

# Python による日本語自然言語処理 ～系列ラベリングによる実世界テキスト分析～

PyCon JP 2019

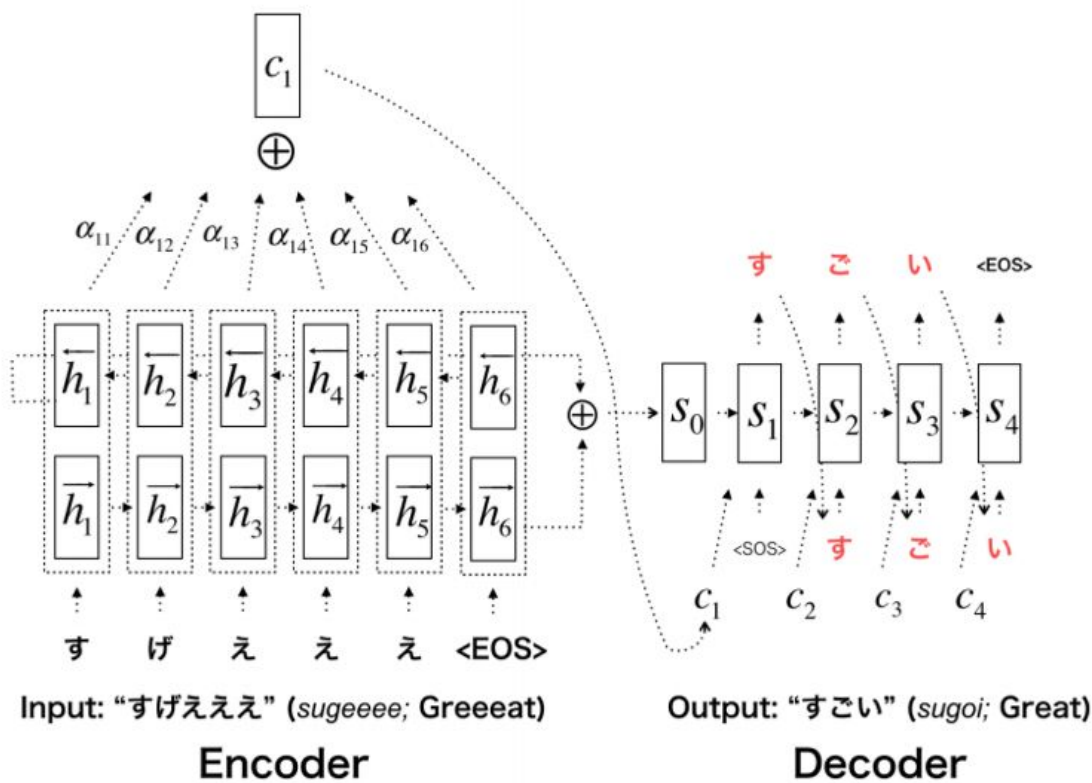
池田 大志

# 自己紹介

- 名前
  - 池田 大志 (Taishi Ikeda)
- 略歴
  - 2015 - 2017 : 奈良先端大 自然言語処理学研究室 出身
  - 2017 - 現在 : 企業にて自然言語処理に関する研究開発に従事
- これまでの取り組み
  - WNUT@COLING2016
  - 形態素解析の今とこれから@言語処理学会2018
  - ポスターセッション@PyCon JP2018
  - 全国大会@人工知能学会2019

# WNUT@COLING2016 での発表

- Japanese Text Normalization with Encoder-Decoder Model
- 口語表現「すげえええ」を辞書表現「すごい」に正規化する研究



# ポスターセッション@PyCon2018 での発表

- NLP初心者のための日本語単語分割/品詞タグ付けツールの紹介

**NLP（自然言語処理）初心者のための単語分割/品詞タグ付けツールの紹介**

**はじめに**  
現在、RNNによる日本語単語分割・品詞タグ付けツールの開発を行っています。  
本ツールは、これから自然言語処理を始めようと考えている Python ユーザーの方にも使いやすいツールを目指し開発されており、オープンソースの Python ライブラリとして公開中です。

**基本的な使い方**  
import nagisa と nagisa.tagging の「たった二行のコード」で単語分割と品詞タグ付けを行うことができます。

```
>>> import nagisa
>>> text = "Pythonで簡単に使えるツールです"
>>> words = nagisa.tagging(text)
>>> print(words)
Python/名詞 で/助詞 簡単/形容詞 に/助動詞 使える/動詞 ツール/名詞 です/助動詞
```

**顔文字やURLに対して適切な解析**  
文字単位の Bidirectional-LSTM を用いて入力文の単語境界を求めるので、顔文字やURL を一つの単語として出力します。

```
>>> text = "(人+。+)"
>>> words = nagisa.tagging(text)
>>> print(words)
(人+。+)/補助記号 こんばんは/感動詞 /補助記号
```

**インストール方法**  
pip install nagisa

**ツールの特徴**  
Bidirectional-LSTM を用いて系列ラベリングによる単語分割と品詞推定を行います。

**単語分割の結果を調整する方法**  
引数 single\_word\_list に任意の単語を追加することで、単語分割の調整が可能です。

```
>>> text = "ニューラルネットワークを使っています。"
>>> print(nagisa.tagging(text))
ニューラル/名詞 ネットワーク/名詞 を/助詞 使っ/動詞 て/助動詞 ます/助動詞 /補助記号
```

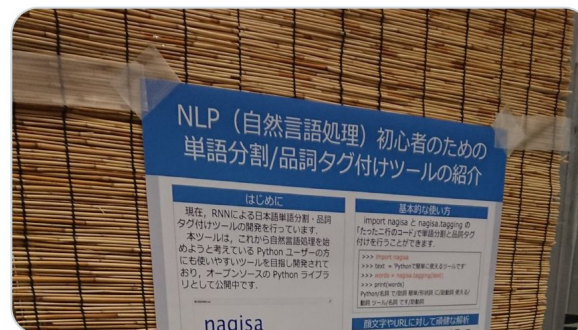
**PyCon JP 2018**  
ひろがる Python

**nagisa**  
https://github.com/taishi-i/nagisa



nadare @Py2K4 · 9月18日

形態素解析ライブラリ「nagisa」のポスター一つでインストールできる手軽さと一単語として認識したい単語を簡単に追加できる便利さがあってmecabより小回りの効く分析ツールって感じで試したい #PyConJP



2

152

500

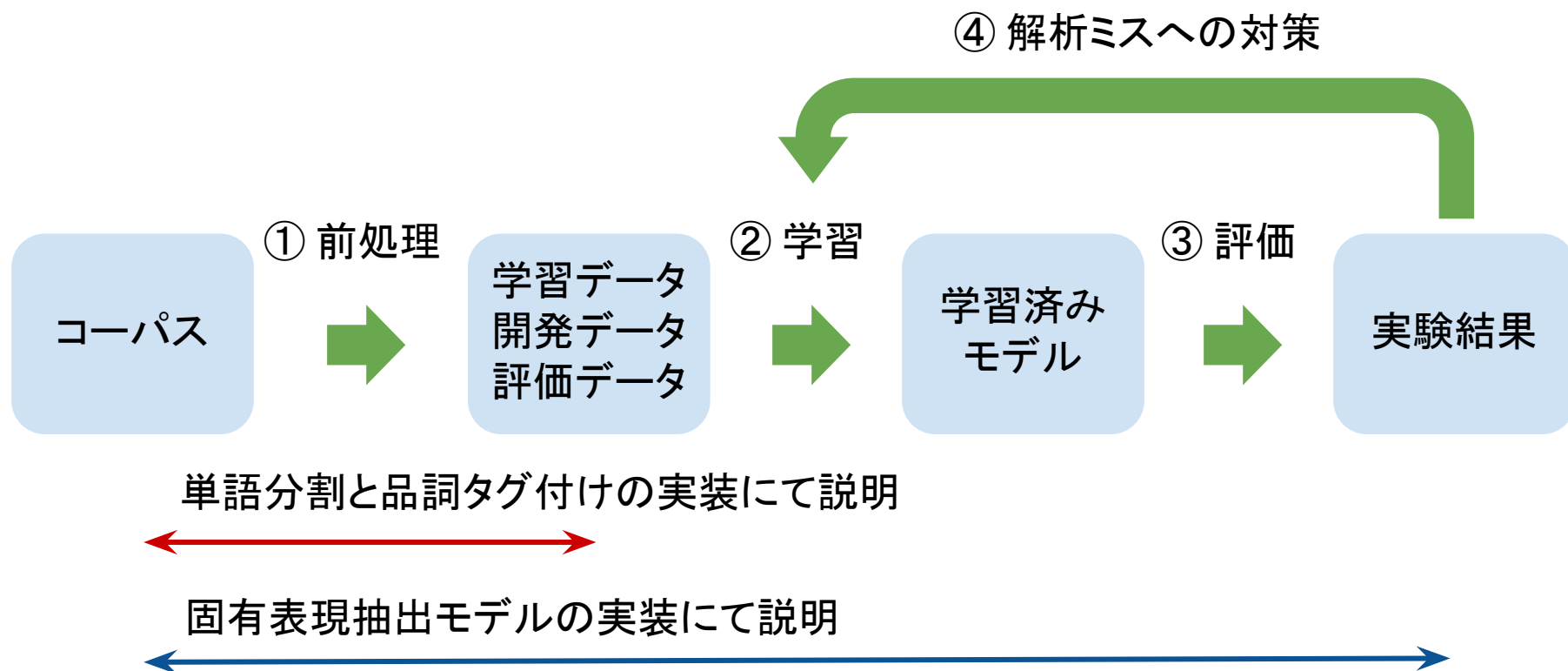


# 本発表について

- 概要
  - Python による単語分割と品詞タグ付けの実装方法の説明
  - 日本語を対象とした固有表現抽出モデルの実装方法の説明
- 対象となる聞き手
  - 自然言語処理に興味があり、テキストデータの分析を行いたい人
  - 一通り Python の基礎を勉強して、次に何か作ってみたい人
- ハンズオン形式による発表
  - 本発表を聞きつつ、手元で Python プログラムを動かしながら、自然言語処理を体験してもらう内容
- 本スライドとサンプルコードは GitHub にて公開中
  - <https://github.com/taishi-i/nagisa-tutorial-pycon2019>

