Лекция 2

Векторное представление текста

План лекции

- Базовые методы и термины NLP: токенизация, нормализация, n-граммы, стоп-слова, словари
- Задача векторного представления текста
- Разреженные векторные пространства, one-hot encoding, схема взвешивания TF-IDF
- Модели языка, приложения

Установка зависимостей

1. открываем терминал
2. cd mcs-nlp
3. git pull
4. cd lecture2
5. conda install --yes --file requirements.txt *или*
   pip install -r requirements.txt

Токенизация

- **токен** - смысловая единица языка, последовательность символов с определенным значением.
  Обычно (но не всегда!) токенами в NLP считаются слова

- **токенизация** - процесс преобразования текста (последовательности символов) в упорядоченную последовательность токенов

Для токенизации используют

- сегментацию по разделителям (например по пробелам)

- регулярные выражения

- сложные модели, учитывающие специфику языков (см. nltk.tokenize)

Упражнение 1

(15 минут)

```
def tokenize_by_split(text):
    """Tokenizes a given string of text by splitting words by whitespace"""
    tokens = text.split()
    return tokens

def remove_punkt_and_tokenize_by_split(text):
    """Replaces punktuation from given string of text with whitespace, then
    tokenizes it by splitting words by whitespace"""
    punkt_symbols = string.punctuation
    punkt_removed = ''.join([t for t in text if t not in punkt_symbols])
    tokens = punkt_removed.split()
    return tokens
```

```
def tokenize_by_regex(text):
    """Tokenizes a given string of text by applying the 'tokenize' method
    of the provided 'tokenizer' object"""
    tokenizer = nltk.RegexpTokenizer('\w+')
    tokens = tokenizer.tokenize(text)
    return tokens

def tokenize_by_punkt_model(text):
    """Tokenizes a given string of text by applying the NLTK Punkt tokenizer model.
    Uses nltk.word_tokenize method"""
    # your code goes here
    return tokens
```

Нормализация

Нормализация - процесс приведения текста к канонической форме

- приведение к нижнему регистру
  [Hello -> hello]

- стеммирование (нахождение основы слова)
  [playing -> play]

- удаление диакритических знаков
  [résumé -> resume]

