PythonによるWikipediaを活用した自然言語処理

山田 育矢 (Ikuya Yamada)

株式会社Studio Ousia

山田 育矢 (Ikuya Yamada)

Studio Ousia CTO

理化学研究所AIP 客員研究員

博士（学術）

- 引き籠もり系ソフトウェアエンジニア
- 中学の頃からソフトウェアエンジニアとして活動
- 学生時代にベンチャー企業を起業し、売却
- Studio Ousiaを共同創業し、自然言語処理をはじめる
- 好きなもの：
  - Python
  - 映画 (スターウォーズ、ハリーポッター) やドラマなど
  - 寝ること

自然言語処理を用いた製品の開発に
Wikipediaを積極的に活用しています

QA ENGINE

SemanticKernel

Wikipediaを自然言語処理に使う

- Wikipediaは、大規模テキストコーパスとして幅広く使われている
- Wikipediaにはテキスト以外にも様々な有用な情報が含まれている
  - 大量の見出し語（エンティティ）
  - エンティティのリダイレクト構造
  - 記事内のリンク
- Wikipediaは多言語で展開されているため、複数言語への対応が同時に可能

Outline

- Wikipediaを活用した自然言語処理の４つの便利なレシピ
  1. Wikipediaをテキストコーパス として使う
  2. Wikipediaから表記揺れしやすい文字ペアを抽出する
  3. Wikipediaエンティティ辞書を活用する
  4. Wikipediaから単語とエンティティのベクトル表現を学習する
- 事例: クイズを解くAIを作る

スライド中のコードはこちらにあります:
https://github.com/ikuyamada/wikipedia-nlp

Wikipediaをテキストコーパスとして使う

DBpedia NIFデータセット

- DBpediaプロジェクトがWikipediaのテキストやリンクなどをRDF形式で配布
- Wikiマークアップの除去を行わなくてもテキストをそのまま取り出すことが出来る
- DBpediaプロジェクトによるWikipediaクローンをクローリングして抽出しており、エラーが少ない

https://wiki.dbpedia.org/dbpedia-nif-dataset

DBpedia NIFデータセットからコーパスを作成

```python
import bz2
import sys
from rdflib import Graph

def read_ttl(f):
  lines = []
  for line in f:
    lines.append(line.decode('utf-8').rstrip())
    if len(lines) == 1000: #1000行をまとめて処理
      for triple in parse_lines(lines):
        yield triple
      lines = []
  if lines:
    for triple in parse_lines(lines):
      yield triple

def parse_lines(lines):
  g = Graph()
  g.parse(data=u'\n'.join(lines), format='n3')
  return g

with bz2.BZ2File(sys.argv[1]) as in_file:
  for (_, p, o) in read_ttl(in_file):
    if p.toPython() == 'http://persistence.uni-
leipzig.org/nlp2rdf/ontologies/nif-
core#isString':
      print(o.toPython())
```

wiki_corpus.py

- Python RDFlibライブラリを使って、Turtle形式からテキストを抜き出す
- Core i7のマシーンで一時間程度で完了
- 4GBのテキストデータが生成される

```
% wget http://downloads.dbpedia.org/2016-10/core-i18n/ja/nif_context_ja.ttl.bz2
% python wiki_corpus.py nif_context_ja.ttl.bz2 > corpus.txt
```

事例: Wikipediaから単語のベクトル表現の学習

```
import logging
import sys
from gensim.models.word2vec import Word2Vec,
LineSentence

logging.basicConfig(level=logging.INFO)

model = Word2Vec(LineSentence(sys.argv[1]), sg=1)
model.save(sys.argv[2])
```

word2vec.py

```
% mecab -Owakati corpus.txt -o
corpus_wakati.txt
% python word2vec.py corpus_wakati.txt
wiki_w2v
```

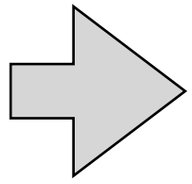
```
>>> model = Word2Vec.load('wiki_w2v')
>>> model.most_similar('日本')[:3]
[('韓国', 0.6719746589660645),
 ('台湾', 0.6447558403015137),
 ('英国', 0.6377681493759155)]
```

- Gensimライブラリに実装されたskip-gramモデル (Word2vec) を用いてWikipediaから単語のベクトル表現を学習
- 一般的なPCで、数時間で高品質な単語の表現を学習できる

Wikipediaから表記揺れしやすい文字ペアを抽出する

表記揺れしやすい文字ペアを抽出

https://ja.wikipedia.org/wiki/慶応大学
https://ja.wikipedia.org/wiki/斉藤工
https://ja.wikipedia.org/wiki/金閣寺
https://ja.wikipedia.org/wiki/New_York



https://ja.wikipedia.org/wiki/慶應義塾大学
https://ja.wikipedia.org/wiki/斎藤工
https://ja.wikipedia.org/wiki/鹿苑寺
https://ja.wikipedia.org/wiki/ニューヨーク

- **Wikipediaリダイレクト:** Wikipediaエンティティにアクセスする際のある程度の表記揺れを解消するための機能
  - 日本語のWikipediaで67万件程度登録されている
  - 大規模な”同義語”の辞書として使用することも出来る
- Wikipediaリダイレクトを元データとして、表記揺れとして出現しやすい文字のペアを抽出してみる