# 本発表の目的

- 自然言語処理を行うための一連の実装方法を理解してもらい、開発現場で**自然言語処理を活用するきっかけ**となることが本発表の目的



# 目次

- 単語分割と品詞タグ付けについて
  - **nagisa とは？**
  - **nagisa の利用事例**
  - **Bidirectional LSTMs (BLSTMs) とは？**
  - **BLSTMs による単語分割**
  - **BLSTMs による品詞タグ付け**
  - **「形態素解析器」ではないのか？**
- **nagisa による単語分割と品詞タグ付けの実装方法**
  - **インストール方法**
  - **シンプルな単語分割と品詞タグ付け機能を提供**
  - **品詞による出力単語のフィルタリングが可能**
  - **ユーザー辞書の追加が容易**
  - **顔文字やURLに対して頑健な解析が可能**
  - **単語分割と品詞タグ付けの応用例**
- **Python による固有表現抽出モデルの実装方法**
  - **BLSTMs による固有表現抽出**
  - **事前準備**
  - **京都大学ウェブ文書リードコーパスの前処理**
  - **前処理スクリプトの実行**
  - **固有表現抽出モデルの学習**
  - **学習済みモデルの利用方法**
  - **固有表現ごとの正解率の確認**
  - **出力結果の確認によるエラー分析**
  - **解析ミスへの対応方法**
  - **系列ラベリングの応用例**

# 単語分割と品詞タグ付けについて



# 本発表を通じて体験する自然言語処理

入力文: Pythonで簡単に使えるツールです

# 本発表を通じて体験する自然言語処理

入力文: Pythonで簡単に使えるツールです



単語分割

Python

で

簡単

に

使える

ツール

です

# 本発表を通じて体験する自然言語処理

入力文: Pythonで簡単に使えるツールです



単語分割

Python

で

簡単

に

使える

ツール

です



品詞タグ付け

Python

名詞

で

助詞

簡単

形状詞

に

助詞

使える

動詞

ツール

名詞

です

助動詞

# 本発表を通じて体験する自然言語処理

- Python による単語分割と品詞タグ付けの実装
- 単語分割
  - 与えられた入力文を単語に分割する処理
- 品詞タグ付け
  - 与えられた入力文に対して、その文中の各単語の品詞を判定し、品詞情報を付与する処理

# nagisa

A Japanese tokenizer based on recurrent neural networks

# nagisa とは？

- 特徴
  - BLSTMs による単語分割と品詞タグ付けの機能を提供
  - 系列ラベリングモデルの学習が可能
  - `pip install nagisa` でインストール可能
- 開発方針
  - シンプルで使いやすいツールを目指し、開発を行っている
- 想定ユーザー
  - これから自然言語処理を始めようと考えている Python ユーザー

# nagisa の利用事例

- TextRank Demo
  - 抽出式文書要約のデモサイト(英語、中国語、日本語に対応)
  - 日本語の単語分割に nagisa を利用している

## Results

自治会への加入を拒否したら、「ごみ捨て場を使わせない」と言われた——。今春、新居に引っ越したという40代の女性が悩んでいるのをツイッターで見つけた。SNSで読者の困りごとを募って取材する朝日新聞「#ニュース4U(フォーユー)」取材班が、女性の話を聞きに出かけた。

女性は働きながら3人の子どもの育てるシングルマザー。大阪と京都の間に位置する大阪府高槻市の落ち着いた住宅街に中古の一軒家を買って、5月に引っ越してきた。

引っ越しのあいさつ回りをしたとき、「自治会には入りません」と近所の人に伝えた。活動に携わる時間がなかった。

すると、自治会側はこう返してきたという。「ごみ集積所にごみを出せなくなります」。集積所は自治会が管理している。女性は「自治会には入らないが、集積所の掃除はする」と伝えたが、「役員をしたくなくて自治会に入らない人が増えると困る」と断られた。ごみを捨てられず、自宅にどんどんたまっていった。子どもに「ごみ捨て場がそこにあるのに、なんで捨てたらあかんの?」と聞かれた。

女性は市役所にどうすればいいか相談した。市は当初、「住民同士の話し合いで解決してほしい」と回答したという。

[https://github.com/ceshine/textrank\\_demo](https://github.com/ceshine/textrank_demo)

# nagisa の利用事例

- Wordless
  - 多言語(80言語以上)に対応した語学学習のためのコーパスツール
  - 日本語の単語分割と品詞タグ付けに nagisa を利用している

**Wordless** An Integrated Corpus Tool  
with Multi-language Support  
for the Study of Language, Literature and Translation

<https://github.com/BLKSerene/Wordless>



# nagisa の利用事例

- 日本語形態素解析エンジン nagisa は古典中国語(漢文)を学習できるのか
  - 漢文の形態素解析の学習に nagisa を利用した事例
  - 品詞タグ付けのF1値 nagisa: 90.07% UDPipe: 87.8%

```
>>> lzh=nagisa.Tagger(vocabs="lzh_kyoto-nagisa.vocabs",params="lzh_kyoto-nagisa.params",hp="lzh_kyoto-nagisa.hp").tagging
```

```
>>> s=lzh("不入虎穴不得虎子")
```

不/v, 副詞, 否定, 無界 入/v, 動詞, 行為, 移動 虎/n, 名詞, 主体, 動物 穴/n, 名詞, 固定物, 地形 不/v, 副詞, 否定, 無界 得/v, 動詞, 行為, 得失 虎/n, 名詞, 主体, 動物 子/n, 名詞, 人, 関係

一字ずつ全てが切られていて、品詞も妥当なようだ。次に「有兄子曰甫」を、形態素解析してみよう。

<https://srad.jp/~yasuoka/journal/632602/>

# nagisa の利用事例

- NTT's Machine Translation Systems for WMT19 Robustness Task
  - 口語表現を対象とした機械翻訳の研究にて nagisa を利用した事例
  - 入力文に含まれる顔文字を抽出する処理に利用している

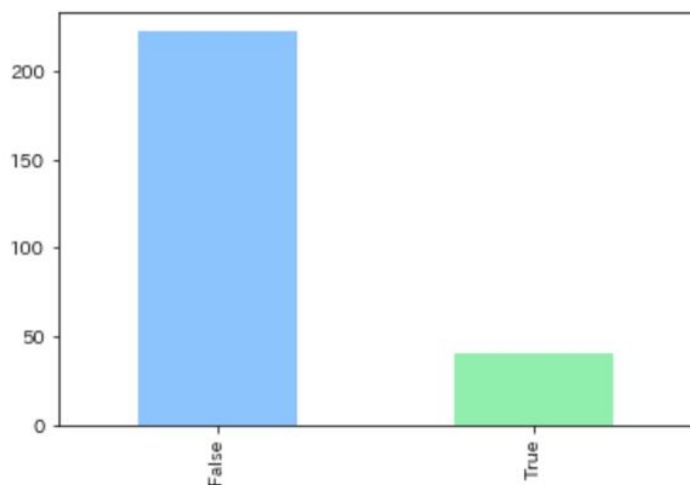
	Output	BLEU+1
Ref	Kawaii 🎵 (*・ω・人)	
NTT	Cute (*・ω・人)	76.0
NLE	It 's cute .	0.0

<https://www.aclweb.org/anthology/W19-5365>

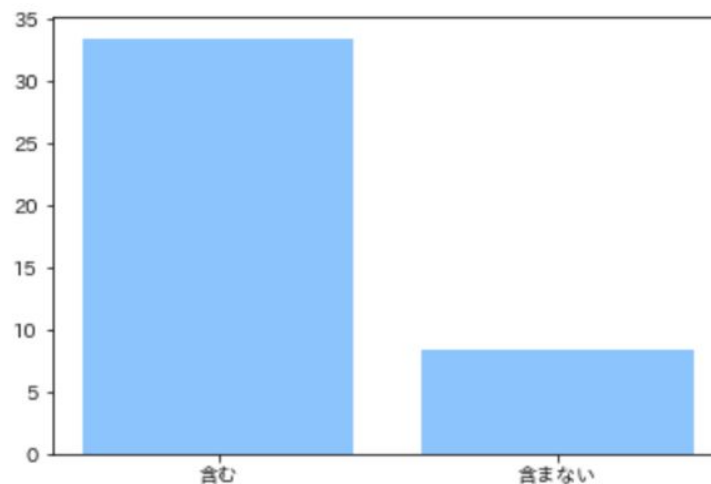
# nagisa の利用事例

- Python で自分のどんなツイートが「いいね」されやすいかを分析してみた
  - 過去のツイートの分析で nagisa を利用した事例
  - 彼氏に関するツイートをするといいいねされやすいとの結果