- нет универсального метода подходящего для всех приложений

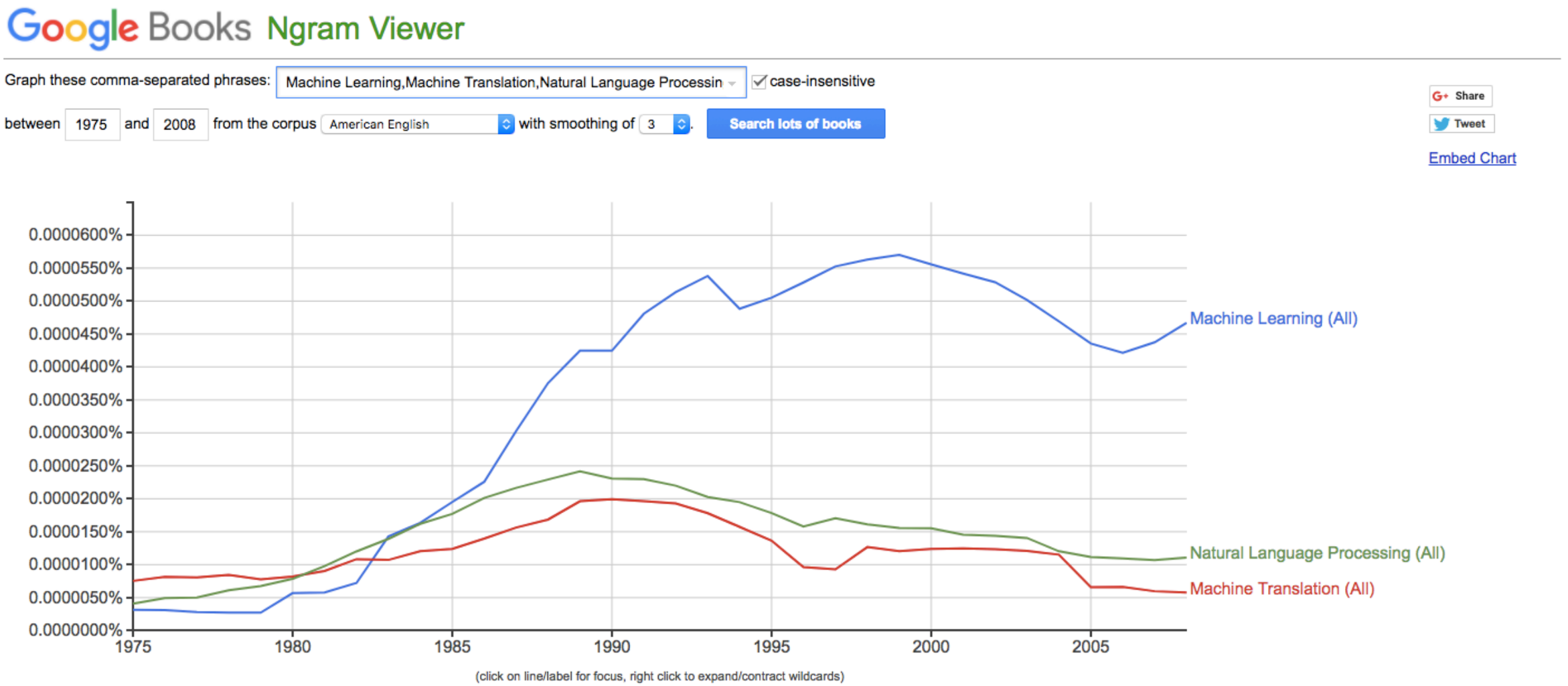
N-граммы

- **n-грамма** - последовательность из n элементов некоего множества (например множества токенов)
  - **“обработка”** - униграмма
  - **“обработка языка”** - биграмма
  - **“обработка естественного языка”** - триграмма

Широко применяются в

- языковом моделировании
- информационном поиске

Google Ngram Viewer



https://books.google.com/ngrams

Стоп-слова

- слова, удаляемые из текста перед автоматической обработкой
- обычно - самые частотные слова языка
- артикли, предлоги, частицы, служебные части речи
- универсального списка не существует
- в зависимости от конкретного приложения/модели, удаление их может улучшать или ухудшать качество

Упражнение 2

(15 минут)

```
# remove stopwords from the list of 'lowered_tokens'
stopword_filtered_tokens = [tok for tok in lowered_tokens if tok not in stopwords]

# turn your filtered list of unigrams into a list of bigrams, joint by whitespace
filtered_bigrams = [' '.join(bigram) for bigram in list(nltk.ngrams(stopword_filtered_tokens, 2))]

# now count the occurances of bigrams using a new Counter instance
bigram_counter = collections.Counter(filtered_bigrams)
```

Словари

- Современный английский насчитывает 13M слов
- Обычно нет ни возможности, ни необходимости учитывать их все при построении моделей NLP
- Рассмотрение ограничивают самыми частотными словами в корпусе, из которых составляют словарь
- В литературе размер словаря обозначают за |V|
  V - vocabulary. обычно |V| ~ 10^4 - 10^6
- **словарь** ставит в соответствие каждому токену из словаря его уникальный номер

Упражнение 3

(30 минут)

```
def build_vocabulary(tokens, max_size=20000):
    """
    Builds a vocabulary of at most max_size words from the supplied list of lists of tokens.
    If a word embedding model is provided, adds only the words present in the model vocabulary.
    """
    vocabulary = {}
    reserved_symbols = ["NULL", "UNKN"]

    counter = collections.Counter(tokens)

    freq_toks = counter.most_common(max_size-len(reserved_symbols))

    voc_words = [k[0] for k in freq_toks]

    for i, reserved in enumerate(reserved_symbols):
        vocabulary[reserved] = i

    for i, k in enumerate(voc_words):
        vocabulary[k] = i+len(reserved_symbols)

    return vocabulary
```

```
def vectorize_tokens(sentence, tokenizer, token_to_id, max_len):
    """
    Converts a list of tokens to a list of token ids using the supplied dictionary.
    Pads resulting list with NULL identifiers up to max_len length.
    """
    # STEP 1: convert sentence to a list of tokens
    tokens = tokenizer(sentence)
    ids = []

    # STEP 2: replace tokens with their identifiers from the vocabulary
    # If the token is not present in the vocabulary, replace it with UNKN identifier
    for token in tokens:
        ids.append(token_to_id.get(token, token_to_id["UNKN"]))

    # STEP 3: pad the sequence id's with NULL identifiers until so that it's length is equal to max_len
    if len(ids) < max_len:
        ids += (max_len-len(ids))*[token_to_id["NULL"]]
    else:
        ids = ids[:max_len]

    return ids
```

О смысле слов

Определение: **смысл**

- Идеальное содержание, идея, сущность, предназначение, конечная цель (ценность) чего-либо

- Содержание знакового выражения; мысль, содержащаяся в словах (знаках, выражениях)

Как записать смысл в форме, понятной машине?

Задача векторного представления текста

Модели машинного обучения работают с данными в векторной форме.