Wikipedia2Vecのインストール

```
% pip install wikipedia2vec
```

- これ以降のスライドでは、オープンソースのPythonライブラリであるWikipedia2Vecを頻繁に使います
- pipでインストールすることが出来ます

準備

- DBファイルをダウンロードする場合:

```
% wget http://wikipedia2vec.s3.amazonaws.com/models/ja/2018-04-20/
jawiki_20180420.db.bz2 -O jawiki.db.bz2
% bunzip2 jawiki.db.bz2
```

- DBファイルを生成する場合:

```
% wget https://dumps.wikimedia.org/jawiki/latest/jawiki-latest-pages-
articles.xml.bz2
% wikipedia2vec build_dump_db jawiki-latest-pages-articles.xml.bz2
jawiki.db
```

表記揺れしやすい文字ペアを抽出する

```python
import sys
import Levenshtein
from collections import Counter
from wikipedia2vec.dump_db import DumpDB

dump_db = DumpDB(sys.argv[1])
pair_counter = Counter()

for (title1, title2) in dump_db.redirects():
    ops = Levenshtein.editops(title1.lower(),
title2.lower())
    if len(ops) == 1:
        (op, p1, p2) = ops[0]
        if op == 'replace':
            pair_counter[frozenset((title1[p1],
title2[p2]))] += 1

for (pair, count) in pair_counter.most_common():
    print('%s\t%s\t%d' % (*list(pair), count))
```

similar_char.py

```
% python similar_char.py jawiki.db >
out.tsv
% cat out.tsv
イ       ー      1857
澤       沢      1747
・       =       1124
```

- Wikipediaのリダイレクト（エンティティ名のペア）を全て集計する
- python-Levenshteinを用いて、１文字のみが異なるエンティティ名のペアを抽出し、異なる文字のペアをカウントする
- 日本語における表記揺れしやすい文字ペアの集合とその出現数を取得できる

https://github.com/ikuyamada/wikipedia-nlp

抽出結果 (全体):

文字1	文字2	数
イ	ー	1857
澤	沢	1747
・	＝	1124
ズ	ス	1118
ブ	ヴ	968
町	村	966
イ	ィ	754
・		747
ェ	エ	594
龍	竜	553
国	國	550
～	～	505
廣	広	493
ヤ	ア	479
ケ	ヶ	467
ド	ト	449
ウ	ー	447
ク	グ	396
真	眞	386
ウ	ヴ	376
ル	ー	366
浜	濱	350
ッ	ー	306
条	條	304
ザ	サ	297

抽出結果(漢字のみ):

文字1	文字2	数
澤	沢	1747
町	村	966
龍	竜	553
国	國	550
廣	広	493
真	眞	386
浜	濱	350
条	條	304
町	市	283
櫻	桜	248
県	市	233
2	二	228
萬	万	224
崎	﨑	222
嶋	島	221
滝	瀧	206
形	系	203
三	3	190
曾	曽	190
高	髙	185
邊	辺	183
齋	斎	181
峰	峯	93
徳	德	88
熙	熙	85

- カタカナ、漢字、記号などの表記揺れしやすい文字がいい感じに検出できている
- 漢字だと旧字体と新字体のペアが多数検出されている
- 「・」と「＝」、「～」（波ダッシュ）と「～」（全角チルダ）などの記号のペア検出されている
- 一部のペア（例：「町」と「村」）を除いて、タスクによっては、文字ベースの正規化ルールのベースとして、用いることができそう

Wikipediaエンティティ辞書を活用する

エンティティ辞書とは？

エンティティ名: アップル

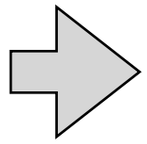
エンティティ:アップル (企業)
https://ja.wikipedia.org/wiki/アップル_(企業)

エンティティ:リンゴ
https://ja.wikipedia.org/wiki/リンゴ

- エンティティ辞書は、エンティティ名（文字列）を可能なエンティティにマップする辞書
  - エンティティ名: エンティティを指し示す可能性のある文字列
  - エンティティ: Wikipediaで一意のURLを持つエントリ
- Wikipediaやウェブ上にあるエンティティへのリンクから作成されることが多い

Wikipediaからエンティティ辞書を作る

パブロ・ピカソは、マラガに生まれ、フランスで制作活動をした画家、素描家、彫刻家。



エンティティ名	エンティティ (見出し語)
マラガ	マラガ
フランス	フランス
画家	画家
素描家	素描
彫刻家	彫刻家

- Wikipedia記事に出現する全ての内部リンクを抽出し、それぞれのリンクテキスト（エンティティ名）とリンク先（エンティティ）を記録して集計する
- 作成した辞書は、様々な用途に使用できる:
  - 専門用語や固有名詞を含む単語・フレーズ辞書
  - エンティティ抽出のための辞書
- 簡単に高品質な辞書を作成することが出来るため、さまざまな手法で用いられている

