання **B** - один з головних фокусів в чарівному капелюсі глибокого навчання. У різних випадках його називають по-різному: переднавчання (pretraining), transfer learning, і багатоцільове навчання (multi-task learning). Одна із сильних сторін такого підходу - він дозволяє навчати уявлення на декількох видах даних.

Можна провернути цю хитрість по-іншому. Замість настройки уявлень для одного типу даних і використання їх для вирішення завдань різного виду, можна відображати різні типи даних в єдине уявлення!

Нещодавно за допомогою глибокого навчання стали будувати моделі, які «вписують» зображення і слова в єдиний простір уявлень

У попередній роботі моделювали спільний розподіл міток і зображень, але тут все трохи інакше.



Основна ідея в тому, що ми класифікуємо зображення, видаючи вектор з простору уявлень слів. Картинки з собаками відображаються в вектори близько уявлення слова «собака», з кіньми - біля «кінь», з авто - близько «автомобіль». І так далі.

В мережі існує певний сайт (textrazor.com), який надає можливість безкоштовно протестувати можливості NLP (demo), та має кілька функціональних переваг.

* TextRazor використовує сучасні методи обробки природних мов та штучного інтелекту для аналізу, аналізу та вилучення семантичних метаданих із заданого вмісту.
* API TextRazor можна легко інтегрувати з будь-якою мовою, яка може надіслати HTTP-запит і проаналізувати відповідь JSON, завдяки чому можлива потужна аналітика тексту лише за допомогою декількох рядків коду. TextRazor дозволяє витягти будь-яку та всю необхідну інформацію в одному запиті, пов'язуючи витягнуті семантичні метадані, щоб спростити ідентифікацію складних шаблонів.
* Великі дані корисні лише в тому випадку, якщо користувацьке програмне забезпечення може йти в ногу з цим. TextRazor був розроблений з нуля для продуктивності. Написаний на сильно оптимізованому C ++, TextRazor здатний обробляти тисячі слів в секунду. Сервіс створений для автоматичного масштабування до мільярдів запитів у хмарі.
* Стійка інфраструктура TextRazor побудована на хмарі Amazon Web Services і фізичному обладнанні. TextRazor розроблений для забезпечення високої доступності та послідовності продуктивності для аналізу тисяч, мільйонів або мільярдів щоденних документів.
* TextRazor дозволяє додавати назви продуктів, людей, компаній, спеціальні правила класифікації та вдосконалені мовні зразки. Інтегрований двигун Prolog дозволяє швидко поєднувати результати TextRazor з надійною логікою, призначеною для користувача.

**Джерела**

rg\_software, (2009), Заметки об NLP (часть 1).

<https://habr.com/ru/post/79790/>

alexeyev, (2015), Deep Learning, NLP, and Representations.

https://habr.com/ru/post/253227/

grachikova, (2018), Deep Learning in Natural Language Processing

<https://habr.com/ru/company/mipt/blog/349016/>

# Shnurre, (2019),NLP. Основы. Техники. Саморазвитие. Часть 1

<https://habr.com/ru/company/abbyy/blog/437008/>

Shnurre, (2019), NLP. Основы. Техники. Саморазвитие. Часть 2: NER

https://habr.com/ru/company/abbyy/blog/449514/

nvpushkarskiy2, (2019), Основы Natural Language Processing для текста

https://habr.com/ru/company/Voximplant/blog/446738/

Dr. Michael J. Garbade, (2018), A Simple Introduction to Natural Language Processing

[https://becominghuman.ai/a-simple-introduction-to-natural-language- processing-ea66a1747b32](https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32)

alexeyev, (2015), Deep Learning, NLP, and Representations

https://habr.com/ru/post/253227/