Дано: множество текстовых документов
$D = \{d_i, i = 1,2,...,n\}$

Задача: каждому d_i из D сопоставить точку s_i в Гильбертовом пространстве S

Гильбертово пространство

Свойства:

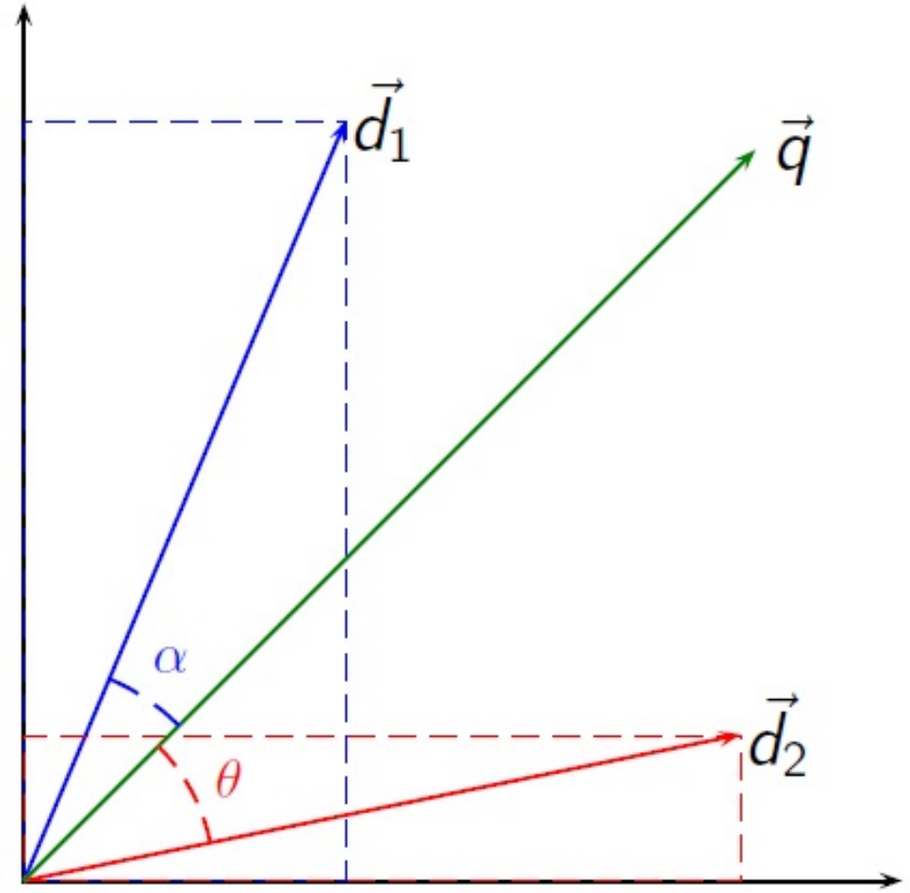
- для двух любых элементов пространства **x, y** определено правило вычисления скалярного произведения **(x, y)**
- это правило удовлетворяет требованиям:
  - **(x, y) = (y, x)** - переместительный закон
  - **(x, y + z) = (x, y) + (x, z)** - распределительный закон
  - **(λx, y) = λ(x, y)** - для любого вещественного λ

Косинусное расстояние

- Наиболее часто применяемая метрика сходства документов в векторном пространстве

- эквивалентно скалярному произведению, если векторы документов нормализованы (норма == 1)

$$\cos\theta = \frac{\mathbf{d} \cdot \mathbf{q}}{\|\mathbf{d}\| \, \|\mathbf{q}\|}$$



one-hot представление

- Рассматривает слова как атомарные, независимые символы
- Представляет слова в виде разреженных векторов размера $\mathbf{R}^{|V|x1}$
- Вектор слова состоит из нулей на всех позициях, кроме номера слова в словаре

  V = {…

  ‘роботы’: 9

  …}

  ‘роботы’ = [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

- Такую модель представления называют one-hot representation

модель: Bag of Words

- С помощью one-hot представления можно кодировать текстовые документы в векторы
- Пример:
  (1) John likes to watch movies. Mary likes movies too.
  (2) John also likes to watch football games.

  (1) [1, 2, 1, 1, 2, 1, 1, 0, 0, 0]
  (2) [1, 1, 1, 1, 0, 0, 0, 1, 1, 1]

модель: TF-IDF

- Term Frequency - Inverse Document Frequency
- схема взвешивания, используемая для оценки важности слова в контексте документа.
- вес слов пропорционален количеству употреблений его в документе и обратно пропорционален частоте употребления в других документах набора.

$$TF - IDF = f_{t,d} * \log(1 + \frac{N}{n_t + 1})$$

- уменьшает вес широкоупотребительных слов и увеличивает вес более редких, что положительно сказывается на качестве векторного представления