エンティティ辞書の有用な二つの指標

- **リンク確率:**
  - エンティティ名がリンクとしてWikipedia上にあらわれる確率
  - 各エンティティ名に対して定義される
  - Wikipedia中にエンティティ名”ワシントン”が100個のページに出現し、うち30回でリンクとして出現していたら、0.3
- **コモンネス:**
  - エンティティ名が特定のエンティティを指し示す確率
  - エンティティ名とエンティティの組み合わせに対し定義される
  - Wikipedia中にエンティティ名”アップル”が100個のページに出現し、うち70回が”アップル (企業)”というエンティティを指していたら、”アップル”というエンティティ名と、”アップル (企業)”というエンティティに対して0.7

準備

- 辞書ファイルをダウンロードする場合:

```
% wget http://wikipedia2vec.s3.amazonaws.com/models/ja/2018-04-20/
jawiki_20180420_mention.pkl.bz2 -O jawiki_mention.pkl.bz2
% wget http://wikipedia2vec.s3.amazonaws.com/models/ja/2018-04-20/
jawiki_20180420_dic.pkl.bz2 -O jawiki_dic.pkl.bz2
% bunzip2 jawiki_dic.pkl.bz2 jawiki_mention.pkl.bz2
```

- 辞書ファイルを生成する場合:

```
% wget https://dumps.wikimedia.org/jawiki/latest/jawiki-latest-pages-
articles.xml.bz2
% wikipedia2vec build_dump_db jawiki-latest-pages-articles.xml.bz2
jawiki.db
% wikipedia2vec build_dictionary jawiki.db jawiki_dic.pkl
% wikipedia2vec build_mention_db jawiki.db jawiki_dic.pkl
jawiki_mention.pkl
```

利用例1: 単語・フレーズ辞書として利用

単語・フレーズ	リンク確率
日本	0.29
環境問題	0.31
カルシウム	0.39
アップル	0.4
自民党	0.47
構造主義	0.54
クレジットカード	0.6
自然言語処理	0.73
二酸化炭素	0.80
幕張メッセ	0.82
機械学習	0.82
コミックマーケット	0.91
応仁の乱	1.0
きゃりーぱみゅぱみゅ	1.0

単語・フレーズとそのリンク確率の例

- エンティティ辞書を単語やフレーズが含まれた辞書として活用
- 人手で作成されたリンクから取得しているので、分割や連結の間違いがない辞書が作成できる
- リンク確率を用いて辞書内のエントリをフィルタリングする
  - 閾値はタスクに応じて調整すると良い。経験的には0.2程度で綺麗な辞書が生成できる

利用例1: 単語・フレーズ辞書として利用

```python
import sys
from wikipedia2vec.dictionary import
Dictionary
from wikipedia2vec.mention_db import
MentionDB

dic = Dictionary.load(sys.argv[1])
db = MentionDB.load(sys.argv[2], dic)
words = set()

for mention in db:
  if mention.link_prob >= 0.2:
    if mention.text not in words:
      words.add(mention.text)
      print(mention.text)
```

word_dic.py

```
% python word_dic.py jawiki_dic.pkl
jawiki_mention.pkl > out.txt
% cat out.txt | sort -R | less
% cat out.txt | wc -l
1441724
```

リンク確率0.2以上の閾値を用いて、
約144万件の辞書を獲得

平修
岡田結実
小熊正二
仁田山城
アフリカ尖鼠目
宝坂村
中山村
浅美結花
本庄道信
ミス・マープル3 復讐の女神
有田鉄道
昔造られた鉄道の高架
春だ!生です旅サラダ
university ingres
鶏龍火車碼頭
吉浜ic
ゴッドファーザーpart ii
辛酉政変
日産スカイラインgt-r
as.laranja kyoto
海上挺進戦隊
高津警察署

冨樫晴貞
おいしい神しゃま
松阪競輪場
捜査の端緒
国家安全局長
ヴォルガ地域の征服
破瓜
ヒルトン東京お台場
フリギア音階
天山久晴
金沢の雨
多田満頼
体重別
fasaコーポレーション
根井行親
阿倍比羅夫
道の駅キララ多伎
荒垣秀雄
国泰寺町二丁目
f-4d戦闘機
柏小学校
gp7b

利用例2: エンティティリンキング

This scientist names a constant that is equal to ***Loschmidt's Constant*** times "RT over P" and is equal to the ***Faraday constant*** over the ***elementary charge***.

- 文書に対して、辞書に含まれるエンティティ名を抜き出して、該当するWikipediaのエンティティを抽出する
- 主な用途:
  - Wikipediaエンティティを用いた文書へのタグ付け
  - エンティティを入力としてテキスト分類などのタスクに用いる

利用例2: エンティティリンキング

```python
import sys
from wikipedia2vec.dictionary import Dictionary
from wikipedia2vec.mention_db import MentionDB
from
wikipedia2vec.utils.tokenizer.mecab_tokenizer
import MeCabTokenizer

dic = Dictionary.load(sys.argv[2])
db = MentionDB.load(sys.argv[3], dic)
with open(sys.argv[1]) as f:
  text = f.read()
tokenizer = MeCabTokenizer()
tokens = tokenizer.tokenize(text)

for mention in db.detect_mentions(text, tokens):
  print(mention)
```

entity_linking.py

```
% python entity_linking.py jawiki_dic.pkl
jawiki_mention.pkl input.txt
<Mention NHK連続テレビ小説 -> 連続テレビ小説>
<Mention 半分、青い。 -> 半分、青い。>
<Mention 永野芽郁 -> 永野芽郁>
<Mention コラムニスト -> コラムニスト>
<Mention 木村隆志 -> 木村隆志>
```