▼彼氏を含む／含まないツイート数



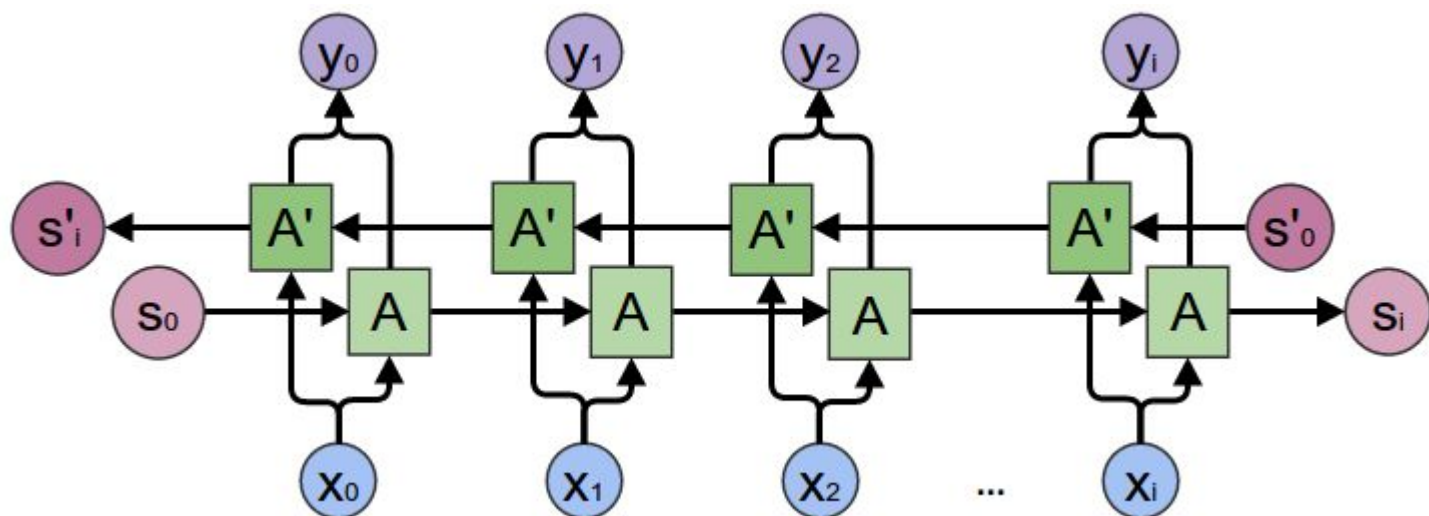
▼彼氏を含む／含まないツイートの平均いいね数



<https://www.pc-koubou.jp/magazine/24224>

# Bidirectional LSTMs (BLSTMs) とは？

- 前向き LSTM と後向き LSTM を計算することで、  
入力全体の情報を各時刻で考慮できるニューラルネットワーク

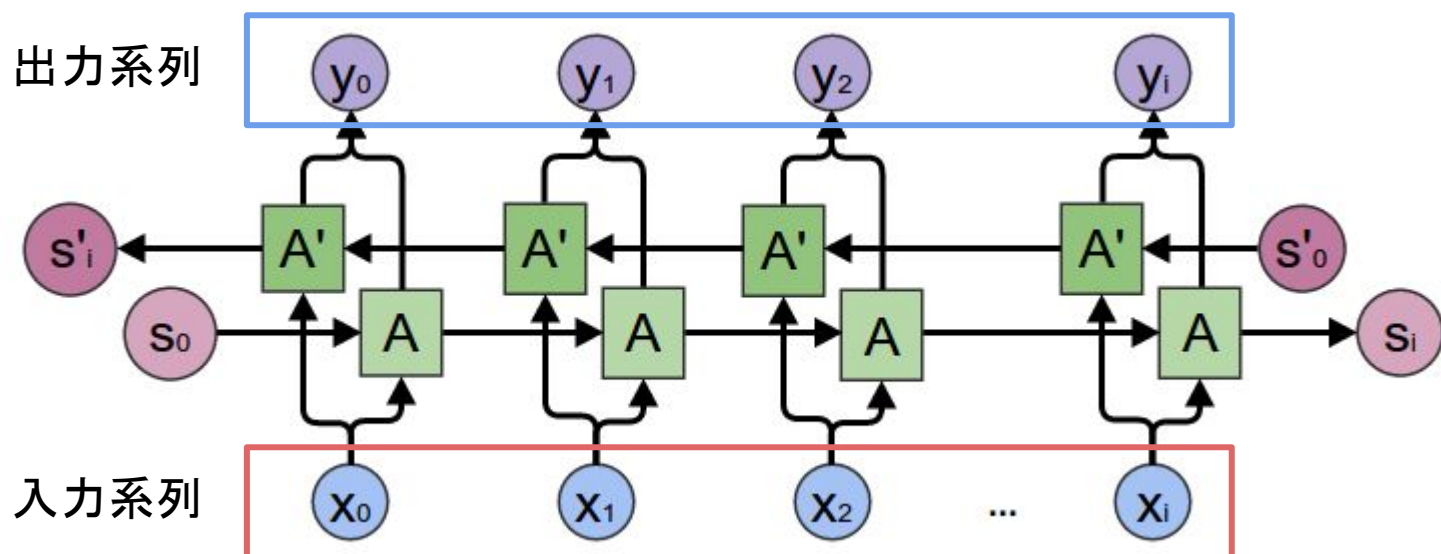


Neural Networks, Types, and Functional Programming:

<http://colah.github.io/posts/2015-09-NN-Types-FP/>

# Bidirectional LSTMs (BLSTMs) とは？

- 前向き LSTM と後向き LSTM を計算することで、  
入力全体の情報を各時刻で考慮できるニューラルネットワーク

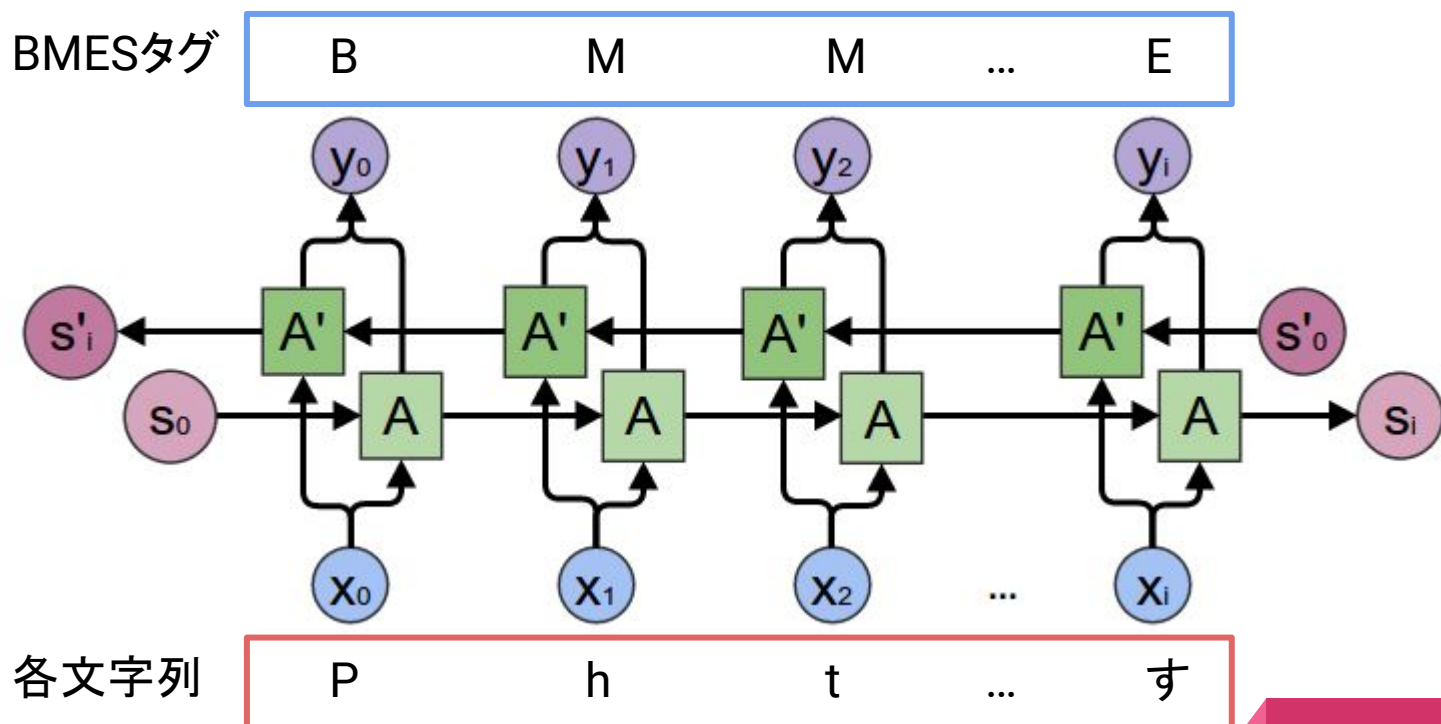


Neural Networks, Types, and Functional Programming:

<http://colah.github.io/posts/2015-09-NN-Types-FP/>

# BLSTMs による単語分割

- 入力文の各文字列を BLSTMs の入力として BMES タグを予測する



# BLSTMs による単語分割

- 各文字に対して B・M・E・S の4つのタグを付与することで単語分割を行う
  - **B**egin: 単語の開始文字とする
  - **M**iddel: 単語の途中文字とする
  - **E**nd: 単語の終了文字とする
  - **S**ingle: 一文字を単語とする

入力文: Pythonで簡単に使えるツールです



各文字に対してタグを付与する

BMMMM

S

BE

S

BME

BME

BE

Python

で

簡単

に

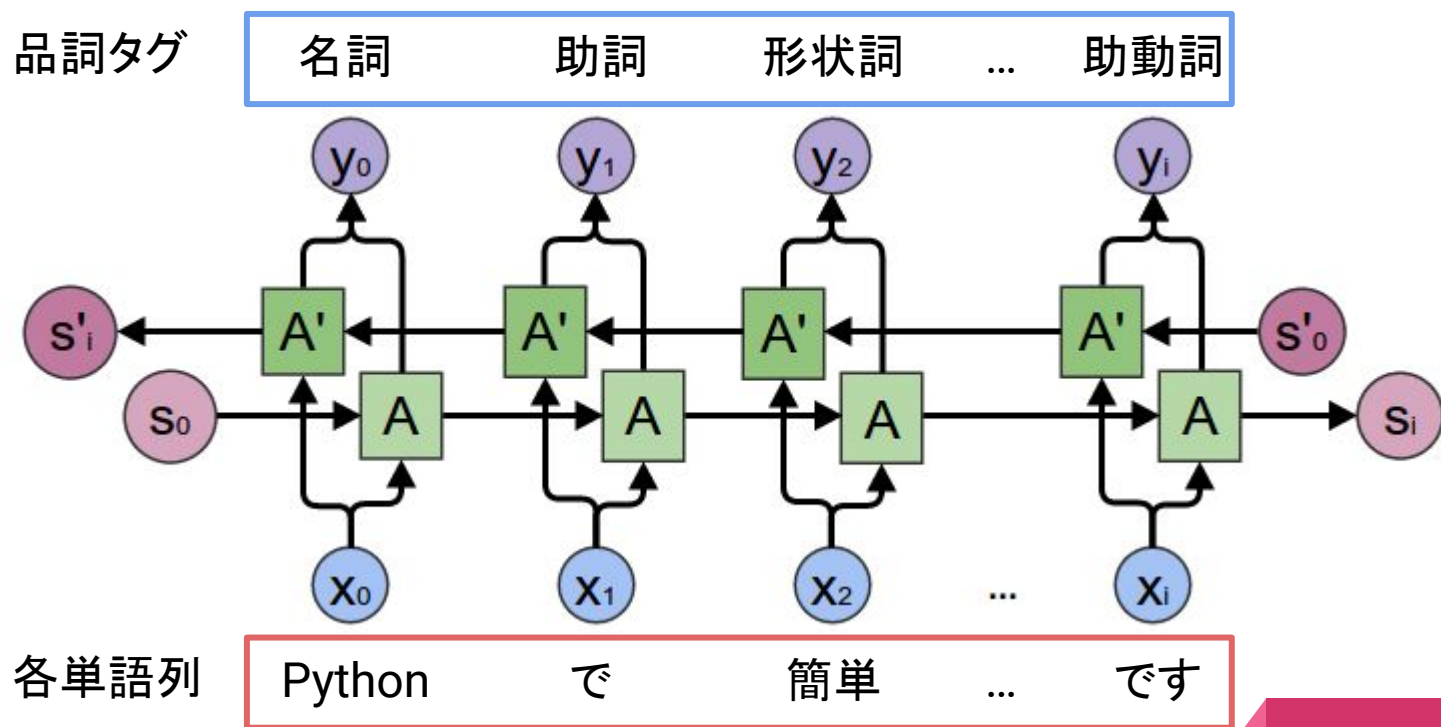
使える

ツール

です

# BLSTMs による品詞タグ付け

- 入力文の各単語列を BLSTMs の入力として品詞タグを予測する





# BLSTMs による品詞タグ付け

- 各単語に対して24個の品詞タグを付与する
  - 名詞, 助詞, 接尾辞, 動詞, 連体詞, 助動詞, 形容詞 など
  - 文字単位 BLSTMs のベクトルを単語ベクトルに結合することで、未知語に対して頑健な品詞付与が可能となる