Упражнение 4

(30 минут)

```
def onehot_sentence_similarity(sent1, sent2, vectorizer):
    """Encodes provided sentences using the 'vectorizer' object,
    then computes the cosine similarity between sentence vectors
    Outputs a real number between [0,1] """

    # CountVectorizer requires a list of sentences as input
    sent1 = [sent1]
    sent2 = [sent2]

    vec1 = vectorizer.transform(sent1)
    vec2 = vectorizer.transform(sent2)
    similarity = cosine_similarity(vec1, vec2)[0][0]

    return similarity
```

```
class SearchEngine(object):
    def __init__(self, knowledge_base, voc_size=5000):
        """
        Implements a simple information retrieval system based on Tf-Idf text representation.
        """


        self.kbase = np.array(knowledge_base)
        self.vectorizer = TfidfVectorizer(max_features=voc_size)
        self.vectorized_kbase = self.vectorizer.fit_transform(knowledge_base)

    def search(self, query, top_k=3):
        """
        Retrieves the top-k documents from the knowledge_base most similar to given query
        """
        vectorized_query = self.vectorizer.transform([query])

        # your code goes here
        # STEP 1: compute similarities between query and all documents in knowledge base
        sims = cosine_similarity(vectorized_query, self.vectorized_kbase)[0]

        # STEP 2: sort the similarities to find most similar document indices
        # HINT: use np.argsort to do that
        # your code goes here
        sorted_sims = np.argsort(-sims)
        topk_ids = sorted_sims[:top_k]

        # STEP 3: gets top-k most similar documents from self.kbase, returns them
        return self.kbase[topk_ids]
```

BoW: плюсы и минусы

1. Вычислительно простая модель основанная на линейной алгебре
2. Легко интерпретировать
3. Позволяет учитывать вес (значимость) слов
4. Позволяет легко ранжировать документы по релевантности

1. Высокая размерность пространства при большом размере словаря
2. Не учитывает семантику слов, полагает слова статистически независимыми
3. Игнорирует синонимы, полисемию
4. Не может обрабатывать слова с опечатками
5. Теряет порядок слов

Языковые модели

- Задача: присвоить вероятность последовательности токенов

- Пригождается для: $P(W) = P(w_1,w_2,w_3,w_4,w_5...w_n)$

  - Машинного перевода
    P(мама мыла раму) > P(мама мыла драму)

  - Исправления орфографии
    P(мама мыла раму) > P(мама дрыла раму)

  - Генерации текста, вопросно-ответных систем, чат-ботов и т. д.

Как посчитать?

$$P(w_1 w_2 \ldots w_n) = \prod_i P(w_i \mid w_1 w_2 \ldots w_{i-1})$$

P(“мама очень долго мыла раму”)=

P(мама) × P(очень|мама) × P(долго|мама очень) ×

× P(мыла|мама очень долго) × P(раму|мама очень долго мыла)

А это как посчитать?

Посчитать и поделить?

$$P(\text{раму}|\text{мама очень долго мыла}) = \frac{\text{count}(\text{мама очень долго мыла раму})}{\text{count}(\text{мама очень долго мыла})}$$

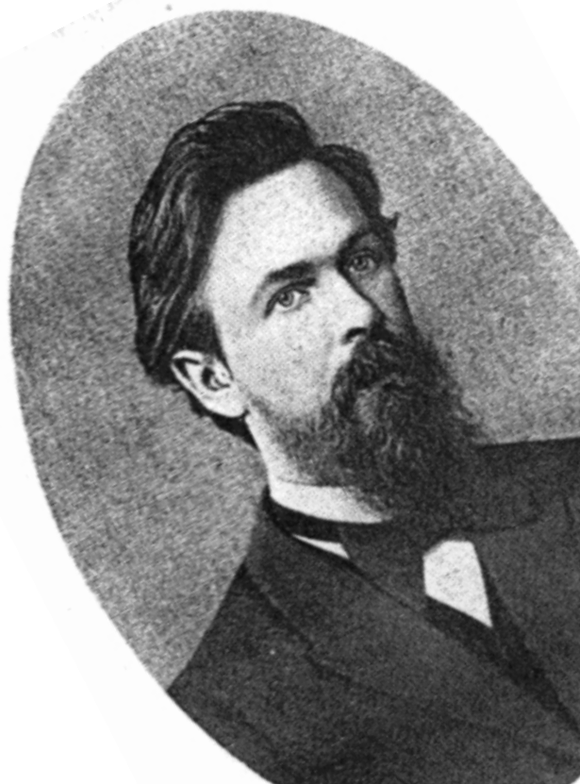
- Слишком много возможных вариантов предложений
- Не хватит никаких данных, чтобы верно оценить вероятность

Допущение Маркова

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$$

P(раму|мама очень долго мыла) ~ P(раму|мыла)

P(раму|мама очень долго мыла) ~ P(раму|долго мыла)



Оцениваем вероятность биграмм

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67$ $P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33$ $P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$

$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5$ $P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5$ $P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$

Упражнение 5