物語はいよいよ終盤を迎えているNHK連続テレビ小説『半分、青い。』。クライマックスに向けてのストーリーは、ヒロイン演じる永野芽郁も驚いて台本をお風呂に落としたほどだという。目まぐるしいストーリー展開で注目を集めた同作は、どんな結末を迎えるのか。コラムニストでテレビ解説者の木村隆志さんが解説する。

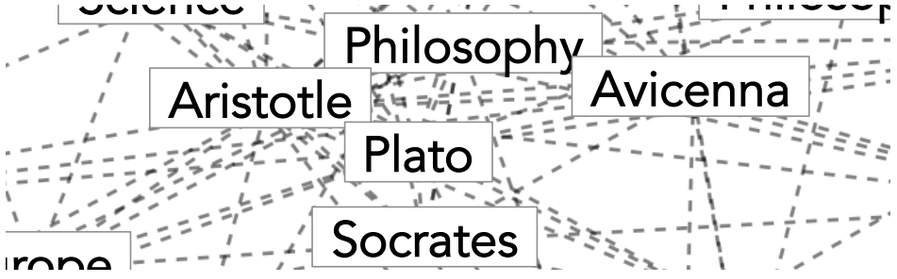
エンティティ名	エンティティ(見出し語)
NHK連続テレビ小説	連続テレビ小説
半分、青い。	半分、青い。
永野芽郁	永野芽郁
コラムニスト	コラムニスト
木村隆志	木村隆志

Wikipediaから単語とエンティティのベクトル表現を学習する

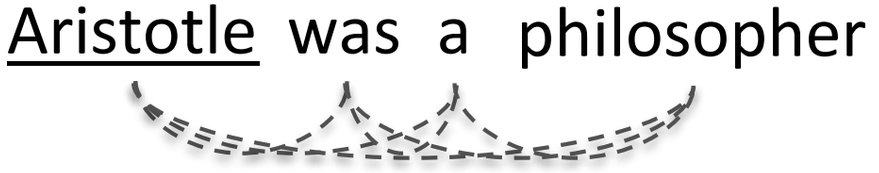
Wikipedia2Vec: 単語とエンティティのベクトル表現の同時学習

- Word2vecを拡張して、単語に加えてWikipediaエンティティを、それぞれ同一の空間にマップするベクトル表現を学習する
  - エンティティは、それぞれ固有のURLを持つため、単語のような曖昧性や表記揺れがない
    - 曖昧性: Apple 食べ物 or Apple Inc.
    - 表記揺れ: ニューヨーク or New York
  - エンティティに関する表現を学習することで、Wikipediaに含まれる世界の事物（固有名詞、専門用語など）の知識を自然にモデル化できる
- エンティティリンキングと組み合わせることで、様々な実タスクに応用できる
  - Wikipediaに記述されているエンティティに関する知識をタスクを解くのに役立てることが出来る

Wikipedia2Vec: 単語とエンティティの ベクトル表現の同時学習



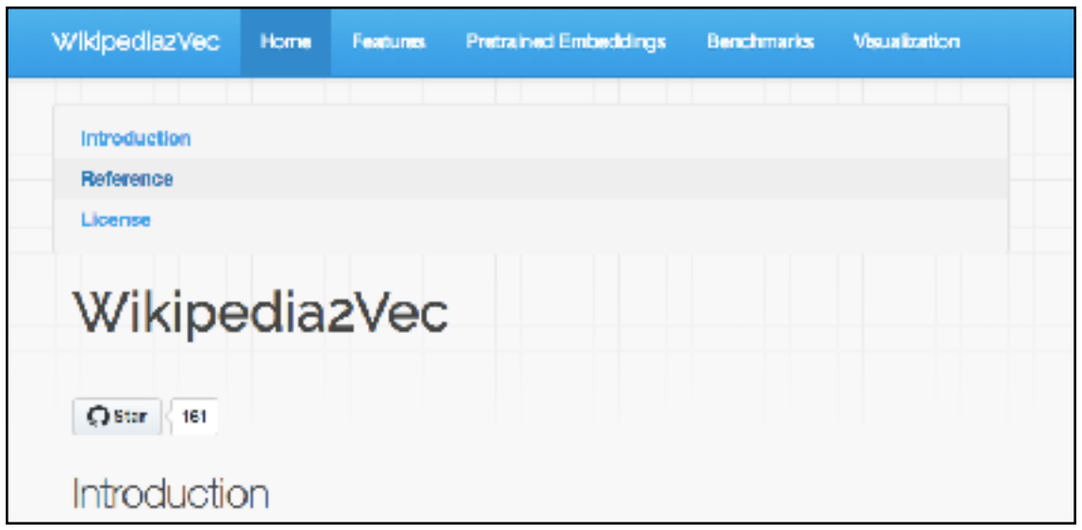
Wikipedia リンクグラフを使った文脈



単語及びリンクの周辺単語を使った文脈

- Wikipediaから単語とエンティティを同一のベクトル空間に置く分散表現を学習するskip-gram (word2vec) を拡張したモデルを採用
- 従来の単語ベースのskip-gramに下記の二つのコンテクストを追加:
  - **KB graph model:** エンティティが与えられた際に、そのエンティティに対してリンクしている他のエンティティを予測するモデル（Wikipediaのリンクのグラフをベクトル空間で表現）
  - **Anchor context model:** エンティティが与えられた際に、そのエンティティを指しているリンクの周辺の単語を予測するモデル