入力文: Pythonで簡単に使えるツールです



各単語に対してタグを付与する

名詞

助詞

係詞

形状詞

動詞

名詞

助動詞

Python

で

簡単

に

使える

ツール

です

# 「形態素解析器」ではないのか？

- 一般には、以下の一連の処理のことを形態素解析と呼ぶ(近代科学社「形態素解析の理論と実装」より)
  - 単語(もしくは形態素)への分割(分かち書き)
  - 品詞の推定
  - 語形変化の処理(原型を求める)



nagisa による  
単語分割と品詞タグ付けの  
実装方法

# サンプルコードへのリンク

- Google Colaboratory

- [https://colab.research.google.com/github/taishi-i/nagisa-tutorial-pycon2019/blob/master/notebooks/basic\\_usage.ipynb?hl=ja](https://colab.research.google.com/github/taishi-i/nagisa-tutorial-pycon2019/blob/master/notebooks/basic_usage.ipynb?hl=ja)

- GitHub

- [https://github.com/taishi-i/nagisa-tutorial-pycon2019/blob/master/notebooks/basic\\_usage.ipynb](https://github.com/taishi-i/nagisa-tutorial-pycon2019/blob/master/notebooks/basic_usage.ipynb)



## Python による日本語自然言語処理 ～系列ラベリングによる実世界テキスト分析～

- 発表スケジュール: 2019/09/16 14:40-15:10 (30min)
- 場所: D会議室
- レベル: Beginner
- 発表/資料: JP
- 発表スライド: [nagisa-tutorial-pycon2019.pdf](#)

## nagisa による単語分割と品詞タグ付けの実装方法

1. 基本的な使い方: [basic\\_usage.ipynb](#) [Colab notebook]
2. ワードクラウドの作成: [word\\_cloud.ipynb](#) [Colab notebook]

# インストール方法

- pip によるインストールが可能

```
$ pip install nagisa
```

- サポートする OS と Python のバージョン
  - Linux/macOS
    - Python 2.7, 3.5, 3.6, 3.7
  - Windows (64bit)
    - Python 3.5, 3.6, 3.7

# シンプルな単語分割と品詞タグ付け機能を提供

- `nagisa.tagging(text)` で単語分割と品詞タグ付けを実行する

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.tagging(text)
>>> print(tokens)
Python/名詞 で/助詞 簡単/形状詞 に/助動詞 使える/動詞 ツール/
名詞 です/助動詞
>>> print(tokens.words)
['Python', 'で', '簡単', 'に', '使える', 'ツール', 'です']
>>> print(tokens.postags)
['名詞', '助詞', '形状詞', '助動詞', '動詞', '名詞', '助動
詞']
```

# シンプルな単語分割と品詞タグ付け機能を提供

- `nagisa.tagging(text)` で単語分割と品詞タグ付けを実行する

インポート

```
>>> import nagisa  
  
>>> text = 'Pythonで簡単に使えるツールです'  
>>> tokens = nagisa.tagging(text)  
>>> print(tokens)  
Python/名詞 で/助詞 簡単/形状詞 に/助動詞 使える/動詞 ツール/  
名詞 です/助動詞  
>>> print(tokens.words)  
['Python', 'で', '簡単', 'に', '使える', 'ツール', 'です']  
>>> print(tokens.postags)  
['名詞', '助詞', '形状詞', '助動詞', '動詞', '名詞', '助動  
詞']
```

# シンプルな単語分割と品詞タグ付け機能を提供

- `nagisa.tagging(text)` で単語分割と品詞タグ付けを実行する

入力テキスト



```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.tagging(text)
>>> print(tokens)
Python/名詞 で/助詞 簡単/形状詞 に/助動詞 使える/動詞 ツール/
名詞 です/助動詞
>>> print(tokens.words)
['Python', 'で', '簡単', 'に', '使える', 'ツール', 'です']
>>> print(tokens.postags)
['名詞', '助詞', '形状詞', '助動詞', '動詞', '名詞', '助動
詞']
```



# シンプルな単語分割と品詞タグ付け機能を提供

- `nagisa.tagging(text)` で単語分割と品詞タグ付けを実行する

解析の実行



```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.tagging(text)
>>> print(tokens)
Python/名詞 で/助詞 簡単/形状詞 に/助動詞 使える/動詞 ツール/
名詞 です/助動詞
>>> print(tokens.words)
['Python', 'で', '簡単', 'に', '使える', 'ツール', 'です']
>>> print(tokens.postags)
['名詞', '助詞', '形状詞', '助動詞', '動詞', '名詞', '助動
詞']
```

# シンプルな単語分割と品詞タグ付け機能を提供

- `nagisa.tagging(text)` で単語分割と品詞タグ付けを実行する

結果の出力

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.tagging(text)
>>> print(tokens)
Python/名詞 で/助詞 簡単/形状詞 に/助動詞 使える/動詞 ツール/
名詞 です/助動詞
>>> print(tokens.words)
['Python', 'で', '簡単', 'に', '使える', 'ツール', 'です']
>>> print(tokens.postags)
['名詞', '助詞', '形状詞', '助動詞', '動詞', '名詞', '助動
詞']
```

# シンプルな単語分割と品詞タグ付け機能を提供

- `nagisa.tagging(text)` で単語分割と品詞タグ付けを実行する

単語列リストの取得

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.tagging(text)
>>> print(tokens)
Python/名詞 で/助詞 簡単/形状詞 に/助動詞 使える/動詞 ツール/
名詞 です/助動詞
>>> print(tokens.words)
['Python', 'で', '簡単', 'に', '使える', 'ツール', 'です']
>>> print(tokens.postags)
['名詞', '助詞', '形状詞', '助動詞', '動詞', '名詞', '助動
詞']
```

# シンプルな単語分割と品詞タグ付け機能を提供

- `nagisa.tagging(text)` で単語分割と品詞タグ付けを実行する

品詞列リストの取得

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.tagging(text)
>>> print(tokens)
Python/名詞 で/助詞 簡単/形状詞 に/助動詞 使える/動詞 ツール/
名詞 です/助動詞
>>> print(tokens.words)
['Python', 'で', '簡単', 'に', '使える', 'ツール', 'です']
>>> print(tokens.postags)
['名詞', '助詞', '形状詞', '助動詞', '動詞', '名詞', '助動
詞']
```

# 品詞による出力単語のフィルタリングが可能

- 引数に品詞を指定することで、出力単語のフィルタリングを行う

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.extract(
    text, extract_postags=['名詞']
)
>>> print(tokens)
Python/名詞 ツール/名詞
```

# 品詞による出力単語のフィルタリングが可能

- 引数に品詞を指定することで、出力単語のフィルタリングを行う

取得したい品詞を指定

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.extract(
    text, extract_postags=['名詞']
)
>>> print(tokens)
Python/名詞 ツール/名詞
```

# 品詞による出力単語のフィルタリングが可能

- 引数に品詞を指定することで、出力単語のフィルタリングを行う

**nagisa.extract の実行**

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.extract(
    text, extract_postags=['名詞']
)
>>> print(tokens)
Python/名詞 ツール/名詞
```

# 品詞による出力単語のフィルタリングが可能

- 引数に品詞を指定することで、出力単語のフィルタリングを行う

「名詞」のみを抽出

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.extract(
    text, extract_postags=['名詞']
)
>>> print(tokens)
Python/名詞 ツール/名詞
```



# 品詞による出力単語のフィルタリングが可能

- 引数に品詞を指定することで、出力単語のフィルタリングを行う

除外したい品詞を指定

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.filter(
    text, filter_postags=['助詞', '助動詞']
)
>>> print(tokens)
Python/名詞 簡単/形状詞 使える/動詞 ツール/名詞
```

# 品詞による出力単語のフィルタリングが可能

- 引数に品詞を指定することで、出力単語のフィルタリングを行う

nagisa.filter の実行

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.filter(
    text, filter_postags=['助詞', '助動詞']
)
>>> print(tokens)
Python/名詞 簡単/形状詞 使える/動詞 ツール/名詞
```

# 品詞による出力単語のフィルタリングが可能

- 引数に品詞を指定することで、出力単語のフィルタリングを行う

「助詞」と「助動詞」以外を抽出

```
>>> import nagisa

>>> text = 'Pythonで簡単に使えるツールです'
>>> tokens = nagisa.filter(
    text, filter_postags=['助詞', '助動詞']
)
>>> print(tokens)
Python/名詞 簡単/形状詞 使える/動詞 ツール/名詞
```

# ユーザー辞書の追加が容易

- single\_word\_list に追加した単語は、ひとつの単語として分割する

```
>>> import nagisa

>>> text = "3月に見た「3月のライオン」"
>>> print(nagisa.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3/名詞 月/名詞 の/助詞 ライオン/名詞 」/補助記号
>>> new_tagger = nagisa.Tagger(
    single_word_list=['3月のライオン']
)
>>> print(new_tagger.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3月のライオン/名詞 」/補助記号
```