Wikipedia2Vec: オープンソース化



https://wikipedia2vec.github.io

- Word2vecのC実装やGensimと同等 or 高速に動作する実装をオープンソース化

実装と、事前学習済みモデルを12個の言語で公開しています

https://wikipedia2vec.github.io

学習済みモデルのある言語: 日本語, 英語, アラビア語, 中国語, オランダ語, フランス語, ドイツ語, イタリア語, ポーランド語, ポルトガル語, ロシア語, スペイン語

https://github.com/ikuyamada/wikipedia-nlp

Wikipedia2Vecの高速化

```
cdef inline void _train_pair(int32_t index1, int32_t index2, float32_t alpha, int32_t
negative, int32_t [:] neg_table) nogil:
```

単語やエンティティのペアからの訓練を行う関数の定義。GILを外した (nogil) inline関数を定義できる

```
f_dot = <float32_t>(blas.sdot(&dim_size, &syn0[index1, 0], &one, &syn1[index, 0], &one))
```

BLASを用いた内積計算の例。NumPyの行列をfloat32のポインタとして渡す

- Pythonベースで実装し、実装の大半をCythonを用いてCに変換
- skip-gramモデルでは、特に学習対象である単語やエンティティのペアからベクトルを更新する処理を高速に行う必要がある
  - Cythonを用いてPython側にデータを戻さずに処理する (単なるCのinline関数; Global interpreter lockも外せる!)
  - ベクトルを含んだNumPy行列を、CythonのTyped Memoryviewを用いて、float32のC配列とみなして、共有メモリに置いてロック無しで並列に更新する
  - 基本線型代数操作ライブラリ (BLAS) の関数 (sdot, saxpy) をC配列のポインタを引数として直接呼ぶことで、ベクトルの更新やベクトル同士の内積計算を高速化
  - Cythonによって生成されたCコードをみながら余分な処理が入っていないかを確認

Wikipedia2Vecによるベクトル表現の学習

- Wikipediaダンプのダウンロード

```
% wget https://dumps.wikimedia.org/jawiki/latest/jawiki-latest-pages-articles.xml.bz2
```

- 学習の実行

```
% wikipedia2vec train enwiki-latest-pages-articles.xml.bz2 OUT_FILE
```

- パラメータはたくさんあるので、ドキュメントを参照してください
- もちろん、学習済み日本語モデルをダウンロードして使うことも可能です

Wikipedia2Vec: 単語・エンティティ表現の評価

単語表現の単語類似度による評価:

Dataset	Wikipedia2Vec	gensim
MEN-TR-3k	**0.749**	0.7315
RG-65	**0.7837**	0.7582
SimLex999	**0.3815**	0.3471
WS-353-ALL	**0.6952**	0.6933
WS-353-REL	0.6233	**0.625**
WS-353-SIM	0.7597	**0.7833**

単語表現の単語アナロジーによる評価:

Dataset	Wikipedia2Vec	gensim
GOOGLE ANALOGY (Semantic)	**0.7892**	0.782
GOOGLE ANALOGY (Syntactic)	**0.6812**	0.5783

多言語単語表現の単語アナロジーによる評価:

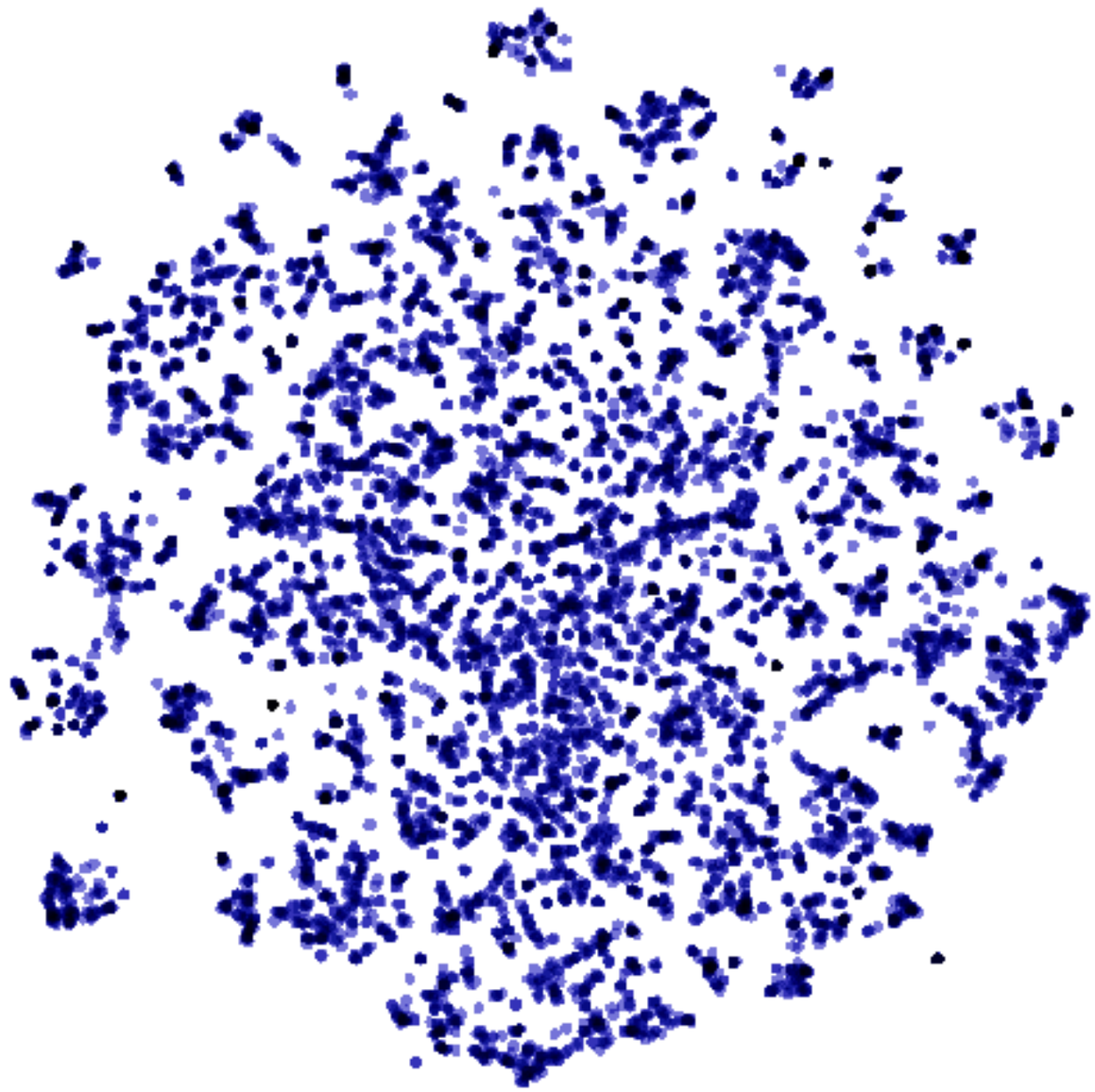
language	Wikipedia2Vec	fastText (Grave et.al)
German	**0.617**	0.61
French	**0.68**	0.642
Spanish	**0.574**	**0.574**
Portuguese	0.53	**0.54**
Polish	0.516	**0.534**
Chinese	0.572	**0.631**

エンティティ表現のエンティティ類似度 (KORE) による評価:

Category	Wikipedia2Vec	RDF2Vec (Ristoski et.al)
IT companies	**0.7934**	0.743
Hollywood Celebrities	0.6887	**0.734**
Television Series	**0.6415**	0.635
Video Games	**0.7261**	0.669
Chuck Norris	**0.6286**	0.628
All	**0.7084**	0.692

- 学習された単語・エンティティ表現をスタンダードなデータセットを用いて評価
- 単語表現では、単語類似度と単語アナロジーにおいて、gensimライブラリを用いてWikipediaコーパスから学習した単語表現に比べて良い性能を達成
- 単語表現の多言語での評価では、単語アナロジーにおいて、FacebookのfastTextとほぼ同等の性能を達成
- エンティティ類似度 (KOREデータセット) では、RDF2Vecを越えて、SOTAを達成

Wikipedia2Vec: 視覚化



https://wikipedia2vec.github.io/visualization/

Wikipedia2Vecとエンティティリンキングを使ってクイズを解くAIを作る

早押しクイズAIを解くニューラルネットワーク

Human-Computer Question Answering Match @ NIPS 2017

- Wikipedia2Vecとエンティティリンキングを用いたニューラルネットワークを使って、早押しでクイズに答えるAIを開発
- 世界最大の機械学習に関する国際会議「NIPS」で開催されたクイズゲームである「Quiz Bowl」コンペティションに出場
- AIシステム同士での対戦のあと、人間のチームで構成されるクイズ王チームとNIPSのイベント中に対戦