# ユーザー辞書の追加が容易

- single\_word\_list に追加した単語は、ひとつの単語として分割する

入力テキスト

```
>>> import nagisa

>>> text = "3月に見た「3月のライオン」"
>>> print(nagisa.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3/名詞 月/名詞 の/助詞 ライオン/名詞 」/補助記号
>>> new_tagger = nagisa.Tagger(
    single_word_list=['3月のライオン']
)
>>> print(new_tagger.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3月のライオン/名詞 」/補助記号
```

# ユーザー辞書の追加が容易

- single\_word\_list に追加した単語は、ひとつの単語として分割する

解析結果

```
>>> import nagisa

>>> text = "3月に見た「3月のライオン」"
>>> print(nagisa.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3/名詞 月/名詞 の/助詞 ライオン/名詞 」/補助記号
>>> new_tagger = nagisa.Tagger(
    single_word_list=['3月のライオン']
)
>>> print(new_tagger.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3月のライオン/名詞 」/補助記号
```

# ユーザー辞書の追加が容易

- single\_word\_list に追加した単語は、ひとつの単語として分割する

「3月のライオン」をリストに追加

```
>>> import nagisa

>>> text = "3月に見た「3月のライオン」"
>>> print(nagisa.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3/名詞 月/名詞 の/助詞 ライオン/名詞 」/補助記号
>>> new_tagger = nagisa.Tagger(
    single_word_list=['3月のライオン']
)
>>> print(new_tagger.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3月のライオン/名詞 」/補助記号
```

# ユーザー辞書の追加が容易

- single\_word\_list に追加した単語は、ひとつの単語として分割する

**new\_tagger を定義**

```
>>> import nagisa

>>> text = "3月に見た「3月のライオン」"
>>> print(nagisa.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3/名詞 月/名詞 の/助詞 ライオン/名詞 」/補助記号
>>> new_tagger = nagisa.Tagger(
    single_word_list=['3月のライオン']
)
>>> print(new_tagger.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3月のライオン/名詞 」/補助記号
```



# ユーザー辞書の追加が容易

- single\_word\_list に追加した単語は、ひとつの単語として分割する

解析の実行

```
>>> import nagisa

>>> text = "3月に見た「3月のライオン」"
>>> print(nagisa.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3/名詞 月/名詞 の/助詞 ライオン/名詞 」/補助記号
>>> new_tagger = nagisa.Tagger(
    single_word_list=['3月のライオン']
)
>>> print(new_tagger.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3月のライオン/名詞 」/補助記号
```

# ユーザー辞書の追加が容易

- single\_word\_list に追加した単語は、ひとつの単語として分割する

解析結果

```
>>> import nagisa

>>> text = "3月に見た「3月のライオン」"
>>> print(nagisa.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3/名詞 月/名詞 の/助詞 ライオン/名詞 」/補助記号
>>> new_tagger = nagisa.Tagger(
    single_word_list=['3月のライオン']
)
>>> print(new_tagger.tagging(text))
3/名詞 月/名詞 に/助詞 見/動詞 た/助動詞 「/補助記号
3月のライオン/名詞 」/補助記号
```

# 顔文字やURLに対して頑健な解析が可能

- 文字単位の BLSTMs による解析で顔文字をひとつの単語として分割する

```
>>> import nagisa

>>> text = 'https://github.com/taishi-i/nagisaでコードを公開中(๑~ω~๑)'
```

>>> tokens = nagisa.tagging(text)

>>> print(tokens)

https://github.com/taishi-i/nagisa/URL で/助詞 コード/名詞  
を/助詞 公開/名詞 中/接尾辞 (๑~ω~๑)/補助記号

# 顔文字やURLに対して頑健な解析が可能

- 文字単位の BLSTMs による解析で顔文字をひとつの単語として分割する

入力テキスト



```
>>> import nagisa  
  
>>> text = 'https://github.com/taishi-i/nagisaでコードを公開中(๑~ω~๑)'  
>>> tokens = tagger.tagging(text)  
>>> print(tokens)  
https://github.com/taishi-i/nagisa/URL で/助詞 コード/名詞  
を/助詞 公開/名詞 中/接尾辞 (๑~ω~๑)/補助記号
```

# Janome による形態素解析

過剰な分割結果

```
https 名詞,固有名称,組織,*,*,*,https,*,*  
:// 名詞,サ変接続,*,*,*,*,://,*,*  
github 名詞,一般,*,*,*,*,github,*,*  
. 名詞,サ変接続,*,*,*,*,.,*,*  
com 名詞,一般,*,*,*,*,com,*,*  
/ 名詞,サ変接続,*,*,*,*,/,*,*  
taishi 名詞,一般,*,*,*,*,taishi,*,*  
- 名詞,サ変接続,*,*,*,*,-,*,*  
i 名詞,一般,*,*,*,*,i,*,*  
/ 名詞,サ変接続,*,*,*,*,/,*,*  
nagisa 名詞,一般,*,*,*,*,nagisa,*,*  
で 助詞,格助詞,一般,*,*,*,*,で,デ,デ  
コード 名詞,一般,*,*,*,*,コード,コード,コード  
を 助詞,格助詞,一般,*,*,*,*,を,ヲ,ヲ  
公開 名詞,サ変接続,*,*,*,*,公開,コウカイ,コーカイ  
中 名詞,接尾,副詞可能,*,*,*,*,中,チュウ,チュー  
( 名詞,サ変接続,*,*,*,*,(,*,*  
Ⓜ 記号,一般,*,*,*,*,Ⓜ,*,*  
記号,空白,*,*,*,*, ,*,*  
記号,一般,*,*,*,*,*,*  
ω 記号,アルファベット,*,*,*,*,ω,オメガ,オメガ  
記号,空白,*,*,*,*, ,*,*  
Ⓜ 記号,一般,*,*,*,*,Ⓜ,*,*  
) 名詞,サ変接続,*,*,*,*,),*,*
```

# 顔文字やURLに対して頑健な解析が可能

- 文字単位の BLSTMs による解析で顔文字をひとつの単語として分割する

解析結果

```
>>> import nagisa  
  
>>> text = 'https://github.com/taishi-i/nagisaでコードを公開中(๑~ω~๑)'  
>>> tokens = nagisa.tagging(text)  
>>> print(tokens)  
https://github.com/taishi-i/nagisa/URLで/助詞 コード/名詞  
を/助詞 公開/名詞 中/接尾辞 (๑~ω~๑)/補助記号
```

# 顔文字やURLに対して頑健な解析が可能

- 文字単位の BLSTMs による解析で顔文字をひとつの単語として分割する

## 解析結果

```
>>> import nagisa

>>> text = 'https://github.com/taishi-i/nagisaでコードを公開中(๑~ω~๑)'
```

>>> tokens = nagisa.tagging(text)

>>> print(tokens)

https://github.com/taishi-i/nagisa/URL で/助詞 コード/名詞  
を/助詞 公開/名詞 中/接尾辞 (๑~ω~๑)/補助記号



# 2019年度版 形態素解析器比較表

ツール名	初期 リリース	最新 リリース	解析手法	分割基準	解析 速度	対応言語	pip	Star	特徴
JUMAN	1992	2014	ラティス/人手によるコスト設定	JUMAN基準	○	perl, Python	△	-	豊富な意味表現の付与が可能
ChaSen	1996	2011	ラティス/HMM	ipadic, UniDic, NAIST-dic	△	コマンドラインのみ	×	-	統計処理により形態素解析を可能とした
MeCab	2006	2013	ラティス/CRF	ipadic, UniDic, JUMAN基準, NEologd	◎	C++, Python, その他ラッパー多数	○	469	日本語NLPのデファクトスタンダード
KyTea	2011	2014	点推定/SVM	UniDic	○	C++, Python	△	156	読み推定が可能
Rakuten MA	2014	2015	系列ラベリング/SCW	UniDic	○	JavaScript	×	403	100% JavaScript による実装
JUMAN++	2016	2019	ラティス/ニューラルLM	JUMAN基準	○	Python	△	173	ニューラルLMを用いることで高精度
Sudachi	2017	2019	ラティス/CRF	長・中・短の3種類	○	Java, Python	○	368	分割単位のコントロールが可能
nagisa	2018	2019	系列ラベリング/BLSTMs	UniDic	△	Python	○	135	Pythonで簡単に使える
Stanford NLP	2019	2019	系列ラベリング/BLSTMs	Universal Dependency	×	Python	○	2377	多言語に対応
GiNZA	2019	2019	ラティス/CRF	Universal Dependency	×	Python	○	233	係り受け解析も可能
SentencePiece	2017	2019	UnigramLM	Subword	◎	C++, Python	○	2977	ニューラル言語処理向けのトークナイザ



# 単語分割と品詞タグ付けの応用例

- テキストマイニング
  - アンケートの自由回答部分やユーザーレビューの解析
  - Web 上のソーシャルメディアテキストの解析



Pythonコードへのリンクは  
←ここをクリック

# Python による 固有表現抽出モデルの 実装方法

# 本発表を通じて体験する自然言語処理

入力文: カナダのモントリオール大学教授ヨシュア・ベンジオ氏

# 本発表を通じて体験する自然言語処理

入力文: カナダのモントリオール大学教授ヨシュア・ベンジオ氏



単語分割

カナダ

の

モントリ  
オール

大学

教授

ヨシュア

・

ベンジオ

氏

# 本発表を通じて体験する自然言語処理

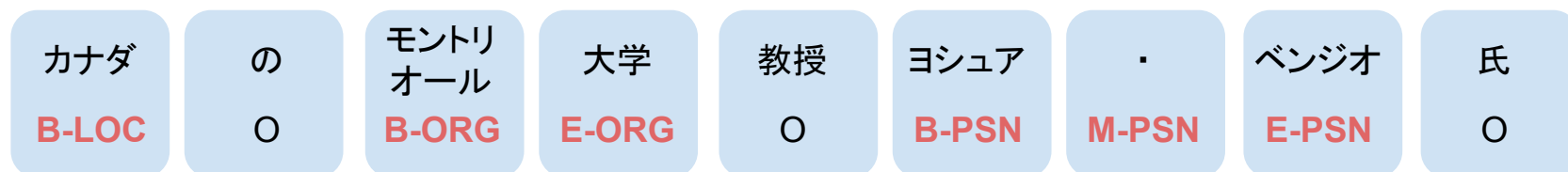
入力文: カナダのモントリオール大学教授ヨシュア・ベンジオ氏



単語分割



固有表現抽出



# 本発表を通じて体験する自然言語処理

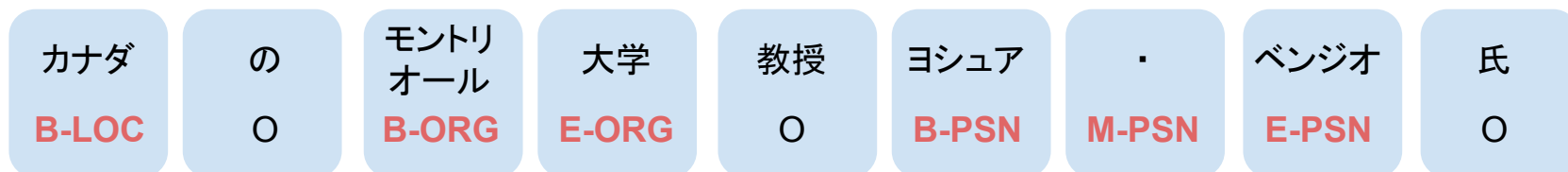
入力文: カナダのモントリオール大学教授ヨシュア・ベンジオ氏



単語分割



固有表現抽出



- 場所: カナダ
- 組織: モントリオール大学
- 人名: ヨシュア・ベンジオ

# 本発表を通じて体験する自然言語処理

- 日本語を対象とした固有表現抽出モデルの実装
- 固有表現抽出
  - 与えられた入力文に対して、その文中の各単語の固有表現を判定し、固有表現情報を付与する処理