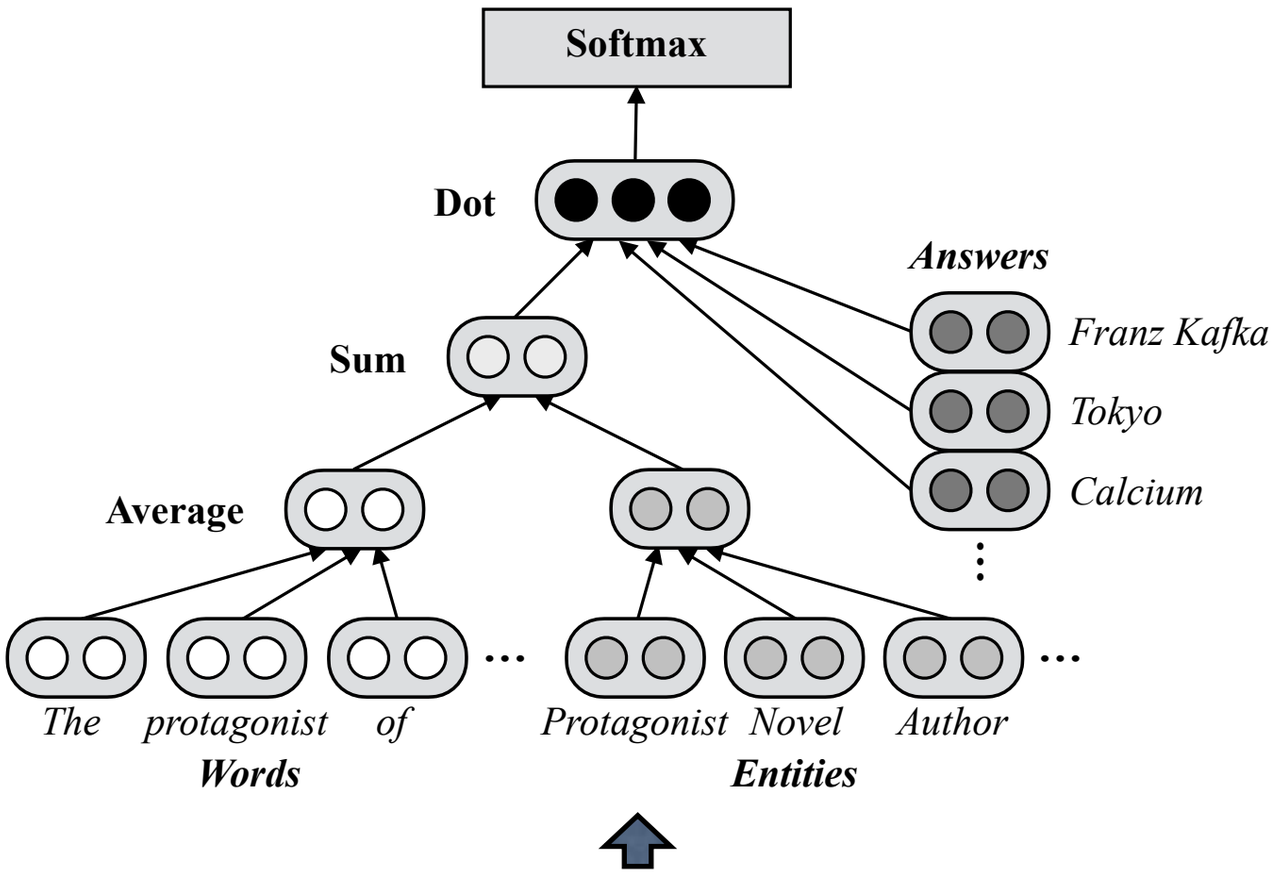
Quiz Bowlとは？

With the assistence of his chief minister, the Duc de Sully, he lowered taxes on peasantry, promoted economic recovery, and instituted a tax on the Paulette. Victor at Ivry and Arquet, he was excluded from succession by the Treaty of Nemours, but won a great victory at Coutras.

Henry IV of France

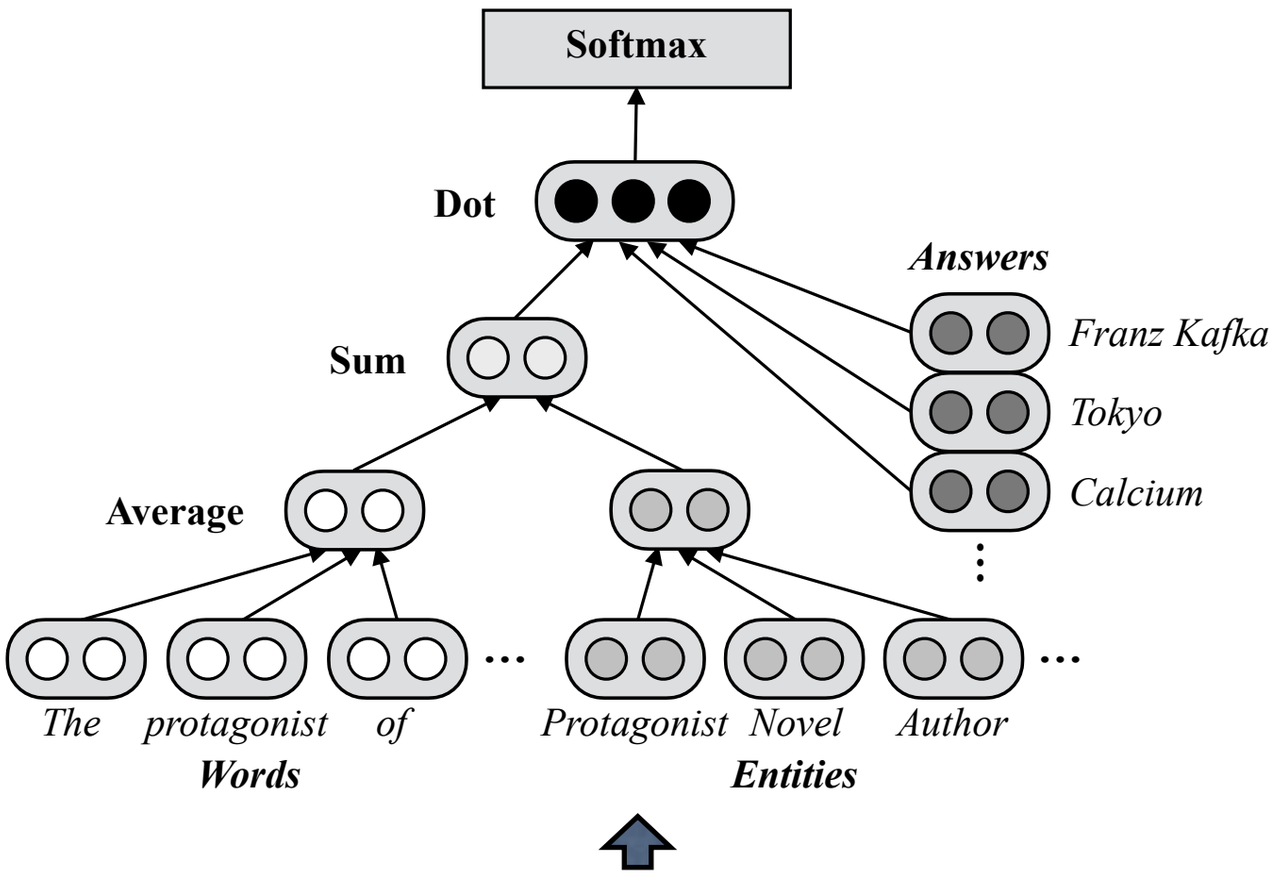
- Quiz Bowlは、英語圏を中心に幅広く人気があるクイズゲームで、米国を中心に年間を通じて多数のクイズボウルのトーナメントが開催されている
- 質問が与えられて、文が説明している解答 (**Wikipediaエンティティ**) を予測するタスク
- 解答は、全てWikipediaのエンティティであるため、エンティティで構成される回答候補に対するテキスト（質問文）の分類問題として解くことができる