入力文: カナダのモントリオール大学教授ヨシュア・ベンジオ氏



固有表現抽出

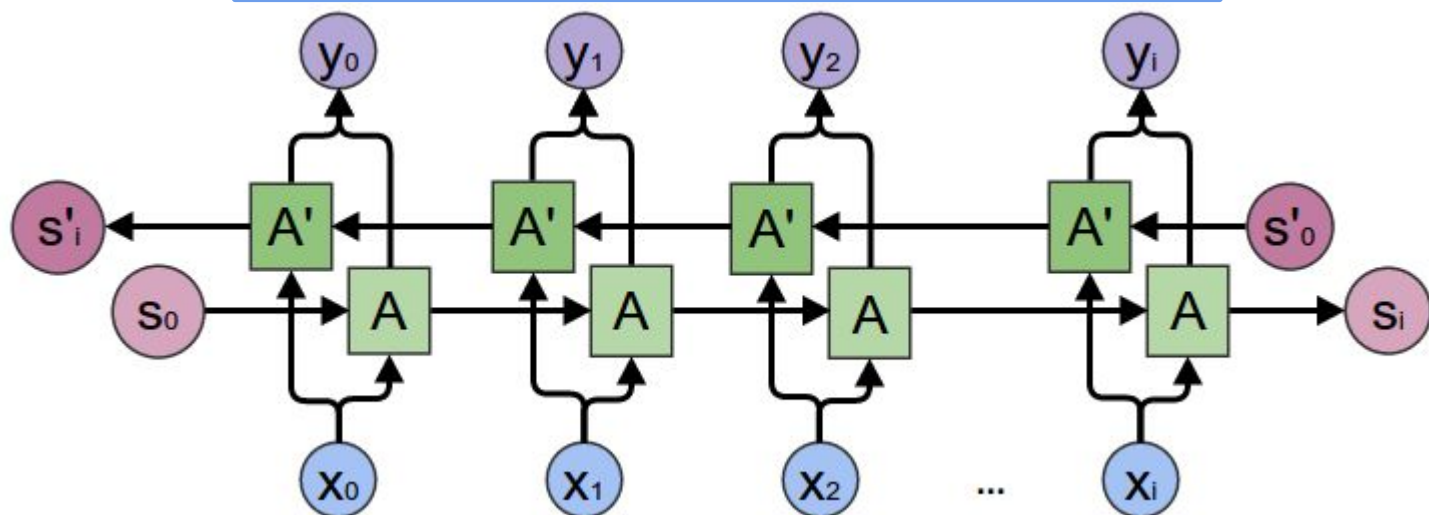
カナダ	の	モントリ オール	大学	教授	ヨシュア	・	ベンジオ	氏
B-LOC	O	B-ORG	E-ORG	O	B-PSN	M-PSN	E-PSN	O

# BLSTMs による固有表現抽出

- 入力文の各単語列を BLSTMs の入力として、固有表現タグを予測する

固有表現タグ

B-LOC      0      B-ORG    ...    0



各単語列

カナダ      の      モントリ  
                                 オール      ...      氏



# BLSTMs による固有表現抽出

- 固有表現の箇所に対応する、抽出すべきテキスト中の箇所をあらかじめタグづけした学習データを用意し、そのデータを機械学習アルゴリズムに与え、そのデータから抽出規則を自動的に学習する  
(コロナ社「自然言語処理の基礎」より)
- 固有表現タグの一覧
  - TIME, DATE, LOCATION, PERSON, MONEY, ARTIFACT  
PERCENT, ORGANIZATION, OPTIONAL
- BMEO によるチャンキング
  - **B**egin: 固有表現の開始単語とする (e.g. **B**-PSN)
  - **M**iddel: 固有表現の途中単語とする (e.g. **M**-PSN)
  - **E**nd: 固有表現の終了単語とする (e.g. **E**-PSN)
  - **O**: 固有表現でないことを示すタグ



# 事前準備

- 京都大学ウェブ文書リードコーパスのダウンロード
  - ウェブ文書の冒頭3文に言語情報を人手で付与したテキストコーパス
  - 言語情報としては、形態素・固有表現・構文等の情報を付与
  - <http://nlp.ist.i.kyoto-u.ac.jp/index.php?KWDLC> よりダウンロード
- Python ライブラリーのインストール



requirements.txt へのリンクは  
←ここをクリック

```
$ pip install nagisa  
$ pip install segeval  
$ pip install beautifulsoup4
```

# 京都大学ウェブ文書リードコーパス

- 各文書は KNP フォーマットで保存されている
  - KWDLC-1.0/dat/rel/w201106-00021201106-0002100303.KNP

```
# S-ID:w201106-0002100303-1 (Revision.8ce44af on 2014-08-14) JUMAN:8.0
(Revision.06f73dd on 2015-01-07) KNP:4.2-93fadae DATE:2015/07/04
SCORE:-11.27431 MOD:2015/11/02 MEMO:
* 0 3D
+ 0 4D <ne type="DATE" target="1981年"/>
1981 せんきゅうひゃくはちじゅういち * 名詞 数詞 * *
年 ねん * 接尾辞 名詞性名詞助数辞 * *
に に * 助詞 格助詞 * *
* 1 2D
+ 1 2D <ne type="LOCATION" target="ワシントン州"/>
ワシントン わしんとん * 名詞 地名 * *
州 しゅう * 接尾辞 名詞性特殊接尾辞 * *
+ 2 3D <rel type="ノ?" target="ワシントン州" sid="w201106-0002100303-1"
tag="1"/>
```

# 京都大学ウェブ文書リードコーパスの前処理

- モデル学習のため、KNP フォーマットをタブ区切りのファイルに変換する

```
カナダ    B-LOCATION
の        O
モントリオール    B-ORGANIZATION
大学      E-ORGANIZATION
教授      O
ヨシュア  B-PERSON
・         M-PERSON
ベンジオ  E-PERSON
氏        O
EOS
```



サンプルデータへのリンクは  
←ここをクリック

# 京都大学ウェブ文書リードコーパスの前処理

- モデル学習のため、KNP フォーマットをタブ区切りのファイルに変換する

## 入力単語列

```
カナダ    B-LOCATION
の        O
モントリオール    B-ORGANIZATION
大学      E-ORGANIZATION
教授      O
ヨシュア  B-PERSON
・        M-PERSON
ベンジオ  E-PERSON
氏        O
EOS
```

# 京都大学ウェブ文書リードコーパスの前処理

- モデル学習のため、KNP フォーマットをタブ区切りのファイルに変換する  
タブ(\t)

```
カナダ    B-LOCATION
の        O
モントリオール  B-ORGANIZATION
大学      E-ORGANIZATION
教授      O
ヨシュア  B-PERSON
・        M-PERSON
ベンジオ  E-PERSON
氏        O
EOS
```

# 京都大学ウェブ文書リードコーパスの前処理

- モデル学習のため、KNP フォーマットをタブ区切りのファイルに変換する

## 固有表現タグ列

```
カナダ    B-LOCATION
の        O
モントリオール    B-ORGANIZATION
大学      E-ORGANIZATION
教授      O
ヨシュア  B-PERSON
・        M-PERSON
ベンジオ  E-PERSON
氏        O
EOS
```

# 京都大学ウェブ文書リードコーパスの前処理

- モデル学習のため、KNP フォーマットをタブ区切りのファイルに変換する

```
カナダ    B-LOCATION
の        O
モントリオール    B-ORGANIZATION
大学      E-ORGANIZATION
教授      O
ヨシュア  B-PERSON
・        M-PERSON
ベンジオ  E-PERSON
氏        O
EOS
```

文末記号



# 前処理スクリプトの実行



Python コードへのリンクは  
←ここをクリック

## 1. 前処理スクリプトのダウンロードと作業ディレクトリへの移動

```
$ git clone https://github.com/taishi-i/nagisa-tutorial-pycon2019  
$ cd nagisa-tutorial-pycon2019/kwdlc_ner_tutorial
```

## 2. 解凍した京大ウェブ文書リードコーパス (KWDLC-1.0) を引数に指定し、preprocess\_kwdlc.py の実行 (タブ区切りのファイルへの変換と学習・開発・評価データの分割) を行う

```
$ python preprocess_kwdlc.py KWDLC-1.0
```

## 3. 出力ファイルの確認を行う

```
$ ls data  
kwdlc.txt kwdlc.train kwdlc.dev kwdlc.test
```

# 固有表現抽出モデルの学習



Python コードへのリンクは  
←ここをクリック

- 学習スクリプトの実行

```
$ python train_kwdlc_model.py
```

- train\_kwdlc\_model.py

```
import nagisa

nagisa.fit(
    train_file="data/kwdlc.train",
    dev_file="data/kwdlc.dev",
    test_file="data/kwdlc.test",
    model_name="data/kwdlc_ner_model"
)
```

# 固有表現抽出モデルの学習

- 学習スクリプトの実行

```
$ python train_kwdlc_model.py
```

- train\_kwdlc\_model.py

インポート

```
import nagisa
```

```
nagisa.fit(  
    train_file="data/kwdlc.train",  
    dev_file="data/kwdlc.dev",  
    test_file="data/kwdlc.test",  
    model_name="data/kwdlc_ner_model"  
)
```

# 固有表現抽出モデルの学習

- 学習スクリプトの実行

```
$ python train_kwdlc_model.py
```

- train\_kwdlc\_model.py

入力ファイルの指定

```
import nagisa

nagisa.fit(
    train_file="data/kwdlc.train",
    dev_file="data/kwdlc.dev",
    test_file="data/kwdlc.test",
    model_name="data/kwdlc_ner_model"
)
```



# 固有表現抽出モデルの学習

- 学習スクリプトの実行


```
$ python train_kwdlc_model.py
```

- train\_kwdlc\_model.py

出力モデル名の指定

```
import nagisa

nagisa.fit(
    train_file="data/kwdlc.train",
    dev_file="data/kwdlc.dev",
    test_file="data/kwdlc.test",
    model_name="data/kwdlc_ner_model"
)
```



# 固有表現抽出モデルの学習

- 学習スクリプトの実行

```
$ python train_kwdlc_model.py
```

- train\_kwdlc\_model.py

学習の実行

```
import nagisa

nagisa.fit(
    train_file="data/kwdlc.train",
    dev_file="data/kwdlc.dev",
    test_file="data/kwdlc.test",
    model_name="data/kwdlc_ner_model"
)
```

# 学習過程のログ

```
[dynet] random seed: 1234
[dynet] allocating memory: 32MB
[dynet] memory allocation done.
[nagisa] LAYERS: 1
[nagisa] THRESHOLD: 3
[nagisa] DECAY: 1
[nagisa] EPOCH: 10
...

```

Epoch	LR	Loss	Time_m	DevWS_f1	DevPOS_f1	TestWS_f1	TestPOS_f1
1	0.100	15.09	0.632	92.41	83.14	91.70	82.63
2	0.100	8.818	0.637	93.59	85.59	93.21	85.28
3	0.100	6.850	0.637	93.98	85.60	93.75	86.01
4	0.100	5.751	0.634	94.44	87.29	94.01	86.99
5	0.050	5.028	0.614	94.35	87.02	94.01	86.99
6	0.050	3.727	0.647	94.84	87.52	94.79	87.91
7	0.025	3.268	0.613	94.52	87.45	94.79	87.91
8	0.012	2.761	0.610	94.75	87.58	94.79	87.91
9	0.012	2.447	0.634	94.95	87.79	95.00	88.28
10	0.006	2.333	0.624	94.73	87.41	95.00	88.28

# 学習過程のログ

## ハイパーパラメータの一覧

```
[dynet] random seed: 1234
[dynet] allocating memory: 32MB
[dynet] memory allocation done.
[nagisa] LAYERS: 1
[nagisa] THRESHOLD: 3
[nagisa] DECAY: 1
[nagisa] EPOCH: 10
...
```

Epoch	LR	Loss	Time_m	DevWS_f1	DevPOS_f1	TestWS_f1	TestPOS_f1
1	0.100	15.09	0.632	92.41	83.14	91.70	82.63
2	0.100	8.818	0.637	93.59	85.59	93.21	85.28
3	0.100	6.850	0.637	93.98	85.60	93.75	86.01
4	0.100	5.751	0.634	94.44	87.29	94.01	86.99
5	0.050	5.028	0.614	94.35	87.02	94.01	86.99
6	0.050	3.727	0.647	94.84	87.52	94.79	87.91
7	0.025	3.268	0.613	94.52	87.45	94.79	87.91
8	0.012	2.761	0.610	94.75	87.58	94.79	87.91
9	0.012	2.447	0.634	94.95	87.79	95.00	88.28
10	0.006	2.333	0.624	94.73	87.41	95.00	88.28

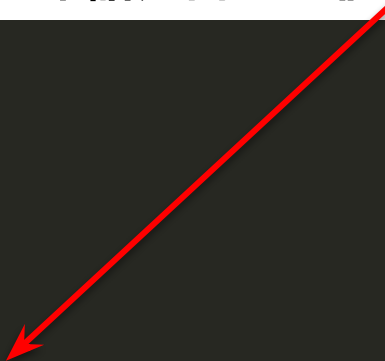


# 学習過程のログ

開発データに対する  
単語分割のF1値とタグ付けのF1値

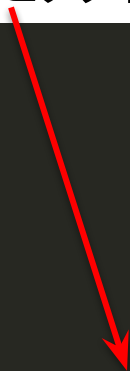
```
[dynet] random seed: 1234
[dynet] allocating memory: 32MB
[dynet] memory allocation done.
[nagisa] LAYERS: 1
[nagisa] THRESHOLD: 3
[nagisa] DECAY: 1
[nagisa] EPOCH: 10
...
```

Epoch	LR	Loss	Time_m	DevWS_f1	DevPOS_f1	TestWS_f1	TestPOS_f1
1	0.100	15.09	0.632	92.41	83.14	91.70	82.63
2	0.100	8.818	0.637	93.59	85.59	93.21	85.28
3	0.100	6.850	0.637	93.98	85.60	93.75	86.01
4	0.100	5.751	0.634	94.44	87.29	94.01	86.99
5	0.050	5.028	0.614	94.35	87.02	94.01	86.99
6	0.050	3.727	0.647	94.84	87.52	94.79	87.91
7	0.025	3.268	0.613	94.52	87.45	94.79	87.91
8	0.012	2.761	0.610	94.75	87.58	94.79	87.91
9	0.012	2.447	0.634	94.95	87.79	95.00	88.28
10	0.006	2.333	0.624	94.73	87.41	95.00	88.28



# 学習過程のログ

評価データに対する  
単語分割のF1値とタグ付けのF1値



```
[dynet] random seed: 1234
[dynet] allocating memory: 32MB
[dynet] memory allocation done.
[nagisa] LAYERS: 1
[nagisa] THRESHOLD: 3
[nagisa] DECAY: 1
[nagisa] EPOCH: 10
...
```

Epoch	LR	Loss	Time_m	DevWS_f1	DevPOS_f1	TestWS_f1	TestPOS_f1
1	0.100	15.09	0.632	92.41	83.14	91.70	82.63
2	0.100	8.818	0.637	93.59	85.59	93.21	85.28
3	0.100	6.850	0.637	93.98	85.60	93.75	86.01
4	0.100	5.751	0.634	94.44	87.29	94.01	86.99
5	0.050	5.028	0.614	94.35	87.02	94.01	86.99
6	0.050	3.727	0.647	94.84	87.52	94.79	87.91
7	0.025	3.268	0.613	94.52	87.45	94.79	87.91
8	0.012	2.761	0.610	94.75	87.58	94.79	87.91
9	0.012	2.447	0.634	94.95	87.79	95.00	88.28
10	0.006	2.333	0.624	94.73	87.41	95.00	88.28

# 学習済みモデルの利用方法



Python コードへのリンクは  
←ここをクリック

- モデルファイルをロードし、固有表現抽出モデルを利用する

```
>>> import nagisa

>>> ner_tagger = nagisa.Tagger(
    vocabs="data/kwdlc_ner_model.vocabs",
    params="data/kwdlc_ner_model.params",
    hp="data/kwdlc_ner_model.hp"
)

>>> text = "FacebookのAIラボ所長でもあるヤン・ルカン博士"
>>> tokens = ner_tagger.tagging(text)
>>> print(tokens)
Facebook/B-ORGANIZATION の/O AI/O ラボ/O 所長/O で/O も/O
ある/O ヤン/B-PERSON ・/M-PERSON ルカン/E-PERSON 博士/O
```

# 学習済みモデルの利用方法

- モデルファイルをロードし、固有表現抽出モデルを利用する

モデルファイルの指定

```
>>> import nagisa

>>> ner_tagger = nagisa.Tagger(
    vocabs="data/kwdlc_ner_model.vocabs",
    params="data/kwdlc_ner_model.params",
    hp="data/kwdlc_ner_model.hp"
)

>>> text = "FacebookのAIラボ所長でもあるヤン・ルカン博士"
>>> tokens = ner_tagger.tagging(text)
>>> print(tokens)
Facebook/B-ORGANIZATION の/O AI/O ラボ/O 所長/O で/O も/O
ある/O ヤン/B-PERSON ・/M-PERSON ルカン/E-PERSON 博士/O
```

# 学習済みモデルの利用方法

- モデルファイルをロードし、固有表現抽出モデルを利用する

**ner\_tagger の定義**

```
>>> import nagisa

>>> ner_tagger = nagisa.Tagger(
    vocabs="data/kwdlc_ner_model.vocabs",
    params="data/kwdlc_ner_model.params",
    hp="data/kwdlc_ner_model.hp"
)

>>> text = "FacebookのAIラボ所長でもあるヤン・ルカン博士"
>>> tokens = ner_tagger.tagging(text)
>>> print(tokens)
Facebook/B-ORGANIZATION の/O AI/O ラボ/O 所長/O で/O も/O
ある/O ヤン/B-PERSON ・/M-PERSON ルカン/E-PERSON 博士/O
```

# 学習済みモデルの利用方法

- モデルファイルをロードし、固有表現抽出モデルを利用する

入力テキスト

```
>>> import nagisa

>>> ner_tagger = nagisa.Tagger(
    vocabs="data/kwdlc_ner_model.vocabs",
    params="data/kwdlc_ner_model.params",
    hp="data/kwdlc_ner_model.hp"
)

>>> text = "FacebookのAIラボ所長でもあるヤン・ルカン博士"
>>> tokens = ner_tagger.tagging(text)
>>> print(tokens)
Facebook/B-ORGANIZATION の/O AI/O ラボ/O 所長/O で/O も/O
ある/O ヤン/B-PERSON ・/M-PERSON ルカン/E-PERSON 博士/O
```