クイズをテキスト分類で解く



- テキスト分類を用いて、質問に対して解答を予測するニューラルネットワーク
- ヒントを逃さずに、早い段階で解答するため、質問中の単語だけでなく出現したエンティティも入力する
  - 質問中の単語とエンティティから、解答のエンティティを予測する
- Wikipedia2Vecを用いて、質問文中の単語、質問文中のエンティティ、解答のエンティティに対応するベクトルを初期化
- エンティティは、質問中からエンティティリンキングして抽出

クイズをテキスト分類で解く



- 質問中の単語、質問中のエンティティ、解答をそれぞれベクトル p_w, q_e, a_eにマップ
- 単語のベクトル (p_w) とエンティティのベクトル (q_e) の平均を計算し、それらを加算し、質問の特徴ベクトル (v_D)として用いる

$$\mathbf{v}_{D_w} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{W}_w\mathbf{p}_{w_n} \quad \mathbf{v}_{D_e} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{W}_e\mathbf{q}_{e_k}$$

$$\mathbf{v}_D = \mathbf{v}_{D_w} + \mathbf{v}_{D_e}$$

- 質問の特徴ベクトル (v_D) と、解答に対応するベクトル (a_e) の内積をベースにsoftmaxを使って解答に対応する確率を計算

$$\hat{y}_{e_t} = \frac{\exp(\mathbf{a}_{e_t}{}^{\top}\mathbf{v}_D)}{\sum_{e' \in \Gamma}\exp(\mathbf{a}_{e'}{}^{\top}\mathbf{v}_D)}$$

実装と訓練

- 約10万件の質問ー解答のペアをコンペティション主催者の配布したデータセットをベースに作成し、訓練
- Wikipedia2Vecで初期化した解答エンティティに対応するベクトルは更新せずに、そのほかの全てのパラメータを更新
  - 与えられた質問文をWikipedia2Vecのエンティティの空間にマップするように解く
- 学習時に質問をランダムな位置で切って入力することで、早押しに対応
- PyTorchで実装し、Adamで学習率を管理

コンペティションの結果

Name	Accuracy
Our system	**0.85**
System 1	0.675
System 2	0.6
Baseline	0.55

AI間の対戦でのシステムの解答精度

クイズエクスパートとの対戦の様子

- AIシステム間の対戦で優勝
- 人間のクイズ王チームと対戦し、大差（465 対 200）で勝利！
- 対戦者は著名なクイズ王を含む六名：
  - Raj Dhuwalia氏: クイズ番組「ジェパディ!」の優勝者で、テレビ番組「Who Wants to be a Millionaire)」で過去に25万ドルを獲得
  - David Farris氏: 数学者で、複数の米国のクイズコンペティションで優勝した経験を持つ

まとめ

- Wikipediaを自然言語処理に活用するための便利なテクニックを４つ紹介しました
  1. Wikipediaをテキストコーパス として使う
  2. Wikipediaから表記揺れしやすい文字ペアを抽出する
  3. Wikipediaエンティティ辞書を活用する
  4. Wikipediaから単語とエンティティのベクトル表現を学習する
- 事例として、クイズを解くAIを紹介しました

- 自然言語処理の業務にぜひWikipediaを活用してみてください！

ありがとうございました!

コード: https://github.com/ikuyamada/wikipedia-nlp

STUDIO OUSIA