# 学習済みモデルの利用方法

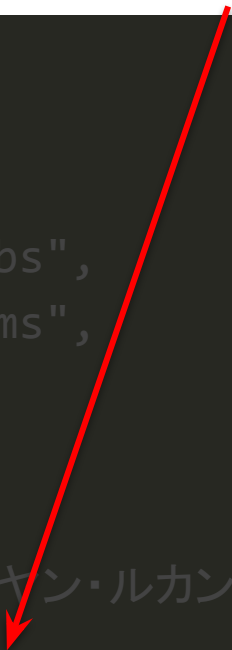
- モデルファイルをロードし、固有表現抽出モデルを利用する

解析の実行

```
>>> import nagisa

>>> ner_tagger = nagisa.Tagger(
    vocabs="data/kwdlc_ner_model.vocabs",
    params="data/kwdlc_ner_model.params",
    hp="data/kwdlc_ner_model.hp"
)

>>> text = "FacebookのAIラボ所長でもあるヤン・ルカン博士"
>>> tokens = ner_tagger.tagging(text)
>>> print(tokens)
Facebook/B-ORGANIZATION の/O AI/O ラボ/O 所長/O で/O も/O
ある/O ヤン/B-PERSON ・/M-PERSON ルカン/E-PERSON 博士/O
```



# 学習済みモデルの利用方法

- モデルファイルをロードし、固有表現抽出モデルを利用する

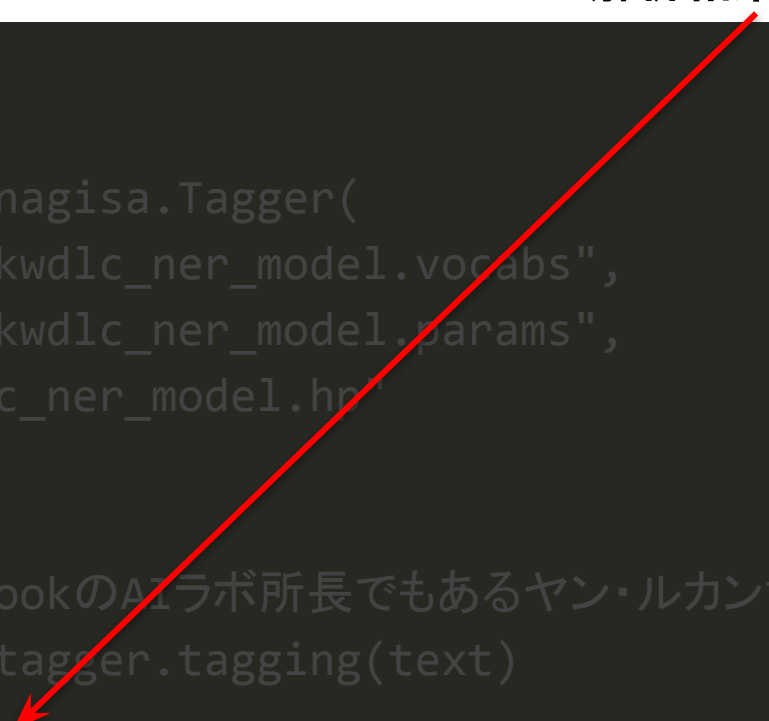
解析結果の出力

```
>>> import nagisa

>>> ner_tagger = nagisa.Tagger(
    vocabs="data/kwdlc_ner_model.vocabs",
    params="data/kwdlc_ner_model.params",
    hp="data/kwdlc_ner_model.hp"
)

>>> text = "FacebookのAIラボ所長でもあるヤン・ルカン博士"
>>> tokens = ner_tagger.tagging(text)
>>> print(tokens)
```

Facebook/B-ORGANIZATION の/O AI/O ラボ/O 所長/O で/O も/O  
ある/O ヤン/B-PERSON ・/M-PERSON ルカン/E-PERSON 博士/O





# 固有表現ごとの正解率の確認



Python コードへのリンクは  
←ここをクリック

- タグ単位の正解率ではなく、固有表現単位の正解率も確認する

	precision	recall	f1-score	support
LOCATION	0.64	0.67	0.66	127
DATE	0.84	0.88	0.86	78
ORGANIZATION	0.40	0.41	0.40	46
ARTIFACT	0.42	0.41	0.41	44
OPTIONAL	0.29	0.11	0.15	19
PERSON	0.59	0.75	0.66	51
MONEY	1.00	1.00	1.00	5
TIME	0.50	0.40	0.44	5
PERCENT	1.00	1.00	1.00	3
micro avg	0.62	0.64	0.63	378
macro avg	0.61	0.64	0.62	378

# 出力結果の確認によるエラー分析

- 正解率だけではなく、目視による出力結果の確認を

```
>>> text = "米Googleのエンジニアリングフェローも務めるジェフリー・  
ヒントン博士"  
>>> tokens = ner_tagger.tagging(text)  
>>> print(tokens)  
米/B-LOCATION Google/B-LOCATION の/O エンジニアリング/O フェ  
ロー/O も/O 務める/O ジェ/O フリー/O ・/O ヒントン/B-PERSON 博士  
/O
```

# 出力結果の確認によるエラー分析

- 正解率だけではなく、目視による出力結果の確認を

入力テキスト

```
>>> text = "米Googleのエンジニアリングフェローも務めるジェフリー・  
ヒントン博士"  
>>> tokens = ner_tagger.tagging(text)  
>>> print(tokens)  
米/B-LOCATION Google/B-LOCATION の/O エンジニアリング/O フェ  
ロー/O も/O 務める/O ジェ/O フリー/O ・/O ヒントン/B-PERSON 博士  
/O
```

# 出力結果の確認によるエラー分析

- 正解率だけではなく、目視による出力結果の確認を

## 解析結果の出力

```
>>> text = "米Googleのエンジニアリングフェローも務めるジェフリー・  
ヒントン博士"  
>>> tokens = ner_tagger.tagging(text)  
>>> print(tokens)  
米/B-LOCATION Google/B-LOCATION の/O エンジニアリング/O フェ  
ロー/O も/O 務める/O ジェ/O フリー/O ・/O ヒントン/B-PERSON 博士  
/O
```

# 出力結果の確認によるエラー分析

- 正解率だけではなく、目視による出力結果の確認を

タグ付け部分の解析ミス  
Google が LOCATION に

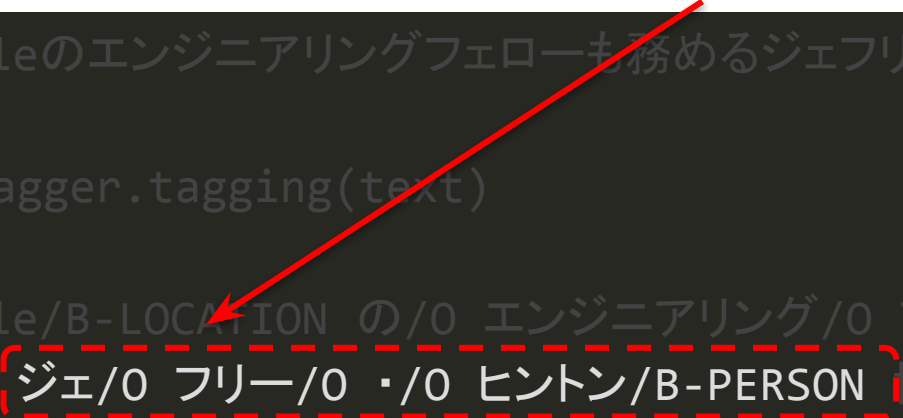
```
>>> text = "米Googleのエンジニアリングフェローも務めるジェフリー・  
ヒントン博士"  
>>> tokens = ner_tagger.tagging(text)  
>>> print(tokens)  
米/B-LOCATION Google/B-LOCATION の/O エンジニアリング/O フェ  
ロー/O も/O 務める/O ジェ/O フリー/O ・/O ヒントン/B-PERSON 博士  
/O
```

# 出力結果の確認によるエラー分析

- 正解率だけではなく、目視による出力結果の確認を

単語分割部分の解析ミス  
「ジェ」と「フリー」に

```
>>> text = "米Googleのエンジニアリングフェローを務めるジェフリー・  
ヒントン博士"  
>>> tokens = ner_tagger.tagging(text)  
>>> print(tokens)  
米/B-LOCATION Google/B-LOCATION の/O エンジニアリング/O フェ  
ロー/O も/O 務める/O ジェ/O フリー/O ・/O ヒントン/B-PERSON 博士  
/O
```



# 解析ミスへの対応方法

- 解析ミスを修正したデータを学習データに追加し、再学習を行う

```
米 B-LOCATION  
Google B-LOCATION  
の 0  
エンジニアリング 0  
フェロー 0  
も 0  
務める 0  
ジェ 0  
フリー 0  
・ 0  
hinton E-PERSON  
博士 0  
EOS
```

修正

修正

修正

```
米 B-LOCATION  
Google B-ORGANIZATION  
の 0  
エンジニアリング 0  
フェロー 0  
も 0  
務める 0  
ジェフリー B-PERSON  
・ M-PERSON  
hinton E-PERSON  
博士 0  
EOS
```

学習データに  
追加

# 系列ラベリングの応用例

- 系列ラベリングの利点
  - 辞書に存在しないパターンを抽出することができる
  - 文脈によって判断することができる
- サービス名抽出
  - 系列ラベリングは、新しいサービス名の抽出等に応用可能

入力文: 「Apple TV+」は日本でも月額600円で11月開始。



系列ラベリング

- サービス名: Apple TV+
- 料金: 月額600円
- 提供日: 11月開始



# おわりに

- nagisa を有効活用できる場面
  - 小規模なテキストデータ(~2万文)をサクッと分析したいとき
  - Colaboratory や heroku (無料枠) で自然言語処理をやりたいとき
  - 文書分類等の応用タスクで、品詞のフィルタリングやユーザー辞書追加による精度の影響を確認したいとき
  - 日本語を対象とした系列ラベリングのベースラインとして
- 本発表の内容や nagisa に関する質問等は GitHub で全て対応します
  - issues お待ちしております
  - <https://github.com/taishi-i/nagisa-tutorial-pycon2019>

# 参考リンク

- GitHub
  - <https://github.com/taishi-i/nagisa>
- nagisa: RNNによる日本語単語分割・品詞タグ付けツール
  - <https://qiita.com/taishi-i/items/5b9275a606b392f7f58e>
- Pythonで動く形態素解析ツール「nagisa」を使ってみた
  - <https://upura.hatenablog.com/entry/2018/09/18/203540>
- 形態素解析ツールのnagisa(なぎさ)を知っていますか？
  - <https://yolo.love/nlp/nagisa/>